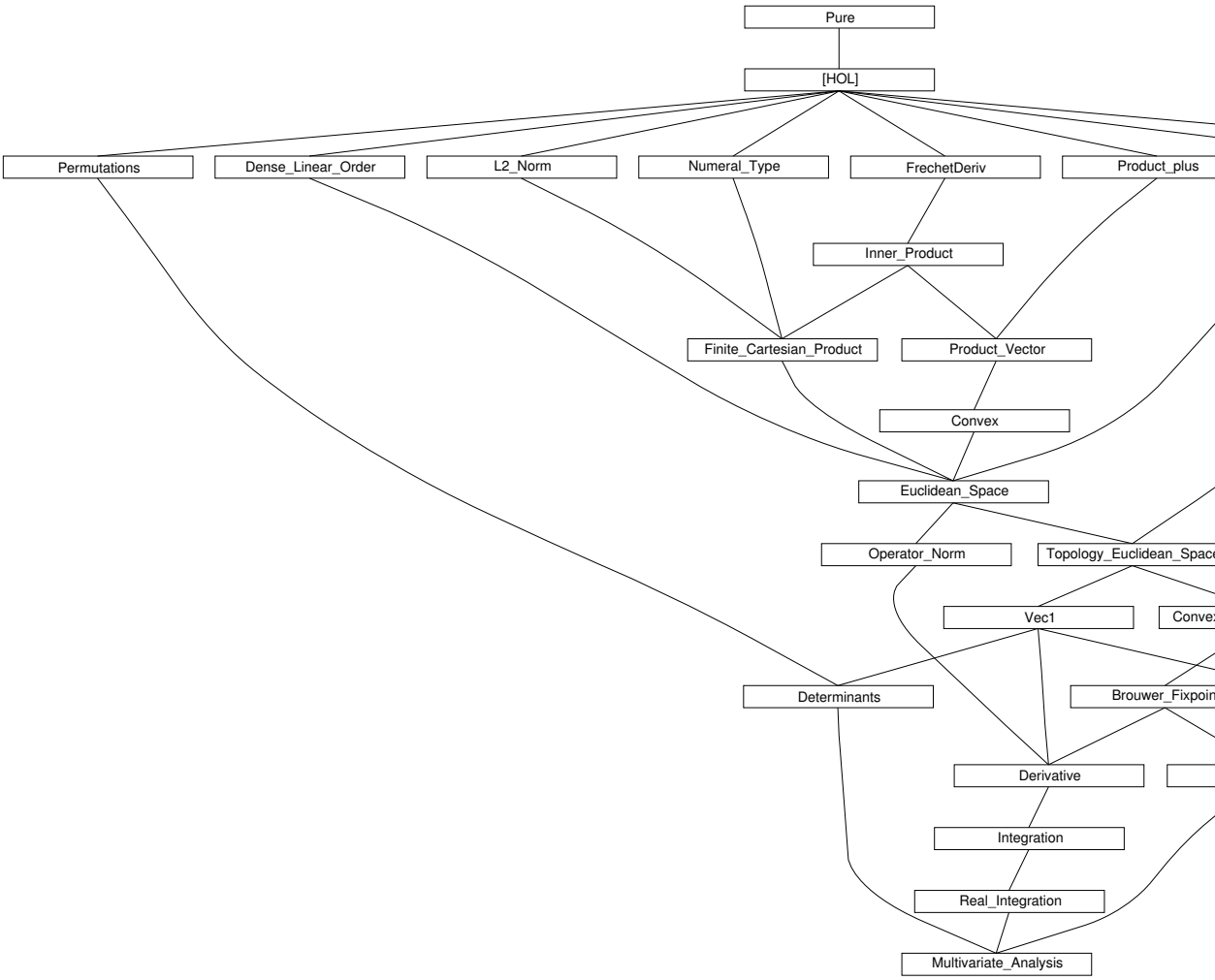
Multivariate Analysis

June 21, 2010

Contents



1 Dense-Linear-Order: Dense linear order without endpoints and a quantifier elimination procedure in Ferrante and Rackoff style

theory *Dense-Linear-Order*
imports *Main*
uses
langford-data.ML
ferrante-rackoff-data.ML
(*langford.ML*)
(*ferrante-rackoff.ML*)
begin

setup ⟪ *Langford-Data.setup #> Ferrante-Rackoff-Data.setup* ⟫

context *linorder*
begin

lemma *less-not-permute*[*no-atp*]: ¬ (*x* < *y* ∧ *y* < *x*) **by** (*simp add*: *not-less linear*)

lemma *gather-simps*[*no-atp*]:
shows
(∃ *x*. (∀ *y* ∈ *L*. *y* < *x*) ∧ (∀ *y* ∈ *U*. *x* < *y*) ∧ *x* < *u* ∧ *P x*) ⟷ (∃ *x*. (∀ *y* ∈ *L*. *y* < *x*) ∧ (∀ *y* ∈ (*insert u U*). *x* < *y*) ∧ *P x*)
and (∃ *x*. (∀ *y* ∈ *L*. *y* < *x*) ∧ (∀ *y* ∈ *U*. *x* < *y*) ∧ *l* < *x* ∧ *P x*) ⟷ (∃ *x*. (∀ *y* ∈ (*insert l L*). *y* < *x*) ∧ (∀ *y* ∈ *U*. *x* < *y*) ∧ *P x*)
(∃ *x*. (∀ *y* ∈ *L*. *y* < *x*) ∧ (∀ *y* ∈ *U*. *x* < *y*) ∧ *x* < *u*) ⟷ (∃ *x*. (∀ *y* ∈ *L*. *y* < *x*) ∧ (∀ *y* ∈ (*insert u U*). *x* < *y*))
and (∃ *x*. (∀ *y* ∈ *L*. *y* < *x*) ∧ (∀ *y* ∈ *U*. *x* < *y*) ∧ *l* < *x*) ⟷ (∃ *x*. (∀ *y* ∈ (*insert l L*). *y* < *x*) ∧ (∀ *y* ∈ *U*. *x* < *y*)) **by** *auto*

lemma
gather-start[*no-atp*]: (∃ *x*. *P x*) ≡ (∃ *x*. (∀ *y* ∈ {}. *y* < *x*) ∧ (∀ *y*∈ {}. *x* < *y*) ∧ *P x*)
by *simp*

Theorems for ∃ *z*. ∀ *x*. *x* < *z* ⟶ (*P x* ⟷ $P_{-\infty}$)

lemma *minf-lt*[*no-atp*]: ∃ *z* . ∀ *x*. *x* < *z* ⟶ (*x* < *t* ⟷ *True*) **by** *auto*
lemma *minf-gt*[*no-atp*]: ∃ *z* . ∀ *x*. *x* < *z* ⟶ (*t* < *x* ⟷ *False*)
by (*simp add*: *not-less*) (*rule exI*[**where** *x*=*t*], *auto simp add*: *less-le*)

lemma *minf-le*[*no-atp*]: ∃ *z*. ∀ *x*. *x* < *z* ⟶ (*x* ≤ *t* ⟷ *True*) **by** (*auto simp add*: *less-le*)
lemma *minf-ge*[*no-atp*]: ∃ *z*. ∀ *x*. *x* < *z* ⟶ (*t* ≤ *x* ⟷ *False*)
by (*auto simp add*: *less-le not-less not-le*)
lemma *minf-eq*[*no-atp*]: ∃ *z*. ∀ *x*. *x* < *z* ⟶ (*x* = *t* ⟷ *False*) **by** *auto*
lemma *minf-neq*[*no-atp*]: ∃ *z*. ∀ *x*. *x* < *z* ⟶ (*x* ≠ *t* ⟷ *True*) **by** *auto*
lemma *minf-P*[*no-atp*]: ∃ *z*. ∀ *x*. *x* < *z* ⟶ (*P* ⟷ *P*) **by** *blast*

Theorems for ∃ *z*. ∀ *x*. *x* < *z* ⟶ (*P x* ⟷ $P_{+\infty}$)

lemma *pinf-gt*[*no-atp*]: ∃ *z* . ∀ *x*. *z* < *x* ⟶ (*t* < *x* ⟷ *True*) **by** *auto*
lemma *pinf-lt*[*no-atp*]: ∃ *z* . ∀ *x*. *z* < *x* ⟶ (*x* < *t* ⟷ *False*)
by (*simp add*: *not-less*) (*rule exI*[**where** *x*=*t*], *auto simp add*: *less-le*)

lemma *pinf-ge*[*no-atp*]: ∃ *z*. ∀ *x*. *z* < *x* ⟶ (*t* ≤ *x* ⟷ *True*) **by** (*auto simp add*: *less-le*)
lemma *pinf-le*[*no-atp*]: ∃ *z*. ∀ *x*. *z* < *x* ⟶ (*x* ≤ *t* ⟷ *False*)
by (*auto simp add*: *less-le not-less not-le*)
lemma *pinf-eq*[*no-atp*]: ∃ *z*. ∀ *x*. *z* < *x* ⟶ (*x* = *t* ⟷ *False*) **by** *auto*
lemma *pinf-neq*[*no-atp*]: ∃ *z*. ∀ *x*. *z* < *x* ⟶ (*x* ≠ *t* ⟷ *True*) **by** *auto*
lemma *pinf-P*[*no-atp*]: ∃ *z*. ∀ *x*. *z* < *x* ⟶ (*P* ⟷ *P*) **by** *blast*

lemma *nmi-lt*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg True \wedge x < t \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)$ **by** *auto*
lemma *nmi-gt*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg False \wedge t < x \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)$
by (*auto simp add*: *le-less*)
lemma *nmi-le*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg True \wedge x{\leq}\ t \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)$ **by** *auto*
lemma *nmi-ge*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg False \wedge t{\leq}\ x \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)$ **by** *auto*
lemma *nmi-eq*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg False \wedge\ x = t \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)$ **by** *auto*
lemma *nmi-neq*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg True \wedge x \neq t \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)$ **by** *auto*
lemma *nmi-P*[*no-atp*]: $\forall\ x.\ {\sim}P \wedge P \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)$ **by** *auto*
lemma *nmi-conj*[*no-atp*]: $[\![\forall x.\ \neg P1' \wedge P1\ x \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)\ ;$
$\forall x.\ \neg P2' \wedge P2\ x \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)]\!] \Longrightarrow$
$\forall x.\ \neg(P1' \wedge P2') \wedge (P1\ x \wedge P2\ x) \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)$ **by** *auto*
lemma *nmi-disj*[*no-atp*]: $[\![\forall x.\ \neg P1' \wedge P1\ x \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)\ ;$
$\forall x.\ \neg P2' \wedge P2\ x \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)]\!] \Longrightarrow$
$\forall x.\ \neg(P1' \vee P2') \wedge (P1\ x \vee P2\ x) \longrightarrow (\exists\ u{\in}\ U.\ u \leq x)$ **by** *auto*

lemma *npi-lt*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg False \wedge\ x < t \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)$ **by** (*auto simp add*: *le-less*)
lemma *npi-gt*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg True \wedge t < x \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)$ **by** *auto*
lemma *npi-le*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg False \wedge\ x \leq t \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)$ **by** *auto*
lemma *npi-ge*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg True \wedge t \leq x \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)$ **by** *auto*
lemma *npi-eq*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg False \wedge\ x = t \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)$ **by** *auto*
lemma *npi-neq*[*no-atp*]: $t \in U \Longrightarrow \forall x.\ \neg True \wedge x \neq t \longrightarrow (\exists\ u{\in}\ U.\ x \leq u\ )$ **by** *auto*
lemma *npi-P*[*no-atp*]: $\forall\ x.\ {\sim}P \wedge P \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)$ **by** *auto*
lemma *npi-conj*[*no-atp*]: $[\![\forall x.\ \neg P1' \wedge P1\ x \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)\ ;\ \forall x.\ \neg P2' \wedge P2\ x \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)]\!]$
$\Longrightarrow \forall x.\ \neg(P1' \wedge P2') \wedge (P1\ x \wedge P2\ x) \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)$ **by** *auto*
lemma *npi-disj*[*no-atp*]: $[\![\forall x.\ \neg P1' \wedge P1\ x \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)\ ; \forall x.\ \neg P2' \wedge P2\ x \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)]\!]$
$\Longrightarrow \forall x.\ \neg(P1' \vee P2') \wedge (P1\ x \vee P2\ x) \longrightarrow (\exists\ u{\in}\ U.\ x \leq u)$ **by** *auto*

lemma *lin-dense-lt*[*no-atp*]: $t \in U \Longrightarrow \forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l{<}\ x \wedge x < u \wedge x < t \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow y < t)$
proof(*clarsimp*)
fix *x l u y* **assume** *tU*: $t \in U$ **and** *noU*: $\forall t.\ l < t \wedge t < u \longrightarrow t \notin U$ **and** *lx*: $l < x$
and *xu*: $x{<}u$ **and** *px*: $x < t$ **and** *ly*: $l{<}y$ **and** *yu*:$y < u$
from *tU noU ly yu* **have** *tny*: $t{\neq}y$ **by** *auto*
{assume *H*: $t < y$
from *less-trans*[*OF lx px*] *less-trans*[*OF H yu*]

have $l < t \wedge t < u$ **by** *simp*
with *tU noU* **have** *False* **by** *auto*}
hence $\neg\ t < y$ **by** *auto* **hence** $y \leq t$ **by** (*simp add*: *not-less*)
thus $y < t$ **using** *tny* **by** (*simp add*: *less-le*)
qed

lemma *lin-dense-gt*[*no-atp*]: $t \in U \Longrightarrow \forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge t < x \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow t < y)$
proof(*clarsimp*)
fix *x l u y*
assume *tU*: $t \in U$ **and** *noU*: $\forall t.\ l < t \wedge t < u \longrightarrow t \notin U$ **and** *lx*: $l < x$ **and** *xu*: $x<u$
and *px*: $t < x$ **and** *ly*: $l<y$ **and** *yu*:$y < u$
from *tU noU ly yu* **have** *tny*: $t\neq y$ **by** *auto*
{**assume** *H*: $y< t$
from *less-trans*[*OF ly H*] *less-trans*[*OF px xu*] **have** $l < t \wedge t < u$ **by** *simp*
with *tU noU* **have** *False* **by** *auto*}
hence $\neg\ y<t$ **by** *auto* **hence** $t \leq y$ **by** (*auto simp add*: *not-less*)
thus $t < y$ **using** *tny* **by** (*simp add*:*less-le*)
qed

lemma *lin-dense-le*[*no-atp*]: $t \in U \Longrightarrow \forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t< u \longrightarrow t \notin U) \wedge l< x \wedge x < u \wedge x \leq t \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow y\leq t)$
proof(*clarsimp*)
fix *x l u y*
assume *tU*: $t \in U$ **and** *noU*: $\forall t.\ l < t \wedge t < u \longrightarrow t \notin U$ **and** *lx*: $l < x$ **and** *xu*: $x<u$
and *px*: $x \leq t$ **and** *ly*: $l<y$ **and** *yu*:$y < u$
from *tU noU ly yu* **have** *tny*: $t\neq y$ **by** *auto*
{**assume** *H*: $t < y$
from *less-le-trans*[*OF lx px*] *less-trans*[*OF H yu*]
have $l < t \wedge t < u$ **by** *simp*
with *tU noU* **have** *False* **by** *auto*}
hence $\neg\ t < y$ **by** *auto* **thus** $y \leq t$ **by** (*simp add*: *not-less*)
qed

lemma *lin-dense-ge*[*no-atp*]: $t \in U \Longrightarrow \forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t< u \longrightarrow t \notin U) \wedge l< x \wedge x < u \wedge t \leq x \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow t \leq y)$
proof(*clarsimp*)
fix *x l u y*
assume *tU*: $t \in U$ **and** *noU*: $\forall t.\ l < t \wedge t < u \longrightarrow t \notin U$ **and** *lx*: $l < x$ **and** *xu*: $x<u$
and *px*: $t \leq x$ **and** *ly*: $l<y$ **and** *yu*:$y < u$
from *tU noU ly yu* **have** *tny*: $t\neq y$ **by** *auto*
{**assume** *H*: $y< t$
from *less-trans*[*OF ly H*] *le-less-trans*[*OF px xu*]
have $l < t \wedge t < u$ **by** *simp*
with *tU noU* **have** *False* **by** *auto*}
hence $\neg\ y<t$ **by** *auto* **thus** $t \leq y$ **by** (*simp add*: *not-less*)

qed
lemma *lin-dense-eq*[*no-atp*]: $t \in U \Longrightarrow \forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge x = t \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow y = t)$ **by** *auto*
lemma *lin-dense-neq*[*no-atp*]: $t \in U \Longrightarrow \forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge x \neq t \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow y \neq t)$ **by** *auto*
lemma *lin-dense-P*[*no-atp*]: $\forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge P \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow P)$ **by** *auto*

lemma *lin-dense-conj*[*no-atp*]:
$[\![\forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge P1\ x \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow P1\ y)\ ;$
$\forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge P2\ x \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow P2\ y)]\!] \Longrightarrow$
$\forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge (P1\ x \wedge P2\ x) \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow (P1\ y \wedge P2\ y))$
by *blast*
lemma *lin-dense-disj*[*no-atp*]:
$[\![\forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge P1\ x \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow P1\ y)\ ;$
$\forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge P2\ x \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow P2\ y)]\!] \Longrightarrow$
$\forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge (P1\ x \vee P2\ x) \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow (P1\ y \vee P2\ y))$
by *blast*

lemma *npmibnd*[*no-atp*]: $[\![\forall x.\ \neg\ MP \wedge P\ x \longrightarrow (\exists\ u \in U.\ u \leq x);\ \forall x.\ \neg PP \wedge P\ x \longrightarrow (\exists\ u \in U.\ x \leq u)]\!]$
$\Longrightarrow \forall x.\ \neg\ MP \wedge \neg PP \wedge P\ x \longrightarrow (\exists\ u \in U.\ \exists\ u' \in U.\ u \leq x \wedge x \leq u')$
by *auto*

lemma *finite-set-intervals*[*no-atp*]:
assumes *px*: $P\ x$ **and** *lx*: $l \leq x$ **and** *xu*: $x \leq u$ **and** *linS*: $l \in S$
and *uinS*: $u \in S$ **and** *fS*:*finite S* **and** *lS*: $\forall\ x \in S.\ l \leq x$ **and** *Su*: $\forall\ x \in S.\ x \leq u$
shows $\exists\ a \in S.\ \exists\ b \in S.\ (\forall\ y.\ a < y \wedge y < b \longrightarrow y \notin S) \wedge a \leq x \wedge x \leq b \wedge P\ x$
proof−
let *?Mx* $= \{y.\ y \in S \wedge y \leq x\}$
let *?xM* $= \{y.\ y \in S \wedge x \leq y\}$
let *?a* $=$ *Max ?Mx*
let *?b* $=$ *Min ?xM*
have *MxS*: $?Mx \subseteq S$ **by** *blast*
hence *fMx*: *finite ?Mx* **using** *fS finite-subset* **by** *auto*
from *lx linS* **have** *linMx*: $l \in ?Mx$ **by** *blast*
hence *Mxne*: $?Mx \neq \{\}$ **by** *blast*
have *xMS*: $?xM \subseteq S$ **by** *blast*
hence *fxM*: *finite ?xM* **using** *fS finite-subset* **by** *auto*
from *xu uinS* **have** *linxM*: $u \in ?xM$ **by** *blast*
hence *xMne*: $?xM \neq \{\}$ **by** *blast*

have *ax*:*?a* ≤ *x* **using** *Mxne fMx* **by** *auto*
have *xb*:*x* ≤ *?b* **using** *xMne fxM* **by** *auto*
have *?a* ∈ *?Mx* **using** *Max-in*[*OF fMx Mxne*] **by** *simp* **hence** *ainS*: *?a* ∈ *S* **using** *MxS* **by** *blast*
have *?b* ∈ *?xM* **using** *Min-in*[*OF fxM xMne*] **by** *simp* **hence** *binS*: *?b* ∈ *S* **using** *xMS* **by** *blast*
have *noy*:∀ *y*. *?a* < *y* ∧ *y* < *?b* ⟶ *y* ∉ *S*
proof(*clarsimp*)
fix *y* **assume** *ay*: *?a* < *y* **and** *yb*: *y* < *?b* **and** *yS*: *y* ∈ *S*
from *yS* **have** *y*∈ *?Mx* ∨ *y*∈ *?xM* **by** (*auto simp add*: *linear*)
moreover {assume *y* ∈ *?Mx* **hence** *y* ≤ *?a* **using** *Mxne fMx* **by** *auto* **with** *ay* **have** *False* **by** (*simp add*: *not-le*[*symmetric*])**}**
moreover {assume *y* ∈ *?xM* **hence** *?b* ≤ *y* **using** *xMne fxM* **by** *auto* **with** *yb* **have** *False* **by** (*simp add*: *not-le*[*symmetric*])**}**
ultimately show *False* **by** *blast*
qed
from *ainS binS noy ax xb px* **show** *?thesis* **by** *blast*
qed

lemma *finite-set-intervals2*[*no-atp*]:
assumes *px*: *P x* **and** *lx*: *l* ≤ *x* **and** *xu*: *x* ≤ *u* **and** *linS*: *l*∈ *S*
and *uinS*: *u* ∈ *S* **and** *fS*:*finite S* **and** *lS*: ∀ *x*∈ *S*. *l* ≤ *x* **and** *Su*: ∀ *x*∈ *S*. *x* ≤ *u*
shows (∃ *s*∈ *S*. *P s*) ∨ (∃ *a* ∈ *S*. ∃ *b* ∈ *S*. (∀ *y*. *a* < *y* ∧ *y* < *b* ⟶ *y* ∉ *S*) ∧ *a* < *x* ∧ *x* < *b* ∧ *P x*)
proof−
from *finite-set-intervals*[**where** *P*=*P*, *OF px lx xu linS uinS fS lS Su*]
obtain *a* **and** *b* **where**
as: *a*∈ *S* **and** *bs*: *b*∈ *S* **and** *noS*:∀ *y*. *a* < *y* ∧ *y* < *b* ⟶ *y* ∉ *S*
and *axb*: *a* ≤ *x* ∧ *x* ≤ *b* ∧ *P x* **by** *auto*
from *axb* **have** *x*= *a* ∨ *x*= *b* ∨ (*a* < *x* ∧ *x* < *b*) **by** (*auto simp add*: *le-less*)
thus *?thesis* **using** *px as bs noS* **by** *blast*
qed

end

2 The classical QE after Langford for dense linear orders

context *dense-linorder*
begin

lemma *interval-empty-iff*:
{*y*. *x* < *y* ∧ *y* < *z*} = {} ⟷ ¬ *x* < *z*
by (*auto dest*: *dense*)

lemma *dlo-qe-bnds*[*no-atp*]:
assumes *ne*: *L* ≠ {} **and** *neU*: *U* ≠ {} **and** *fL*: *finite L* **and** *fU*: *finite U*

shows $(\exists x.\ (\forall y \in L.\ y < x) \wedge (\forall y \in U.\ x < y)) \equiv (\forall\ l \in L.\ \forall u \in U.\ l < u)$
proof (*simp only*: *atomize-eq*, *rule iffI*)
assume *H*: $\exists x.\ (\forall y{\in}L.\ y < x) \wedge (\forall y{\in}U.\ x < y)$
then obtain *x* **where** *xL*: $\forall y{\in}L.\ y < x$ **and** *xU*: $\forall y{\in}U.\ x < y$ **by** *blast*
{fix *l u* **assume** *l*: $l \in L$ **and** *u*: $u \in U$
have $l < x$ **using** *xL l* **by** *blast*
also have $x < u$ **using** *xU u* **by** *blast*
finally (*less-trans*) **have** $l < u$ **.}**
thus $\forall l{\in}L.\ \forall u{\in}U.\ l < u$ **by** *blast*
next
assume *H*: $\forall l{\in}L.\ \forall u{\in}U.\ l < u$
let *?ML* = *Max L*
let *?MU* = *Min U*
from *fL ne* **have** *th1*: $?ML \in L$ **and** *th1'*: $\forall l{\in}L.\ l \leq ?ML$ **by** *auto*
from *fU neU* **have** *th2*: $?MU \in U$ **and** *th2'*: $\forall u{\in}U.\ ?MU \leq u$ **by** *auto*
from *th1 th2 H* **have** $?ML < ?MU$ **by** *auto*
with *dense* **obtain** *w* **where** *th3*: $?ML < w$ **and** *th4*: $w < ?MU$ **by** *blast*
from *th3 th1'* **have** $\forall l \in L.\ l < w$ **by** *auto*
moreover from *th4 th2'* **have** $\forall u \in U.\ w < u$ **by** *auto*
ultimately show $\exists x.\ (\forall y{\in}L.\ y < x) \wedge (\forall y{\in}U.\ x < y)$ **by** *auto*
qed

lemma *dlo-qe-noub*[*no-atp*]:
assumes *ne*: $L \neq \{\}$ **and** *fL*: *finite L*
shows $(\exists x.\ (\forall y \in L.\ y < x) \wedge (\forall y \in \{\}.\ x < y)) \equiv True$
proof(*simp add*: *atomize-eq*)
from *gt-ex*[*of Max L*] **obtain** *M* **where** *M*: $Max\ L < M$ **by** *blast*
from *ne fL* **have** $\forall x \in L.\ x \leq Max\ L$ **by** *simp*
with *M* **have** $\forall x{\in}L.\ x < M$ **by** (*auto intro*: *le-less-trans*)
thus $\exists x.\ \forall y{\in}L.\ y < x$ **by** *blast*
qed

lemma *dlo-qe-nolb*[*no-atp*]:
assumes *ne*: $U \neq \{\}$ **and** *fU*: *finite U*
shows $(\exists x.\ (\forall y \in \{\}.\ y < x) \wedge (\forall y \in U.\ x < y)) \equiv True$
proof(*simp add*: *atomize-eq*)
from *lt-ex*[*of Min U*] **obtain** *M* **where** *M*: $M < Min\ U$ **by** *blast*
from *ne fU* **have** $\forall x \in U.\ Min\ U \leq x$ **by** *simp*
with *M* **have** $\forall x{\in}U.\ M < x$ **by** (*auto intro*: *less-le-trans*)
thus $\exists x.\ \forall y{\in}U.\ x < y$ **by** *blast*
qed

lemma *exists-neq*[*no-atp*]: $\exists (x::'a).\ x \neq t\ \exists (x::'a).\ t \neq x$
using *gt-ex*[*of t*] **by** *auto*

lemmas *dlo-simps*[*no-atp*] = *order-refl less-irrefl not-less not-le exists-neq*
le-less neq-iff linear less-not-permute

lemma *axiom*[*no-atp*]: *class.dense-linorder* (*op* $\leq$) (*op* $<$) **by** (*rule dense-linorder-axioms*)

lemma *atoms*[*no-atp*]:
  shows *TERM* (*less* :: *'a* ⇒ -)
    and *TERM* (*less-eq* :: *'a* ⇒ -)
    and *TERM* (*op* = :: *'a* ⇒ -) **.**

declare *axiom*[*langford qe*: *dlo-qe-bnds dlo-qe-nolb dlo-qe-noub gather*: *gather-start gather-simps atoms*: *atoms*]
declare *dlo-simps*[*langfordsimp*]

end

lemma *dnf*[*no-atp*]:
  (*P* & (*Q* | *R*)) = ((*P*&*Q*) | (*P*&*R*))
  ((*Q* | *R*) & *P*) = ((*Q*&*P*) | (*R*&*P*))
  by *blast+*

lemmas *weak-dnf-simps*[*no-atp*] = *simp-thms dnf*

lemma *nnf-simps*[*no-atp*]:
    (¬(*P* ∧ *Q*)) = (¬*P* ∨ ¬*Q*) (¬(*P* ∨ *Q*)) = (¬*P* ∧ ¬*Q*) (*P* ⟶ *Q*) = (¬*P* ∨ *Q*)
    (*P* = *Q*) = ((*P* ∧ *Q*) ∨ (¬*P* ∧ ¬ *Q*)) (¬ ¬(*P*)) = *P*
  by *blast+*

lemma *ex-distrib*[*no-atp*]: (∃ *x*. *P x* ∨ *Q x*) ⟷ ((∃ *x*. *P x*) ∨ (∃ *x*. *Q x*)) **by** *blast*

lemmas *dnf-simps*[*no-atp*] = *weak-dnf-simps nnf-simps ex-distrib*

use *langford.ML*
method-setup *dlo* = ⟪
  Scan.succeed (*SIMPLE-METHOD' o LangfordQE.dlo-tac*)
⟫ *Langford's algorithm for quantifier elimination in dense linear orders*

3 Contructive dense linear orders yield QE for linear arithmetic over ordered Fields

Linear order without upper bounds

locale *linorder-stupid-syntax* = *linorder*
begin
notation
  less-eq (*op* ⊑) **and**
  less-eq ((-/ ⊑ -) [*51*, *51*] *50*) **and**
  less (*op* ⊏) **and**
  less ((-/ ⊏ -) [*51*, *51*] *50*)

end

locale *linorder-no-ub* = *linorder-stupid-syntax* +

assumes *gt-ex*: $\exists\, y.\ less\ x\ y$
begin
lemma *ge-ex*[*no-atp*]: $\exists\, y.\ x \sqsubseteq y$ **using** *gt-ex* **by** *auto*

Theorems for $\exists\, z.\ \forall\, x.\ z \sqsubset x \longrightarrow (P\ x \longleftrightarrow P_{+\infty})$

lemma *pinf-conj*[*no-atp*]:
assumes *ex1*: $\exists\, z1.\ \forall\, x.\ z1 \sqsubset x \longrightarrow (P1\ x \longleftrightarrow P1')$
and *ex2*: $\exists\, z2.\ \forall\, x.\ z2 \sqsubset x \longrightarrow (P2\ x \longleftrightarrow P2')$
shows $\exists\, z.\ \forall\, x.\ z \sqsubset\ x \longrightarrow ((P1\ x \wedge P2\ x) \longleftrightarrow (P1' \wedge P2'))$
proof−
from *ex1 ex2* **obtain** *z1* **and** *z2* **where** *z1*: $\forall\, x.\ z1 \sqsubset x \longrightarrow (P1\ x \longleftrightarrow P1')$
and *z2*: $\forall\, x.\ z2 \sqsubset x \longrightarrow (P2\ x \longleftrightarrow P2')$ **by** *blast*
from *gt-ex* **obtain** *z* **where** *z*:*ord.max less-eq z1 z2* $\sqsubset z$ **by** *blast*
from *z* **have** *zz1*: $z1 \sqsubset z$ **and** *zz2*: $z2 \sqsubset z$ **by** *simp-all*
{fix *x* **assume** *H*: $z \sqsubset x$
from *less-trans*[*OF zz1 H*] *less-trans*[*OF zz2 H*]
have $(P1\ x \wedge P2\ x) \longleftrightarrow (P1' \wedge P2')$ **using** *z1 zz1 z2 zz2* **by** *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *pinf-disj*[*no-atp*]:
assumes *ex1*: $\exists\, z1.\ \forall\, x.\ z1 \sqsubset x \longrightarrow (P1\ x \longleftrightarrow P1')$
and *ex2*: $\exists\, z2.\ \forall\, x.\ z2 \sqsubset x \longrightarrow (P2\ x \longleftrightarrow P2')$
shows $\exists\, z.\ \forall\, x.\ z \sqsubset\ x \longrightarrow ((P1\ x \vee P2\ x) \longleftrightarrow (P1' \vee P2'))$
proof−
from *ex1 ex2* **obtain** *z1* **and** *z2* **where** *z1*: $\forall\, x.\ z1 \sqsubset x \longrightarrow (P1\ x \longleftrightarrow P1')$
and *z2*: $\forall\, x.\ z2 \sqsubset x \longrightarrow (P2\ x \longleftrightarrow P2')$ **by** *blast*
from *gt-ex* **obtain** *z* **where** *z*:*ord.max less-eq z1 z2* $\sqsubset z$ **by** *blast*
from *z* **have** *zz1*: $z1 \sqsubset z$ **and** *zz2*: $z2 \sqsubset z$ **by** *simp-all*
{fix *x* **assume** *H*: $z \sqsubset x$
from *less-trans*[*OF zz1 H*] *less-trans*[*OF zz2 H*]
have $(P1\ x \vee P2\ x) \longleftrightarrow (P1' \vee P2')$ **using** *z1 zz1 z2 zz2* **by** *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *pinf-ex*[*no-atp*]: **assumes** *ex*:$\exists\, z.\ \forall\, x.\ z \sqsubset x \longrightarrow (P\ x \longleftrightarrow P1)$ **and** *p1*: *P1* **shows** $\exists\ x.\ P\ x$
proof−
from *ex* **obtain** *z* **where** *z*: $\forall\, x.\ z \sqsubset x \longrightarrow (P\ x \longleftrightarrow P1)$ **by** *blast*
from *gt-ex* **obtain** *x* **where** *x*: $z \sqsubset x$ **by** *blast*
from *z x p1* **show** *?thesis* **by** *blast*
qed

end

Linear order without upper bounds

locale *linorder-no-lb* = *linorder-stupid-syntax* +

assumes *lt-ex*: $\exists\, y.\ less\ y\ x$
begin
lemma *le-ex*[*no-atp*]: $\exists\, y.\ y \sqsubseteq x$ **using** *lt-ex* **by** *auto*

Theorems for $\exists\, z.\ \forall\, x.\ x \sqsubset z \longrightarrow (P\ x \longleftrightarrow P_{-\infty})$

lemma *minf-conj*[*no-atp*]:
assumes *ex1*: $\exists\, z1.\ \forall\, x.\ x \sqsubset z1 \longrightarrow (P1\ x \longleftrightarrow P1')$
and *ex2*: $\exists\, z2.\ \forall\, x.\ x \sqsubset z2 \longrightarrow (P2\ x \longleftrightarrow P2')$
shows $\exists\, z.\ \forall\, x.\ x \sqsubset\ z \longrightarrow ((P1\ x \wedge P2\ x) \longleftrightarrow (P1' \wedge P2'))$
proof−
from *ex1 ex2* **obtain** *z1* **and** *z2* **where** *z1*: $\forall\, x.\ x \sqsubset z1 \longrightarrow (P1\ x \longleftrightarrow P1')$**and** *z2*: $\forall\, x.\ x \sqsubset z2 \longrightarrow (P2\ x \longleftrightarrow P2')$ **by** *blast*
from *lt-ex* **obtain** *z* **where** *z*:$z \sqsubset$ *ord.min less-eq z1 z2* **by** *blast*
from *z* **have** *zz1*: $z \sqsubset z1$ **and** *zz2*: $z \sqsubset z2$ **by** *simp-all*
{fix *x* **assume** *H*: $x \sqsubset z$
from *less-trans*[*OF H zz1*] *less-trans*[*OF H zz2*]
have $(P1\ x \wedge P2\ x) \longleftrightarrow (P1' \wedge P2')$ **using** *z1 zz1 z2 zz2* **by** *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *minf-disj*[*no-atp*]:
assumes *ex1*: $\exists\, z1.\ \forall\, x.\ x \sqsubset z1 \longrightarrow (P1\ x \longleftrightarrow P1')$
and *ex2*: $\exists\, z2.\ \forall\, x.\ x \sqsubset z2 \longrightarrow (P2\ x \longleftrightarrow P2')$
shows $\exists\, z.\ \forall\, x.\ x \sqsubset\ z \longrightarrow ((P1\ x \vee P2\ x) \longleftrightarrow (P1' \vee P2'))$
proof−
from *ex1 ex2* **obtain** *z1* **and** *z2* **where** *z1*: $\forall\, x.\ x \sqsubset z1 \longrightarrow (P1\ x \longleftrightarrow P1')$**and** *z2*: $\forall\, x.\ x \sqsubset z2 \longrightarrow (P2\ x \longleftrightarrow P2')$ **by** *blast*
from *lt-ex* **obtain** *z* **where** *z*:$z \sqsubset$ *ord.min less-eq z1 z2* **by** *blast*
from *z* **have** *zz1*: $z \sqsubset z1$ **and** *zz2*: $z \sqsubset z2$ **by** *simp-all*
{fix *x* **assume** *H*: $x \sqsubset z$
from *less-trans*[*OF H zz1*] *less-trans*[*OF H zz2*]
have $(P1\ x \vee P2\ x) \longleftrightarrow (P1' \vee P2')$ **using** *z1 zz1 z2 zz2* **by** *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *minf-ex*[*no-atp*]: **assumes** *ex*:$\exists\, z.\ \forall\, x.\ x \sqsubset z \longrightarrow (P\ x \longleftrightarrow P1)$ **and** *p1*: *P1* **shows** $\exists\ x.\ P\ x$
proof−
from *ex* **obtain** *z* **where** *z*: $\forall\, x.\ x \sqsubset z \longrightarrow (P\ x \longleftrightarrow P1)$ **by** *blast*
from *lt-ex* **obtain** *x* **where** *x*: $x \sqsubset z$ **by** *blast*
from *z x p1* **show** *?thesis* **by** *blast*
qed

end

locale *constr-dense-linorder* = *linorder-no-lb* + *linorder-no-ub* +

fixes *between*
assumes *between-less*: *less x y* $\Longrightarrow$ *less x* (*between x y*) $\wedge$ *less* (*between x y*) *y*
and *between-same*: *between x x* = *x*

sublocale *constr-dense-linorder* < *dense-linorder*
apply *unfold-locales*
using *gt-ex lt-ex between-less*
by (*auto*, *rule-tac x*=*between x y* **in** *exI*, *simp*)

context *constr-dense-linorder*
begin

lemma *rinf-U*[*no-atp*]:
assumes *fU*: *finite U*
and *lin-dense*: $\forall x\ l\ u.\ (\forall\ t.\ l \sqsubset t \wedge t \sqsubset u \longrightarrow t \notin U) \wedge l \sqsubset x \wedge x \sqsubset u \wedge P\ x \longrightarrow (\forall\ y.\ l \sqsubset y \wedge y \sqsubset u \longrightarrow P\ y\ )$
and *nmpiU*: $\forall x.\ \neg\ MP \wedge \neg PP \wedge P\ x \longrightarrow (\exists\ u{\in}\ U.\ \exists\ u' \in U.\ u \sqsubseteq x \wedge x \sqsubseteq u')$
and *nmi*: $\neg$ *MP* **and** *npi*: $\neg$ *PP* **and** *ex*: $\exists\ x.\ P\ x$
shows $\exists\ u{\in}\ U.\ \exists\ u' \in U.\ P$ (*between u u'*)
proof−
from *ex* **obtain** *x* **where** *px*: *P x* **by** *blast*
from *px nmi npi nmpiU* **have** $\exists\ u{\in}\ U.\ \exists\ u' \in U.\ u \sqsubseteq x \wedge x \sqsubseteq u'$ **by** *auto*
then obtain *u* **and** *u'* **where** *uU*:*u*$\in$ *U* **and** *uU'*: *u'* $\in$ *U* **and** *ux*:*u* $\sqsubseteq$ *x* **and** *xu'*:*x* $\sqsubseteq$ *u'* **by** *auto*
from *uU* **have** *Une*: *U* $\neq$ {} **by** *auto*
term *linorder.Min less-eq*
let *?l* = *linorder.Min less-eq U*
let *?u* = *linorder.Max less-eq U*
have *linM*: *?l* $\in$ *U* **using** *fU Une* **by** *simp*
have *uinM*: *?u* $\in$ *U* **using** *fU Une* **by** *simp*
have *lM*: $\forall\ t{\in}\ U.\ ?l \sqsubseteq t$ **using** *Une fU* **by** *auto*
have *Mu*: $\forall\ t{\in}\ U.\ t \sqsubseteq ?u$ **using** *Une fU* **by** *auto*
have *th*:*?l* $\sqsubseteq$ *u* **using** *uU Une lM* **by** *auto*
from *order-trans*[*OF th ux*] **have** *lx*: *?l* $\sqsubseteq$ *x* **.**
have *th*: *u'* $\sqsubseteq$ *?u* **using** *uU' Une Mu* **by** *simp*
from *order-trans*[*OF xu' th*] **have** *xu*: *x* $\sqsubseteq$ *?u* **.**
from *finite-set-intervals2*[**where** *P*=*P*,*OF px lx xu linM uinM fU lM Mu*]
have $(\exists\ s{\in}\ U.\ P\ s) \vee$
$(\exists\ t1{\in}\ U.\ \exists\ t2 \in U.\ (\forall\ y.\ t1 \sqsubset y \wedge y \sqsubset t2 \longrightarrow y \notin U) \wedge t1 \sqsubset x \wedge x \sqsubset t2 \wedge P\ x)$ **.**
moreover { fix *u* **assume** *um*: *u*$\in$*U* **and** *pu*: *P u*
have *between u u* = *u* **by** (*simp add*: *between-same*)
with *um pu* **have** *P* (*between u u*) **by** *simp*
with *um* **have** *?thesis* **by** *blast*}
moreover{
assume $\exists\ t1{\in}\ U.\ \exists\ t2 \in U.\ (\forall\ y.\ t1 \sqsubset y \wedge y \sqsubset t2 \longrightarrow y \notin U) \wedge t1 \sqsubset x \wedge x \sqsubset t2 \wedge P\ x$
then obtain *t1* **and** *t2* **where** *t1M*: *t1* $\in$ *U* **and** *t2M*: *t2*$\in$ *U*

and *noM*: $\forall\ y.\ t1 \sqsubset y \wedge y \sqsubset t2 \longrightarrow y \notin U$ **and** *t1x*: $t1 \sqsubset x$ **and** *xt2*: $x \sqsubset t2$ **and** *px*: $P\ x$
by *blast*
from *less-trans*[*OF t1x xt2*] **have** *t1t2*: $t1 \sqsubset t2$ **.**
let *?u* = *between t1 t2*
from *between-less t1t2* **have** *t1lu*: $t1 \sqsubset ?u$ **and** *ut2*: $?u \sqsubset t2$ **by** *auto*
from *lin-dense noM t1x xt2 px t1lu ut2* **have** $P\ ?u$ **by** *blast*
with *t1M t2M* **have** *?thesis* **by** *blast*}
ultimately show *?thesis* **by** *blast*
qed

theorem *fr-eq*[*no-atp*]:
assumes *fU*: *finite U*
and *lin-dense*: $\forall x\ l\ u.\ (\forall\ t.\ l \sqsubset t \wedge t \sqsubset u \longrightarrow t \notin U) \wedge l \sqsubset x \wedge x \sqsubset u \wedge P\ x \longrightarrow (\forall\ y.\ l \sqsubset y \wedge y \sqsubset u \longrightarrow P\ y\ )$
and *nmibnd*: $\forall x.\ \neg\ MP \wedge P\ x \longrightarrow (\exists\ u{\in}\ U.\ u \sqsubseteq x)$
and *npibnd*: $\forall x.\ \neg PP \wedge P\ x \longrightarrow (\exists\ u{\in}\ U.\ x \sqsubseteq u)$
and *mi*: $\exists z.\ \forall x.\ x \sqsubset z \longrightarrow (P\ x = MP)$ **and** *pi*: $\exists z.\ \forall x.\ z \sqsubset x \longrightarrow (P\ x = PP)$
shows $(\exists\ x.\ P\ x) \equiv (MP \vee PP \vee (\exists\ u \in U.\ \exists\ u'{\in}\ U.\ P\ (between\ u\ u')))$
(**is** - $\equiv$ (- $\vee$ - $\vee$ *?F*) **is** $?E \equiv ?D$)
proof−
{
assume *px*: $\exists\ x.\ P\ x$
have $MP \vee PP \vee (\neg\ MP \wedge \neg\ PP)$ **by** *blast*
moreover {**assume** $MP \vee PP$ **hence** *?D* **by** *blast*}
moreover {**assume** *nmi*: $\neg\ MP$ **and** *npi*: $\neg\ PP$
from *npmibnd*[*OF nmibnd npibnd*]
have *nmpiU*: $\forall x.\ \neg\ MP \wedge \neg PP \wedge P\ x \longrightarrow (\exists\ u{\in}\ U.\ \exists\ u' \in U.\ u \sqsubseteq x \wedge x \sqsubseteq u')$ **.**
from *rinf-U*[*OF fU lin-dense nmpiU nmi npi px*] **have** *?D* **by** *blast*}
ultimately have *?D* **by** *blast*}
moreover
{ **assume** *?D*
moreover {**assume** *m*:*MP* **from** *minf-ex*[*OF mi m*] **have** *?E* **.**}
moreover {**assume** *p*: *PP* **from** *pinf-ex*[*OF pi p*] **have** *?E* **. }**
moreover {**assume** *f*:*?F* **hence** *?E* **by** *blast*}
ultimately have *?E* **by** *blast*}
ultimately have $?E = ?D$ **by** *blast* **thus** $?E \equiv ?D$ **by** *simp*
qed

lemmas *minf-thms*[*no-atp*] = *minf-conj minf-disj minf-eq minf-neq minf-lt minf-le minf-gt minf-ge minf-P*
lemmas *pinf-thms*[*no-atp*] = *pinf-conj pinf-disj pinf-eq pinf-neq pinf-lt pinf-le pinf-gt pinf-ge pinf-P*

lemmas *nmi-thms*[*no-atp*] = *nmi-conj nmi-disj nmi-eq nmi-neq nmi-lt nmi-le nmi-gt nmi-ge nmi-P*
lemmas *npi-thms*[*no-atp*] = *npi-conj npi-disj npi-eq npi-neq npi-lt npi-le npi-gt*

```
npi-ge npi-P
lemmas lin-dense-thms[no-atp] = lin-dense-conj lin-dense-disj lin-dense-eq lin-dense-neq
lin-dense-lt lin-dense-le lin-dense-gt lin-dense-ge lin-dense-P

lemma ferrack-axiom[no-atp]: constr-dense-linorder less-eq less between
  by (rule constr-dense-linorder-axioms)
lemma atoms[no-atp]:
  shows TERM (less :: 'a ⇒ -)
    and TERM (less-eq :: 'a ⇒ -)
    and TERM (op = :: 'a ⇒ -) .

declare ferrack-axiom [ferrack minf: minf-thms pinf: pinf-thms
    nmi: nmi-thms npi: npi-thms lindense:
    lin-dense-thms qe: fr-eq atoms: atoms]

declaration ⟪
let
fun simps phi = map (Morphism.thm phi) [@{thm not-less}, @{thm not-le}]
fun generic-whatis phi =
 let
  val [lt, le] = map (Morphism.term phi) [@{term op ⊏}, @{term op ⊑}]
  fun h x t =
   case term-of t of
     Const(op =, -)$y$z => if term-of x aconv y then Ferrante-Rackoff-Data.Eq
                           else Ferrante-Rackoff-Data.Nox
  | @{term Not}$(Const(op =, -)$y$z) => if term-of x aconv y then Ferrante-Rackoff-Data.NEq
                           else Ferrante-Rackoff-Data.Nox
   | b$y$z => if Term.could-unify (b, lt) then
                if term-of x aconv y then Ferrante-Rackoff-Data.Lt
                else if term-of x aconv z then Ferrante-Rackoff-Data.Gt
                else Ferrante-Rackoff-Data.Nox
              else if Term.could-unify (b, le) then
                if term-of x aconv y then Ferrante-Rackoff-Data.Le
                else if term-of x aconv z then Ferrante-Rackoff-Data.Ge
                else Ferrante-Rackoff-Data.Nox
              else Ferrante-Rackoff-Data.Nox
   | - => Ferrante-Rackoff-Data.Nox
 in h end
 fun ss phi = HOL-ss addsimps (simps phi)
in
 Ferrante-Rackoff-Data.funs  @{thm ferrack-axiom}
  {isolate-conv = K (K (K Thm.reflexive)), whatis = generic-whatis, simpset =
ss}
end
⟫

end

use ferrante-rackoff.ML
```

method-setup *ferrack* = ⟪
  Scan.succeed (*SIMPLE-METHOD'* o *FerranteRackoff.dlo-tac*)
⟫ *Ferrante and Rackoff's algorithm for quantifier elimination in dense linear orders*

3.1 Ferrante and Rackoff algorithm over ordered fields

lemma *neg-prod-lt*:$(c::'a::linordered\text{-}field) < 0 \Longrightarrow ((c*x < 0) == (x > 0))$
proof−
  assume *H*: $c < 0$
 have $c*x < 0 = (0/c < x)$ **by** (*simp only*: *neg-divide-less-eq*[*OF H*] *algebra-simps*)
  also have $\ldots = (0 < x)$ **by** *simp*
  finally show $(c*x < 0) == (x > 0)$ **by** *simp*
qed

lemma *pos-prod-lt*:$(c::'a::linordered\text{-}field) > 0 \Longrightarrow ((c*x < 0) == (x < 0))$
proof−
  assume *H*: $c > 0$
 hence $c*x < 0 = (0/c > x)$ **by** (*simp only*: *pos-less-divide-eq*[*OF H*] *algebra-simps*)
  also have $\ldots = (0 > x)$ **by** *simp*
  finally show $(c*x < 0) == (x < 0)$ **by** *simp*
qed

lemma *neg-prod-sum-lt*: $(c::'a::linordered\text{-}field) < 0 \Longrightarrow ((c*x + t< 0) == (x > (-\ 1/c)*t))$
proof−
  assume *H*: $c < 0$
  have $c*x + t< 0 = (c*x < -t)$ **by** (*subst less-iff-diff-less-0* [*of* $c*x$ $-t$], *simp*)
 also have $\ldots = (-t/c < x)$ **by** (*simp only*: *neg-divide-less-eq*[*OF H*] *algebra-simps*)
  also have $\ldots = ((-\ 1/c)*t < x)$ **by** *simp*
  finally show $(c*x + t < 0) == (x > (-\ 1/c)*t)$ **by** *simp*
qed

lemma *pos-prod-sum-lt*:$(c::'a::linordered\text{-}field) > 0 \Longrightarrow ((c*x + t < 0) == (x < (-\ 1/c)*t))$
proof−
  assume *H*: $c > 0$
  have $c*x + t< 0 = (c*x < -t)$ **by** (*subst less-iff-diff-less-0* [*of* $c*x$ $-t$], *simp*)
 also have $\ldots = (-t/c > x)$ **by** (*simp only*: *pos-less-divide-eq*[*OF H*] *algebra-simps*)
  also have $\ldots = ((-\ 1/c)*t > x)$ **by** *simp*
  finally show $(c*x + t < 0) == (x < (-\ 1/c)*t)$ **by** *simp*
qed

lemma *sum-lt*:$((x::'a::ordered\text{-}ab\text{-}group\text{-}add) + t < 0) == (x < -\ t)$
  using *less-diff-eq*[**where** $a= x$ **and** $b=t$ **and** $c=0$] **by** *simp*

lemma *neg-prod-le*:$(c::'a::linordered\text{-}field) < 0 \Longrightarrow ((c*x <= 0) == (x >= 0))$
proof−
  assume *H*: $c < 0$

have $c*x <= 0 = (0/c <= x)$ **by** (*simp only*: *neg-divide-le-eq*[*OF H*] *algebra-simps*)
also have $\ldots = (0 <= x)$ **by** *simp*
finally show $(c*x <= 0) == (x >= 0)$ **by** *simp*
qed

lemma *pos-prod-le*:$(c::'a::linordered\text{-}field) > 0 \Longrightarrow ((c*x <= 0) == (x <= 0))$
proof−
assume *H*: $c > 0$
hence $c*x <= 0 = (0/c >= x)$ **by** (*simp only*: *pos-le-divide-eq*[*OF H*] *algebra-simps*)
also have $\ldots = (0 >= x)$ **by** *simp*
finally show $(c*x <= 0) == (x <= 0)$ **by** *simp*
qed

lemma *neg-prod-sum-le*: $(c::'a::linordered\text{-}field) < 0 \Longrightarrow ((c*x + t <= 0) == (x >= (-1/c)*t))$
proof−
assume *H*: $c < 0$
have $c*x + t <= 0 = (c*x <= -t)$ **by** (*subst le-iff-diff-le-0* [*of* $c*x$ $-t$], *simp*)
also have $\ldots = (-t/c <= x)$ **by** (*simp only*: *neg-divide-le-eq*[*OF H*] *algebra-simps*)
also have $\ldots = ((-1/c)*t <= x)$ **by** *simp*
finally show $(c*x + t <= 0) == (x >= (-1/c)*t)$ **by** *simp*
qed

lemma *pos-prod-sum-le*:$(c::'a::linordered\text{-}field) > 0 \Longrightarrow ((c*x + t <= 0) == (x <= (-1/c)*t))$
proof−
assume *H*: $c > 0$
have $c*x + t <= 0 = (c*x <= -t)$ **by** (*subst le-iff-diff-le-0* [*of* $c*x$ $-t$], *simp*)
also have $\ldots = (-t/c >= x)$ **by** (*simp only*: *pos-le-divide-eq*[*OF H*] *algebra-simps*)
also have $\ldots = ((-1/c)*t >= x)$ **by** *simp*
finally show $(c*x + t <= 0) == (x <= (-1/c)*t)$ **by** *simp*
qed

lemma *sum-le*:$((x::'a::ordered\text{-}ab\text{-}group\text{-}add) + t <= 0) == (x <= -t)$
using *le-diff-eq*[**where** *a*= *x* **and** *b*=*t* **and** *c*=*0*] **by** *simp*

lemma *nz-prod-eq*:$(c::'a::linordered\text{-}field) \neq 0 \Longrightarrow ((c*x = 0) == (x = 0))$ **by** *simp*
lemma *nz-prod-sum-eq*: $(c::'a::linordered\text{-}field) \neq 0 \Longrightarrow ((c*x + t = 0) == (x = (-1/c)*t))$
proof−
assume *H*: $c \neq 0$
have $c*x + t = 0 = (c*x = -t)$ **by** (*subst eq-iff-diff-eq-0* [*of* $c*x$ $-t$], *simp*)
also have $\ldots = (x = -t/c)$ **by** (*simp only*: *nonzero-eq-divide-eq*[*OF H*] *algebra-simps*)
finally show $(c*x + t = 0) == (x = (-1/c)*t)$ **by** *simp*
qed
lemma *sum-eq*:$((x::'a::ordered\text{-}ab\text{-}group\text{-}add) + t = 0) == (x = -t)$
using *eq-diff-eq*[**where** *a*= *x* **and** *b*=*t* **and** *c*=*0*] **by** *simp*

```
interpretation class-dense-linordered-field: constr-dense-linorder
 op <= op <
   λ x y. 1/2 * ((x::'a::{linordered-field,number-ring}) + y)
proof (unfold-locales, dlo, dlo, auto)
  fix x y::'a assume lt: x < y
  from  less-half-sum[OF lt] show x < (x + y) /2 by simp
next
  fix x y::'a assume lt: x < y
  from  gt-half-sum[OF lt] show (x + y) /2 < y by simp
qed

declaration⟪
let
fun earlier [] x y = false
      | earlier (h::t) x y =
    if h aconvc y then false else if h aconvc x then true else earlier t x y;

fun dest-frac ct = case term-of ct of
   Const (@{const-name Rings.divide},-) $ a $ b=>
    Rat.rat-of-quotient (snd (HOLogic.dest-number a), snd (HOLogic.dest-number
b))
| Const(@{const-name inverse}, -)$a => Rat.rat-of-quotient(1, HOLogic.dest-number
a |> snd)
| t => Rat.rat-of-int (snd (HOLogic.dest-number t))

fun mk-frac phi cT x =
 let val (a, b) = Rat.quotient-of-rat x
 in if b = 1 then Numeral.mk-cnumber cT a
    else Thm.capply
        (Thm.capply (Drule.cterm-rule (instantiate' [SOME cT] []) @{cpat op /})
                (Numeral.mk-cnumber cT a))
        (Numeral.mk-cnumber cT b)
 end

fun whatis x ct = case term-of ct of
  Const(@{const-name Groups.plus}, -)$(Const(@{const-name Groups.times},-)$-$y)$-
=>
     if y aconv term-of x then (c*x+t,[(funpow 2 Thm.dest-arg1) ct, Thm.dest-arg
ct])
     else (Nox,[])
| Const(@{const-name Groups.plus}, -)$y$- =>
     if y aconv term-of x then (x+t,[Thm.dest-arg ct])
     else (Nox,[])
| Const(@{const-name Groups.times}, -)$-$y =>
     if y aconv term-of x then (c*x,[Thm.dest-arg1 ct])
     else (Nox,[])
| t => if t aconv term-of x then (x,[]) else (Nox,[]);
```

```
fun xnormalize-conv ctxt [] ct = Thm.reflexive ct
| xnormalize-conv ctxt (vs as (x::-)) ct =
   case term-of ct of
   Const(@{const-name Orderings.less},-)$-$Const(@{const-name Groups.zero},-)
=>
    (case whatis x (Thm.dest-arg1 ct) of
    (c*x+t,[c,t]) =>
      let
       val cr = dest-frac c
       val clt = Thm.dest-fun2 ct
       val cz = Thm.dest-arg ct
       val neg = cr </ Rat.zero
       val cthp = Simplifier.rewrite (simpset-of ctxt)
             (Thm.capply @{cterm Trueprop}
               (if neg then Thm.capply (Thm.capply clt c) cz
                 else Thm.capply (Thm.capply clt cz) c))
       val cth = Thm.equal-elim (Thm.symmetric cthp) TrueI
         val th = Thm.implies-elim (instantiate' [SOME (ctyp-of-term x)] (map
SOME [c,x,t])
           (if neg then @{thm neg-prod-sum-lt} else @{thm pos-prod-sum-lt})) cth
       val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
                  (Semiring-Normalizer.semiring-normalize-ord-conv ctxt (earlier
vs)))) th
      in rth end
    | (x+t,[t]) =>
      let
       val T = ctyp-of-term x
       val th = instantiate' [SOME T] [SOME x, SOME t] @{thm sum-lt}
       val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
            (Semiring-Normalizer.semiring-normalize-ord-conv ctxt (earlier vs))))
th
      in  rth end
    | (c*x,[c]) =>
      let
       val cr = dest-frac c
       val clt = Thm.dest-fun2 ct
       val cz = Thm.dest-arg ct
       val neg = cr </ Rat.zero
       val cthp = Simplifier.rewrite (simpset-of ctxt)
             (Thm.capply @{cterm Trueprop}
               (if neg then Thm.capply (Thm.capply clt c) cz
                 else Thm.capply (Thm.capply clt cz) c))
       val cth = Thm.equal-elim (Thm.symmetric cthp) TrueI
         val th = Thm.implies-elim (instantiate' [SOME (ctyp-of-term x)] (map
SOME [c,x])
           (if neg then @{thm neg-prod-lt} else @{thm pos-prod-lt})) cth
       val rth = th
      in rth end
    | - => Thm.reflexive ct)
```

```
| Const(@{const-name Orderings.less-eq},-)$-$Const(@{const-name Groups.zero},-)
=>
  (case whatis x (Thm.dest-arg1 ct) of
   (c*x+t,[c,t]) =>
     let
      val T = ctyp-of-term x
      val cr = dest-frac c
      val clt = Drule.cterm-rule (instantiate' [SOME T] []) @{cpat op <}
      val cz = Thm.dest-arg ct
      val neg = cr </ Rat.zero
      val cthp = Simplifier.rewrite (simpset-of ctxt)
            (Thm.capply @{cterm Trueprop}
              (if neg then Thm.capply (Thm.capply clt c) cz
                else Thm.capply (Thm.capply clt cz) c))
      val cth = Thm.equal-elim (Thm.symmetric cthp) TrueI
      val th = Thm.implies-elim (instantiate' [SOME T] (map SOME [c,x,t])
          (if neg then @{thm neg-prod-sum-le} else @{thm pos-prod-sum-le})) cth
      val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
                 (Semiring-Normalizer.semiring-normalize-ord-conv ctxt (earlier
vs)))) th
     in rth end
   | (x+t,[t]) =>
     let
      val T = ctyp-of-term x
      val th = instantiate' [SOME T] [SOME x, SOME t] @{thm sum-le}
      val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
           (Semiring-Normalizer.semiring-normalize-ord-conv ctxt (earlier vs))))
th
     in  rth end
   | (c*x,[c]) =>
     let
      val T = ctyp-of-term x
      val cr = dest-frac c
      val clt = Drule.cterm-rule (instantiate' [SOME T] []) @{cpat op <}
      val cz = Thm.dest-arg ct
      val neg = cr </ Rat.zero
      val cthp = Simplifier.rewrite (simpset-of ctxt)
            (Thm.capply @{cterm Trueprop}
              (if neg then Thm.capply (Thm.capply clt c) cz
                else Thm.capply (Thm.capply clt cz) c))
      val cth = Thm.equal-elim (Thm.symmetric cthp) TrueI
        val th = Thm.implies-elim (instantiate' [SOME (ctyp-of-term x)] (map
SOME [c,x])
          (if neg then @{thm neg-prod-le} else @{thm pos-prod-le})) cth
      val rth = th
     in rth end
   | - => Thm.reflexive ct)
```

```
| Const(op =,-)$-$Const(@{const-name Groups.zero},-) =>
  (case whatis x (Thm.dest-arg1 ct) of
   (c*x+t,[c,t]) =>
     let
      val T = ctyp-of-term x
      val cr = dest-frac c
      val ceq = Thm.dest-fun2 ct
      val cz = Thm.dest-arg ct
      val cthp = Simplifier.rewrite (simpset-of ctxt)
          (Thm.capply @{cterm Trueprop}
           (Thm.capply @{cterm Not} (Thm.capply (Thm.capply ceq c) cz)))
      val cth = Thm.equal-elim (Thm.symmetric cthp) TrueI
      val th = Thm.implies-elim
            (instantiate' [SOME T] (map SOME [c,x,t]) @{thm nz-prod-sum-eq})
cth
      val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
                 (Semiring-Normalizer.semiring-normalize-ord-conv ctxt (earlier
vs)))) th
     in rth end
   | (x+t,[t]) =>
     let
      val T = ctyp-of-term x
      val th = instantiate' [SOME T] [SOME x, SOME t] @{thm sum-eq}
      val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
           (Semiring-Normalizer.semiring-normalize-ord-conv ctxt (earlier vs))))
th
     in  rth end
   | (c*x,[c]) =>
     let
      val T = ctyp-of-term x
      val cr = dest-frac c
      val ceq = Thm.dest-fun2 ct
      val cz = Thm.dest-arg ct
      val cthp = Simplifier.rewrite (simpset-of ctxt)
          (Thm.capply @{cterm Trueprop}
           (Thm.capply @{cterm Not} (Thm.capply (Thm.capply ceq c) cz)))
      val cth = Thm.equal-elim (Thm.symmetric cthp) TrueI
      val rth = Thm.implies-elim
             (instantiate' [SOME T] (map SOME [c,x]) @{thm nz-prod-eq}) cth
     in rth end
   | - => Thm.reflexive ct);

local
  val less-iff-diff-less-0 = mk-meta-eq @{thm less-iff-diff-less-0}
  val le-iff-diff-le-0 = mk-meta-eq @{thm le-iff-diff-le-0}
  val eq-iff-diff-eq-0 = mk-meta-eq @{thm eq-iff-diff-eq-0}
in
fun field-isolate-conv phi ctxt vs ct = case term-of ct of
```

```
  Const(@{const-name Orderings.less},-)$a$b =>
   let val (ca,cb) = Thm.dest-binop ct
       val T = ctyp-of-term ca
       val th = instantiate' [SOME T] [SOME ca, SOME cb] less-iff-diff-less-0
       val nth = Conv.fconv-rule
         (Conv.arg-conv (Conv.arg1-conv
             (Semiring-Normalizer.semiring-normalize-ord-conv @{context} (earlier
vs)))) th
       val rth = Thm.transitive nth (xnormalize-conv ctxt vs (Thm.rhs-of nth))
   in rth end
| Const(@{const-name Orderings.less-eq},-)$a$b =>
   let val (ca,cb) = Thm.dest-binop ct
       val T = ctyp-of-term ca
       val th = instantiate' [SOME T] [SOME ca, SOME cb] le-iff-diff-le-0
       val nth = Conv.fconv-rule
         (Conv.arg-conv (Conv.arg1-conv
             (Semiring-Normalizer.semiring-normalize-ord-conv @{context} (earlier
vs)))) th
       val rth = Thm.transitive nth (xnormalize-conv ctxt vs (Thm.rhs-of nth))
   in rth end

| Const(op =,-)$a$b =>
   let val (ca,cb) = Thm.dest-binop ct
       val T = ctyp-of-term ca
       val th = instantiate' [SOME T] [SOME ca, SOME cb] eq-iff-diff-eq-0
       val nth = Conv.fconv-rule
         (Conv.arg-conv (Conv.arg1-conv
             (Semiring-Normalizer.semiring-normalize-ord-conv @{context} (earlier
vs)))) th
       val rth = Thm.transitive nth (xnormalize-conv ctxt vs (Thm.rhs-of nth))
   in rth end
| @{term Not} $(Const(op =,-)$a$b) => Conv.arg-conv (field-isolate-conv phi ctxt
vs) ct
| - => Thm.reflexive ct
end;

fun classfield-whatis phi =
 let
  fun h x t =
   case term-of t of
     Const(op =, -)$y$z => if term-of x aconv y then Ferrante-Rackoff-Data.Eq
                           else Ferrante-Rackoff-Data.Nox
  | @{term Not}$(Const(op =, -)$y$z) => if term-of x aconv y then Ferrante-Rackoff-Data.NEq
                           else Ferrante-Rackoff-Data.Nox
   | Const(@{const-name Orderings.less},-)$y$z =>
       if term-of x aconv y then Ferrante-Rackoff-Data.Lt
        else if term-of x aconv z then Ferrante-Rackoff-Data.Gt
        else Ferrante-Rackoff-Data.Nox
   | Const (@{const-name Orderings.less-eq},-)$y$z =>
```

```
        if term-of x aconv y then Ferrante-Rackoff-Data.Le
        else if term-of x aconv z then Ferrante-Rackoff-Data.Ge
        else Ferrante-Rackoff-Data.Nox
   | - => Ferrante-Rackoff-Data.Nox
 in h end;
fun class-field-ss phi =
    HOL-basic-ss addsimps ([@{thm linorder-not-less}, @{thm linorder-not-le}])
    addsplits [@{thm abs-split},@{thm split-max}, @{thm split-min}]

in
Ferrante-Rackoff-Data.funs @{thm class-dense-linordered-field.ferrack-axiom}
 {isolate-conv = field-isolate-conv, whatis = classfield-whatis, simpset = class-field-ss}
end
⟩⟩

lemma upper-bound-finite-set:
  assumes fS: finite S
  shows ∃(a::'a::linorder). ∀ x ∈ S. f x ≤ a
proof(induct rule: finite-induct[OF fS])
  case 1 thus ?case by simp
next
  case (2 x F)
  from 2.hyps obtain a where a:∀ x ∈F. f x ≤ a by blast
  let ?a = max a (f x)
  have m: a ≤ ?a f x ≤ ?a by simp-all
  {fix y assume y: y ∈ insert x F
    {assume y = x hence f y ≤ ?a using m by simp}
    moreover
    {assume yF: y∈ F from a[rule-format, OF yF] m have f y ≤ ?a by (simp
add: max-def)}
    ultimately have f y ≤ ?a using y by blast}
  then show ?case by blast
qed

lemma lower-bound-finite-set:
  assumes fS: finite S
  shows ∃(a::'a::linorder). ∀ x ∈ S. f x ≥ a
proof(induct rule: finite-induct[OF fS])
  case 1 thus ?case by simp
next
  case (2 x F)
  from 2.hyps obtain a where a:∀ x ∈F. f x ≥ a by blast
  let ?a = min a (f x)
  have m: a ≥ ?a f x ≥ ?a by simp-all
  {fix y assume y: y ∈ insert x F
    {assume y = x hence f y ≥ ?a using m by simp}
    moreover
    {assume yF: y∈ F from a[rule-format, OF yF] m have f y ≥ ?a by (simp
add: min-def)}
```

ultimately have $f\ y \geq ?a$ **using** y **by** *blast*}
then show *?case* **by** *blast*
qed

lemma *bound-finite-set*: **assumes** f: *finite* S
shows $\exists\, a.\ \forall\, x \in S.\ (f\ x ::\ 'a ::\ linorder) \leq a$
proof−
let *?F* = f ' S
from f **have** *fF*: *finite ?F* **by** *simp*
let *?a* = *Max ?F*
{**assume** $S = \{\}$ **hence** *?thesis* **by** *blast*}
moreover
{**assume** *Se*: $S \neq \{\}$ **hence** *Fe*: $?F \neq \{\}$ **by** *simp*
{**fix** x **assume** x: $x \in S$
hence *th0*: $f\ x \in ?F$ **by** *simp*
hence $f\ x \leq ?a$ **using** *Max-ge*[*OF fF th0*] **..**}
hence *?thesis* **by** *blast*}
ultimately show *?thesis* **by** *blast*
qed

end

4 FrechetDeriv: Frechet Derivative

theory *FrechetDeriv*
imports *Lim Complex-Main*
begin

definition
fderiv ::
['*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*, '*a*, '*a* ⇒ '*b*] ⇒ *bool*
— Frechet derivative: D is derivative of function f at x
((*FDERIV* (-)/ (-)/ :> (-)) [*1000*, *1000*, *60*] *60*) **where**
FDERIV f x :> *D* = (*bounded-linear D* ∧
(λ*h*. *norm* (*f* (*x* + *h*) − *f x* − *D h*) / *norm h*) −− *0* −−> *0*)

lemma *FDERIV-I*:
⟦*bounded-linear D*; (λ*h*. *norm* (*f* (*x* + *h*) − *f x* − *D h*) / *norm h*) −− *0* −−> *0*⟧
⟹ *FDERIV f x* :> *D*
by (*simp add*: *fderiv-def*)

lemma *FDERIV-D*:
FDERIV f x :> *D* ⟹ (λ*h*. *norm* (*f* (*x* + *h*) − *f x* − *D h*) / *norm h*) −− *0* −−> *0*
by (*simp add*: *fderiv-def*)

lemma *FDERIV-bounded-linear*: *FDERIV f x :> D* $\Longrightarrow$ *bounded-linear D*
by (*simp add*: *fderiv-def*)

lemma *bounded-linear-zero*:
bounded-linear ($\lambda x::'a::$*real-normed-vector*. $0::'b::$*real-normed-vector*)
proof
show $(0::'b) = 0 + 0$ **by** *simp*
fix *r* **show** $(0::'b) =$ *scaleR r 0* **by** *simp*
have $\forall x::'a.$ *norm* $(0::'b) \leq$ *norm x* $*$ *0* **by** *simp*
thus $\exists K. \forall x::'a.$ *norm* $(0::'b) \leq$ *norm x* $*$ *K* **..**
qed

lemma *FDERIV-const*: *FDERIV* ($\lambda x.$ *k*) *x :>* ($\lambda h.$ *0*)
by (*simp add*: *fderiv-def bounded-linear-zero*)

lemma *bounded-linear-ident*:
bounded-linear ($\lambda x::'a::$*real-normed-vector*. *x*)
proof
fix *x y* :: *'a* **show** *x* + *y* = *x* + *y* **by** *simp*
fix *r* **and** *x* :: *'a* **show** *scaleR r x* = *scaleR r x* **by** *simp*
have $\forall x::'a.$ *norm x* $\leq$ *norm x* $*$ *1* **by** *simp*
thus $\exists K. \forall x::'a.$ *norm x* $\leq$ *norm x* $*$ *K* **..**
qed

lemma *FDERIV-ident*: *FDERIV* ($\lambda x.$ *x*) *x :>* ($\lambda h.$ *h*)
by (*simp add*: *fderiv-def bounded-linear-ident*)

4.1 Addition

lemma *bounded-linear-add*:
assumes *bounded-linear f*
assumes *bounded-linear g*
shows *bounded-linear* ($\lambda x.$ *f x* + *g x*)
proof −
interpret *f*: *bounded-linear f* **by** *fact*
interpret *g*: *bounded-linear g* **by** *fact*
show *?thesis* **apply** (*unfold-locales*)
apply (*simp only*: *f.add g.add add-ac*)
apply (*simp only*: *f.scaleR g.scaleR scaleR-right-distrib*)
apply (*rule f.pos-bounded* [*THEN exE*], *rename-tac Kf*)
apply (*rule g.pos-bounded* [*THEN exE*], *rename-tac Kg*)
apply (*rule-tac x*=*Kf* + *Kg* **in** *exI*, *safe*)
apply (*subst right-distrib*)
apply (*rule order-trans* [*OF norm-triangle-ineq*])
apply (*rule add-mono*, *erule spec*, *erule spec*)
done
qed

lemma *norm-ratio-ineq*:
fixes $x\ y$ :: *'a*::*real-normed-vector*
fixes h :: *'b*::*real-normed-vector*
shows $norm\ (x + y)\ /\ norm\ h \leq norm\ x\ /\ norm\ h + norm\ y\ /\ norm\ h$
apply (*rule ord-le-eq-trans*)
apply (*rule divide-right-mono*)
apply (*rule norm-triangle-ineq*)
apply (*rule norm-ge-zero*)
apply (*rule add-divide-distrib*)
done

lemma *FDERIV-add*:
assumes f: $FDERIV\ f\ x :> F$
assumes g: $FDERIV\ g\ x :> G$
shows $FDERIV\ (\lambda x.\ f\ x + g\ x)\ x :> (\lambda h.\ F\ h + G\ h)$
proof (*rule FDERIV-I*)
show *bounded-linear* $(\lambda h.\ F\ h + G\ h)$
apply (*rule bounded-linear-add*)
apply (*rule FDERIV-bounded-linear* [*OF f*])
apply (*rule FDERIV-bounded-linear* [*OF g*])
done
next
have f': $(\lambda h.\ norm\ (f\ (x + h) - f\ x - F\ h)\ /\ norm\ h) -- 0 --> 0$
using f **by** (*rule FDERIV-D*)
have g': $(\lambda h.\ norm\ (g\ (x + h) - g\ x - G\ h)\ /\ norm\ h) -- 0 --> 0$
using g **by** (*rule FDERIV-D*)
from $f'\ g'$
have $(\lambda h.\ norm\ (f\ (x + h) - f\ x - F\ h)\ /\ norm\ h$
$+ norm\ (g\ (x + h) - g\ x - G\ h)\ /\ norm\ h) -- 0 --> 0$
by (*rule LIM-add-zero*)
thus $(\lambda h.\ norm\ (f\ (x + h) + g\ (x + h) - (f\ x + g\ x) - (F\ h + G\ h))$
$/\ norm\ h) -- 0 --> 0$
apply (*rule real-LIM-sandwich-zero*)
apply (*simp add*: *divide-nonneg-pos*)
apply (*simp only*: *add-diff-add*)
apply (*rule norm-ratio-ineq*)
done
qed

4.2 Subtraction

lemma *bounded-linear-minus*:
assumes *bounded-linear f*
shows $bounded\text{-}linear\ (\lambda x.\ -\ f\ x)$
proof −
interpret f: *bounded-linear f* **by** *fact*
show *?thesis* **apply** (*unfold-locales*)
apply (*simp add*: *f.add*)
apply (*simp add*: *f.scaleR*)

apply (*simp add*: *f.bounded*)
done
qed

lemma *FDERIV-minus*:
FDERIV f x :> F $\Longrightarrow$ *FDERIV* ($\lambda x.\ -\ f\ x$) *x :>* ($\lambda h.\ -\ F\ h$)
apply (*rule FDERIV-I*)
apply (*rule bounded-linear-minus*)
apply (*erule FDERIV-bounded-linear*)
apply (*simp only*: *fderiv-def minus-diff-minus norm-minus-cancel*)
done

lemma *FDERIV-diff*:
⟦*FDERIV f x :> F*; *FDERIV g x :> G*⟧
$\Longrightarrow$ *FDERIV* ($\lambda x.\ f\ x\ -\ g\ x$) *x :>* ($\lambda h.\ F\ h\ -\ G\ h$)
by (*simp only*: *diff-minus FDERIV-add FDERIV-minus*)

4.3 Continuity

lemma *FDERIV-isCont*:
assumes *f*: *FDERIV f x :> F*
shows *isCont f x*
proof −
from *f* **interpret** *F*: *bounded-linear F* **by** (*rule FDERIV-bounded-linear*)
have ($\lambda h.$ *norm* (*f* (*x* + *h*) − *f x* − *F h*) / *norm h*) −− *0* −−> *0*
by (*rule FDERIV-D* [*OF f*])
hence ($\lambda h.$ *norm* (*f* (*x* + *h*) − *f x* − *F h*) / *norm h* ∗ *norm h*) −− *0* −−> *0*
by (*intro LIM-mult-zero LIM-norm-zero LIM-ident*)
hence ($\lambda h.$ *norm* (*f* (*x* + *h*) − *f x* − *F h*)) −− *0* −−> *0*
by (*simp cong*: *LIM-cong*)
hence ($\lambda h.$ *f* (*x* + *h*) − *f x* − *F h*) −− *0* −−> *0*
by (*rule LIM-norm-zero-cancel*)
hence ($\lambda h.$ *f* (*x* + *h*) − *f x* − *F h* + *F h*) −− *0* −−> *0*
by (*intro LIM-add-zero F.LIM-zero LIM-ident*)
hence ($\lambda h.$ *f* (*x* + *h*) − *f x*) −− *0* −−> *0*
by *simp*
thus *isCont f x*
unfolding *isCont-iff* **by** (*rule LIM-zero-cancel*)
qed

4.4 Composition

lemma *real-divide-cancel-lemma*:
fixes *a b c* :: *real*
shows (*b* = *0* $\Longrightarrow$ *a* = *0*) $\Longrightarrow$ (*a* / *b*) ∗ (*b* / *c*) = *a* / *c*
by *simp*

lemma *bounded-linear-compose*:
assumes *bounded-linear f*
assumes *bounded-linear g*

```
  shows bounded-linear (λx. f (g x))
proof −
  interpret f: bounded-linear f by fact
  interpret g: bounded-linear g by fact
  show ?thesis proof (unfold-locales)
    fix x y show f (g (x + y)) = f (g x) + f (g y)
      by (simp only: f.add g.add)
  next
    fix r x show f (g (scaleR r x)) = scaleR r (f (g x))
      by (simp only: f.scaleR g.scaleR)
  next
    from f.pos-bounded
    obtain Kf where f: ⋀x. norm (f x) ≤ norm x * Kf and Kf: 0 < Kf by fast
    from g.pos-bounded
    obtain Kg where g: ⋀x. norm (g x) ≤ norm x * Kg by fast
    show ∃ K. ∀ x. norm (f (g x)) ≤ norm x * K
    proof (intro exI allI)
      fix x
      have norm (f (g x)) ≤ norm (g x) * Kf
        using f .
      also have ... ≤ (norm x * Kg) * Kf
        using g Kf [THEN order-less-imp-le] by (rule mult-right-mono)
      also have (norm x * Kg) * Kf = norm x * (Kg * Kf)
        by (rule mult-assoc)
      finally show norm (f (g x)) ≤ norm x * (Kg * Kf) .
    qed
  qed
qed

lemma FDERIV-compose:
  fixes f :: 'a::real-normed-vector ⇒ 'b::real-normed-vector
  fixes g :: 'b::real-normed-vector ⇒ 'c::real-normed-vector
  assumes f: FDERIV f x :> F
  assumes g: FDERIV g (f x) :> G
  shows FDERIV (λx. g (f x)) x :> (λh. G (F h))
proof (rule FDERIV-I)
  from FDERIV-bounded-linear [OF g] FDERIV-bounded-linear [OF f]
  show bounded-linear (λh. G (F h))
    by (rule bounded-linear-compose)
next
  let ?Rf = λh. f (x + h) − f x − F h
  let ?Rg = λk. g (f x + k) − g (f x) − G k
  let ?k = λh. f (x + h) − f x
  let ?Nf = λh. norm (?Rf h) / norm h
  let ?Ng = λh. norm (?Rg (?k h)) / norm (?k h)
  from f interpret F: bounded-linear F by (rule FDERIV-bounded-linear)
  from g interpret G: bounded-linear G by (rule FDERIV-bounded-linear)
  from F.bounded obtain kF where kF: ⋀x. norm (F x) ≤ norm x * kF by fast
  from G.bounded obtain kG where kG: ⋀x. norm (G x) ≤ norm x * kG by
```

fast

let *?fun2* = $\lambda h.$ *?Nf h* $*$ *kG* $+$ *?Ng h* $*$ (*?Nf h* $+$ *kF*)

show ($\lambda h.$ *norm* (*g* (*f* (*x* $+$ *h*)) $-$ *g* (*f x*) $-$ *G* (*F h*)) / *norm h*) $--$ *0* $--{>}$ *0*
proof (*rule real-LIM-sandwich-zero*)
have *Nf*: *?Nf* $--$ *0* $--{>}$ *0*
using *FDERIV-D* [*OF f*] **.**

have *Ng1*: *isCont* ($\lambda k.$ *norm* (*?Rg k*) / *norm k*) *0*
by (*simp add*: *isCont-def FDERIV-D* [*OF g*])
have *Ng2*: *?k* $--$ *0* $--{>}$ *0*
apply (*rule LIM-zero*)
apply (*fold isCont-iff*)
apply (*rule FDERIV-isCont* [*OF f*])
done
have *Ng*: *?Ng* $--$ *0* $--{>}$ *0*
using *isCont-LIM-compose* [*OF Ng1 Ng2*] **by** *simp*

have ($\lambda h.$ *?Nf h* $*$ *kG* $+$ *?Ng h* $*$ (*?Nf h* $+$ *kF*))
$--$ *0* $--{>}$ *0* $*$ *kG* $+$ *0* $*$ (*0* $+$ *kF*)
by (*intro LIM-add LIM-mult LIM-const Nf Ng*)
thus ($\lambda h.$ *?Nf h* $*$ *kG* $+$ *?Ng h* $*$ (*?Nf h* $+$ *kF*)) $--$ *0* $--{>}$ *0*
by *simp*
next
fix *h*::*'a* **assume** *h*: *h* $\neq$ *0*
thus *0* $\leq$ *norm* (*g* (*f* (*x* $+$ *h*)) $-$ *g* (*f x*) $-$ *G* (*F h*)) / *norm h*
by (*simp add*: *divide-nonneg-pos*)
next
fix *h*::*'a* **assume** *h*: *h* $\neq$ *0*
have *g* (*f* (*x* $+$ *h*)) $-$ *g* (*f x*) $-$ *G* (*F h*) $=$ *G* (*?Rf h*) $+$ *?Rg* (*?k h*)
by (*simp add*: *G.diff*)
hence *norm* (*g* (*f* (*x* $+$ *h*)) $-$ *g* (*f x*) $-$ *G* (*F h*)) / *norm h*
$=$ *norm* (*G* (*?Rf h*) $+$ *?Rg* (*?k h*)) / *norm h*
by (*rule arg-cong*)
also have $\ldots \leq$ *norm* (*G* (*?Rf h*)) / *norm h* $+$ *norm* (*?Rg* (*?k h*)) / *norm h*
by (*rule norm-ratio-ineq*)
also have $\ldots \leq$ *?Nf h* $*$ *kG* $+$ *?Ng h* $*$ (*?Nf h* $+$ *kF*)
proof (*rule add-mono*)
show *norm* (*G* (*?Rf h*)) / *norm h* $\leq$ *?Nf h* $*$ *kG*
apply (*rule ord-le-eq-trans*)
apply (*rule divide-right-mono* [*OF kG norm-ge-zero*])
apply *simp*
done
next
have *norm* (*?Rg* (*?k h*)) / *norm h* $=$ *?Ng h* $*$ (*norm* (*?k h*) / *norm h*)
apply (*rule real-divide-cancel-lemma* [*symmetric*])
apply (*simp add*: *G.zero*)
done

```
    also have ... ≤ ?Ng h * (?Nf h + kF)
    proof (rule mult-left-mono)
      have norm (?k h) / norm h = norm (?Rf h + F h) / norm h
        by simp
      also have ... ≤ ?Nf h + norm (F h) / norm h
        by (rule norm-ratio-ineq)
      also have ... ≤ ?Nf h + kF
        apply (rule add-left-mono)
        apply (subst pos-divide-le-eq, simp add: h)
        apply (subst mult-commute)
        apply (rule kF)
        done
      finally show norm (?k h) / norm h ≤ ?Nf h + kF .
    next
      show 0 ≤ ?Ng h
      apply (case-tac f (x + h) - f x = 0, simp)
      apply (rule divide-nonneg-pos [OF norm-ge-zero])
      apply simp
      done
    qed
    finally show norm (?Rg (?k h)) / norm h ≤ ?Ng h * (?Nf h + kF) .
  qed
  finally show norm (g (f (x + h)) - g (f x) - G (F h)) / norm h
      ≤ ?Nf h * kG + ?Ng h * (?Nf h + kF) .
 qed
qed
```

4.5 Product Rule

```
lemma (in bounded-bilinear) FDERIV-lemma:
  a' ** b' - a ** b - (a ** B + A ** b)
   = a ** (b' - b - B) + (a' - a - A) ** b' + A ** (b' - b)
by (simp add: diff-left diff-right)

lemma (in bounded-bilinear) FDERIV:
  fixes x :: 'd::real-normed-vector
  assumes f: FDERIV f x :> F
  assumes g: FDERIV g x :> G
  shows FDERIV (λx. f x ** g x) x :> (λh. f x ** G h + F h ** g x)
proof (rule FDERIV-I)
  show bounded-linear (λh. f x ** G h + F h ** g x)
    apply (rule bounded-linear-add)
    apply (rule bounded-linear-compose [OF bounded-linear-right])
    apply (rule FDERIV-bounded-linear [OF g])
    apply (rule bounded-linear-compose [OF bounded-linear-left])
    apply (rule FDERIV-bounded-linear [OF f])
    done
next
  from bounded-linear.bounded [OF FDERIV-bounded-linear [OF f]]
```

obtain *KF* **where** *norm-F*: $\bigwedge x.\ norm\ (F\ x) \leq norm\ x * KF$ **by** *fast*

from *pos-bounded* **obtain** *K* **where** *K*: $0 < K$ **and** *norm-prod*:
$\bigwedge a\ b.\ norm\ (a\ **\ b) \leq norm\ a * norm\ b * K$ **by** *fast*

let *?Rf* $= \lambda h.\ f\ (x + h) - f\ x - F\ h$
let *?Rg* $= \lambda h.\ g\ (x + h) - g\ x - G\ h$

let *?fun1* $= \lambda h.$
$norm\ (f\ x\ **\ ?Rg\ h + ?Rf\ h\ **\ g\ (x + h) + F\ h\ **\ (g\ (x + h) - g\ x)) /$
$norm\ h$

let *?fun2* $= \lambda h.$
$norm\ (f\ x) * (norm\ (?Rg\ h) / norm\ h) * K +$
$norm\ (?Rf\ h) / norm\ h * norm\ (g\ (x + h)) * K +$
$KF * norm\ (g\ (x + h) - g\ x) * K$

have *?fun1* $-- 0 --> 0$
proof (*rule real-LIM-sandwich-zero*)
from *f g isCont-iff* [*THEN iffD1*, *OF FDERIV-isCont* [*OF g*]]
have *?fun2* $-- 0 -->$
$norm\ (f\ x) * 0 * K + 0 * norm\ (g\ x) * K + KF * norm\ (0::'b) * K$
by (*intro LIM-add LIM-mult LIM-const LIM-norm LIM-zero FDERIV-D*)
thus *?fun2* $-- 0 --> 0$
by *simp*
next
fix $h::'d$ **assume** $h \neq 0$
thus $0 \leq$ *?fun1 h*
by (*simp add*: *divide-nonneg-pos*)
next
fix $h::'d$ **assume** $h \neq 0$
have *?fun1 h* $\leq (norm\ (f\ x) * norm\ (?Rg\ h) * K +$
$norm\ (?Rf\ h) * norm\ (g\ (x + h)) * K +$
$norm\ h * KF * norm\ (g\ (x + h) - g\ x) * K) / norm\ h$
by (*intro*
divide-right-mono mult-mono'
order-trans [*OF norm-triangle-ineq add-mono*]
order-trans [*OF norm-prod mult-right-mono*]
mult-nonneg-nonneg order-refl norm-ge-zero norm-F
K [*THEN order-less-imp-le*]
)
also have $\ldots =$ *?fun2 h*
by (*simp add*: *add-divide-distrib*)
finally show *?fun1 h* $\leq$ *?fun2 h* **.**
qed
thus ($\lambda h.$
$norm\ (f\ (x + h)\ **\ g\ (x + h) - f\ x\ **\ g\ x - (f\ x\ **\ G\ h + F\ h\ **\ g\ x))$
$/\ norm\ h) -- 0 --> 0$
by (*simp only*: *FDERIV-lemma*)

qed

lemmas *FDERIV-mult = mult.FDERIV*

lemmas *FDERIV-scaleR = scaleR.FDERIV*

4.6 Powers

lemma *FDERIV-power-Suc*:
fixes *x* :: *'a*::{*real-normed-algebra*,*comm-ring-1*}
shows *FDERIV* (λ*x*. *x* ^ *Suc n*) *x* :> (λ*h*. (*1* + *of-nat n*) ∗ *x* ^ *n* ∗ *h*)
apply (*induct n*)
apply (*simp add*: *FDERIV-ident*)
apply (*drule FDERIV-mult* [*OF FDERIV-ident*])
apply (*simp only*: *of-nat-Suc left-distrib mult-1-left*)
apply (*simp only*: *power-Suc right-distrib add-ac mult-ac*)
done

lemma *FDERIV-power*:
fixes *x* :: *'a*::{*real-normed-algebra*,*comm-ring-1*}
shows *FDERIV* (λ*x*. *x* ^ *n*) *x* :> (λ*h*. *of-nat n* ∗ *x* ^ (*n* − *1*) ∗ *h*)
apply (*cases n*)
apply (*simp add*: *FDERIV-const*)
apply (*simp add*: *FDERIV-power-Suc del*: *power-Suc*)
done

4.7 Inverse

lemmas *bounded-linear-mult-const* =
mult.bounded-linear-left [*THEN bounded-linear-compose*]

lemmas *bounded-linear-const-mult* =
mult.bounded-linear-right [*THEN bounded-linear-compose*]

lemma *FDERIV-inverse*:
fixes *x* :: *'a*::*real-normed-div-algebra*
assumes *x*: *x* ≠ *0*
shows *FDERIV inverse x* :> (λ*h*. − (*inverse x* ∗ *h* ∗ *inverse x*))
(**is** *FDERIV ?inv* - :> -)
proof (*rule FDERIV-I*)
show *bounded-linear* (λ*h*. − (*?inv x* ∗ *h* ∗ *?inv x*))
apply (*rule bounded-linear-minus*)
apply (*rule bounded-linear-mult-const*)
apply (*rule bounded-linear-const-mult*)
apply (*rule bounded-linear-ident*)
done
next
show (λ*h*. *norm* (*?inv* (*x* + *h*) − *?inv x* − − (*?inv x* ∗ *h* ∗ *?inv x*)) / *norm h*)
−− *0* −−> *0*
proof (*rule LIM-equal2*)

```
  show 0 < norm x using x by simp
next
  fix h::'a
  assume 1: h ≠ 0
  assume norm (h - 0) < norm x
  hence h ≠ -x by clarsimp
  hence 2: x + h ≠ 0
    apply (rule contrapos-nn)
    apply (rule sym)
    apply (erule minus-unique)
    done
  show norm (?inv (x + h) - ?inv x - - (?inv x * h * ?inv x)) / norm h
      = norm ((?inv (x + h) - ?inv x) * h * ?inv x) / norm h
    apply (subst inverse-diff-inverse [OF 2 x])
    apply (subst minus-diff-minus)
    apply (subst norm-minus-cancel)
    apply (simp add: left-diff-distrib)
    done
next
  show (λh. norm ((?inv (x + h) - ?inv x) * h * ?inv x) / norm h)
      -- 0 --> 0
  proof (rule real-LIM-sandwich-zero)
    show (λh. norm (?inv (x + h) - ?inv x) * norm (?inv x))
        -- 0 --> 0
      apply (rule LIM-mult-left-zero)
      apply (rule LIM-norm-zero)
      apply (rule LIM-zero)
      apply (rule LIM-offset-zero)
      apply (rule LIM-inverse)
      apply (rule LIM-ident)
      apply (rule x)
      done
  next
    fix h::'a assume h: h ≠ 0
    show 0 ≤ norm ((?inv (x + h) - ?inv x) * h * ?inv x) / norm h
      apply (rule divide-nonneg-pos)
      apply (rule norm-ge-zero)
      apply (simp add: h)
      done
  next
    fix h::'a assume h: h ≠ 0
    have norm ((?inv (x + h) - ?inv x) * h * ?inv x) / norm h
        ≤ norm (?inv (x + h) - ?inv x) * norm h * norm (?inv x) / norm h
      apply (rule divide-right-mono [OF - norm-ge-zero])
      apply (rule order-trans [OF norm-mult-ineq])
      apply (rule mult-right-mono [OF - norm-ge-zero])
      apply (rule norm-mult-ineq)
      done
    also have ... = norm (?inv (x + h) - ?inv x) * norm (?inv x)
```

```
      by simp
    finally show norm ((?inv (x + h) − ?inv x) ∗ h ∗ ?inv x) / norm h
        ≤ norm (?inv (x + h) − ?inv x) ∗ norm (?inv x) .
  qed
 qed
qed
```

4.8 Alternate definition

```
lemma field-fderiv-def:
 fixes x :: 'a::real-normed-field shows
 FDERIV f x :> (λh. h ∗ D) = (λh. (f (x + h) − f x) / h) −− 0 −−> D
apply (unfold fderiv-def)
apply (simp add: mult.bounded-linear-left)
apply (simp cong: LIM-cong add: nonzero-norm-divide [symmetric])
apply (subst diff-divide-distrib)
apply (subst times-divide-eq-left [symmetric])
apply (simp cong: LIM-cong)
apply (simp add: LIM-norm-zero-iff LIM-zero-iff)
done

end
```

5 Inner-Product: Inner Product Spaces and the Gradient Derivative

```
theory Inner-Product
imports Complex-Main FrechetDeriv
begin
```

5.1 Real inner product spaces

Temporarily relax type constraints for *open*, *dist*, and *norm*.

```
setup ⟪ Sign.add-const-constraint
 (@{const-name open}, SOME @{typ 'a::open set ⇒ bool}) ⟫

setup ⟪ Sign.add-const-constraint
 (@{const-name dist}, SOME @{typ 'a::dist ⇒ 'a ⇒ real}) ⟫

setup ⟪ Sign.add-const-constraint
 (@{const-name norm}, SOME @{typ 'a::norm ⇒ real}) ⟫

class real-inner = real-vector + sgn-div-norm + dist-norm + open-dist +
 fixes inner :: 'a ⇒ 'a ⇒ real
 assumes inner-commute: inner x y = inner y x
 and inner-add-left: inner (x + y) z = inner x z + inner y z
 and inner-scaleR-left [simp]: inner (scaleR r x) y = r ∗ (inner x y)
```

and *inner-ge-zero* [*simp*]: $0 \leq inner\ x\ x$
and *inner-eq-zero-iff* [*simp*]: $inner\ x\ x = 0 \longleftrightarrow x = 0$
and *norm-eq-sqrt-inner*: $norm\ x = sqrt\ (inner\ x\ x)$
begin

lemma *inner-zero-left* [*simp*]: $inner\ 0\ x = 0$
using *inner-add-left* [*of 0 0 x*] **by** *simp*

lemma *inner-minus-left* [*simp*]: $inner\ (-\ x)\ y = -\ inner\ x\ y$
using *inner-add-left* [*of x − x y*] **by** *simp*

lemma *inner-diff-left*: $inner\ (x - y)\ z = inner\ x\ z - inner\ y\ z$
by (*simp add*: *diff-minus inner-add-left*)

Transfer distributivity rules to right argument.

lemma *inner-add-right*: $inner\ x\ (y + z) = inner\ x\ y + inner\ x\ z$
using *inner-add-left* [*of y z x*] **by** (*simp only*: *inner-commute*)

lemma *inner-scaleR-right* [*simp*]: $inner\ x\ (scaleR\ r\ y) = r * (inner\ x\ y)$
using *inner-scaleR-left* [*of r y x*] **by** (*simp only*: *inner-commute*)

lemma *inner-zero-right* [*simp*]: $inner\ x\ 0 = 0$
using *inner-zero-left* [*of x*] **by** (*simp only*: *inner-commute*)

lemma *inner-minus-right* [*simp*]: $inner\ x\ (-\ y) = -\ inner\ x\ y$
using *inner-minus-left* [*of y x*] **by** (*simp only*: *inner-commute*)

lemma *inner-diff-right*: $inner\ x\ (y - z) = inner\ x\ y - inner\ x\ z$
using *inner-diff-left* [*of y z x*] **by** (*simp only*: *inner-commute*)

lemmas *inner-add* [*algebra-simps*] = *inner-add-left inner-add-right*
lemmas *inner-diff* [*algebra-simps*] = *inner-diff-left inner-diff-right*
lemmas *inner-scaleR* = *inner-scaleR-left inner-scaleR-right*

Legacy theorem names

lemmas *inner-left-distrib* = *inner-add-left*
lemmas *inner-right-distrib* = *inner-add-right*
lemmas *inner-distrib* = *inner-left-distrib inner-right-distrib*

lemma *inner-gt-zero-iff* [*simp*]: $0 < inner\ x\ x \longleftrightarrow x \neq 0$
by (*simp add*: *order-less-le*)

lemma *power2-norm-eq-inner*: $(norm\ x)^2 = inner\ x\ x$
by (*simp add*: *norm-eq-sqrt-inner*)

lemma *Cauchy-Schwarz-ineq*:
$(inner\ x\ y)^2 \leq inner\ x\ x * inner\ y\ y$
proof (*cases*)
assume $y = 0$

```
  thus ?thesis by simp
next
  assume y: y ≠ 0
  let ?r = inner x y / inner y y
  have 0 ≤ inner (x − scaleR ?r y) (x − scaleR ?r y)
    by (rule inner-ge-zero)
  also have ... = inner x x − inner y x ∗ ?r
    by (simp add: inner-diff)
  also have ... = inner x x − (inner x y)² / inner y y
    by (simp add: power2-eq-square inner-commute)
  finally have 0 ≤ inner x x − (inner x y)² / inner y y .
  hence (inner x y)² / inner y y ≤ inner x x
    by (simp add: le-diff-eq)
  thus (inner x y)² ≤ inner x x ∗ inner y y
    by (simp add: pos-divide-le-eq y)
qed

lemma Cauchy-Schwarz-ineq2:
  |inner x y| ≤ norm x ∗ norm y
proof (rule power2-le-imp-le)
  have (inner x y)² ≤ inner x x ∗ inner y y
    using Cauchy-Schwarz-ineq .
  thus |inner x y|² ≤ (norm x ∗ norm y)²
    by (simp add: power-mult-distrib power2-norm-eq-inner)
  show 0 ≤ norm x ∗ norm y
    unfolding norm-eq-sqrt-inner
    by (intro mult-nonneg-nonneg real-sqrt-ge-zero inner-ge-zero)
qed

subclass real-normed-vector
proof
  fix a :: real and x y :: 'a
  show 0 ≤ norm x
    unfolding norm-eq-sqrt-inner by simp
  show norm x = 0 ⟷ x = 0
    unfolding norm-eq-sqrt-inner by simp
  show norm (x + y) ≤ norm x + norm y
    proof (rule power2-le-imp-le)
      have inner x y ≤ norm x ∗ norm y
        by (rule order-trans [OF abs-ge-self Cauchy-Schwarz-ineq2])
      thus (norm (x + y))² ≤ (norm x + norm y)²
        unfolding power2-sum power2-norm-eq-inner
        by (simp add: inner-add inner-commute)
      show 0 ≤ norm x + norm y
        unfolding norm-eq-sqrt-inner
        by (simp add: add-nonneg-nonneg)
    qed
  have sqrt (a² ∗ inner x x) = |a| ∗ sqrt (inner x x)
    by (simp add: real-sqrt-mult-distrib)
```

then show *norm* $(a *_R x) = |a| * norm\ x$
unfolding *norm-eq-sqrt-inner*
by (*simp add*: *power2-eq-square mult-assoc*)
qed

end

Re-enable constraints for *open*, *dist*, and *norm*.

setup ⟪ *Sign.add-const-constraint*
(@{*const-name open*}, *SOME* @{*typ* 'a::*topological-space set* ⇒ *bool*}) ⟫

setup ⟪ *Sign.add-const-constraint*
(@{*const-name dist*}, *SOME* @{*typ* 'a::*metric-space* ⇒ 'a ⇒ *real*}) ⟫

setup ⟪ *Sign.add-const-constraint*
(@{*const-name norm*}, *SOME* @{*typ* 'a::*real-normed-vector* ⇒ *real*}) ⟫

interpretation *inner*:
bounded-bilinear inner::'a::*real-inner* ⇒ 'a ⇒ *real*
proof
fix *x y z* :: 'a **and** *r* :: *real*
show *inner* $(x + y)\ z = inner\ x\ z + inner\ y\ z$
by (*rule inner-add-left*)
show *inner* $x\ (y + z) = inner\ x\ y + inner\ x\ z$
by (*rule inner-add-right*)
show *inner* $(scaleR\ r\ x)\ y = scaleR\ r\ (inner\ x\ y)$
unfolding *real-scaleR-def* **by** (*rule inner-scaleR-left*)
show *inner* $x\ (scaleR\ r\ y) = scaleR\ r\ (inner\ x\ y)$
unfolding *real-scaleR-def* **by** (*rule inner-scaleR-right*)
show $\exists K.\ \forall x\ y$::'a. $norm\ (inner\ x\ y) \leq norm\ x * norm\ y * K$
proof
show $\forall x\ y$::'a. $norm\ (inner\ x\ y) \leq norm\ x * norm\ y * 1$
by (*simp add*: *Cauchy-Schwarz-ineq2*)
qed
qed

interpretation *inner-left*:
bounded-linear λx::'a::*real-inner*. $inner\ x\ y$
by (*rule inner.bounded-linear-left*)

interpretation *inner-right*:
bounded-linear λy::'a::*real-inner*. $inner\ x\ y$
by (*rule inner.bounded-linear-right*)

5.2 Class instances

instantiation *real* :: *real-inner*
begin

definition *inner-real-def* [*simp*]: *inner* = *op* ∗

instance proof
fix *x y z r* :: *real*
show *inner x y* = *inner y x*
unfolding *inner-real-def* **by** (*rule mult-commute*)
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
unfolding *inner-real-def* **by** (*rule left-distrib*)
show *inner* (*scaleR r x*) *y* = *r* ∗ *inner x y*
unfolding *inner-real-def real-scaleR-def* **by** (*rule mult-assoc*)
show *0* ≤ *inner x x*
unfolding *inner-real-def* **by** *simp*
show *inner x x* = *0* ⟷ *x* = *0*
unfolding *inner-real-def* **by** *simp*
show *norm x* = *sqrt* (*inner x x*)
unfolding *inner-real-def* **by** *simp*
qed

end

instantiation *complex* :: *real-inner*
begin

definition *inner-complex-def*:
inner x y = *Re x* ∗ *Re y* + *Im x* ∗ *Im y*

instance proof
fix *x y z* :: *complex* **and** *r* :: *real*
show *inner x y* = *inner y x*
unfolding *inner-complex-def* **by** (*simp add*: *mult-commute*)
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
unfolding *inner-complex-def* **by** (*simp add*: *left-distrib*)
show *inner* (*scaleR r x*) *y* = *r* ∗ *inner x y*
unfolding *inner-complex-def* **by** (*simp add*: *right-distrib*)
show *0* ≤ *inner x x*
unfolding *inner-complex-def* **by** (*simp add*: *add-nonneg-nonneg*)
show *inner x x* = *0* ⟷ *x* = *0*
unfolding *inner-complex-def*
by (*simp add*: *add-nonneg-eq-0-iff complex-Re-Im-cancel-iff*)
show *norm x* = *sqrt* (*inner x x*)
unfolding *inner-complex-def complex-norm-def*
by (*simp add*: *power2-eq-square*)
qed

end

5.3 Gradient derivative

definition

gderiv ::
  ['*a*::*real-inner* ⇒ *real*, '*a*, '*a*] ⇒ *bool*
      ((*GDERIV* (-)/ (-)/ :> (-)) [*1000*, *1000*, *60*] *60*)
where
  GDERIV f x :> D ⟷ *FDERIV f x :>* (λ*h*. *inner h D*)

lemma *deriv-fderiv*: *DERIV f x :> D* ⟷ *FDERIV f x :>* (λ*h*. *h* ∗ *D*)
  by (*simp only*: *deriv-def field-fderiv-def*)

lemma *gderiv-deriv* [*simp*]: *GDERIV f x :> D* ⟷ *DERIV f x :> D*
  by (*simp only*: *gderiv-def deriv-fderiv inner-real-def*)

lemma *GDERIV-DERIV-compose*:
    ⟦*GDERIV f x :> df*; *DERIV g* (*f x*) :> *dg*⟧
    ⟹ *GDERIV* (λ*x*. *g* (*f x*)) *x :> scaleR dg df*
  unfolding *gderiv-def deriv-fderiv*
  apply (*drule* (*1*) *FDERIV-compose*)
  apply (*simp add*: *mult-ac*)
  done

lemma *FDERIV-subst*: ⟦*FDERIV f x :> df*; *df* = *d*⟧ ⟹ *FDERIV f x :> d*
  by *simp*

lemma *GDERIV-subst*: ⟦*GDERIV f x :> df*; *df* = *d*⟧ ⟹ *GDERIV f x :> d*
  by *simp*

lemma *GDERIV-const*: *GDERIV* (λ*x*. *k*) *x :> 0*
  unfolding *gderiv-def inner-right.zero* **by** (*rule FDERIV-const*)

lemma *GDERIV-add*:
    ⟦*GDERIV f x :> df*; *GDERIV g x :> dg*⟧
    ⟹ *GDERIV* (λ*x*. *f x* + *g x*) *x :> df* + *dg*
  unfolding *gderiv-def inner-right.add* **by** (*rule FDERIV-add*)

lemma *GDERIV-minus*:
    GDERIV f x :> df ⟹ *GDERIV* (λ*x*. − *f x*) *x :> − df*
  unfolding *gderiv-def inner-right.minus* **by** (*rule FDERIV-minus*)

lemma *GDERIV-diff*:
    ⟦*GDERIV f x :> df*; *GDERIV g x :> dg*⟧
    ⟹ *GDERIV* (λ*x*. *f x* − *g x*) *x :> df* − *dg*
  unfolding *gderiv-def inner-right.diff* **by** (*rule FDERIV-diff*)

lemma *GDERIV-scaleR*:
    ⟦*DERIV f x :> df*; *GDERIV g x :> dg*⟧
    ⟹ *GDERIV* (λ*x*. *scaleR* (*f x*) (*g x*)) *x*
     :> (*scaleR* (*f x*) *dg* + *scaleR df* (*g x*))
  unfolding *gderiv-def deriv-fderiv inner-right.add inner-right.scaleR*
  apply (*rule FDERIV-subst*)

```
  apply (erule (1) scaleR.FDERIV)
  apply (simp add: mult-ac)
  done

lemma GDERIV-mult:
    ⟦GDERIV f x :> df; GDERIV g x :> dg⟧
     ⟹ GDERIV (λx. f x * g x) x :> scaleR (f x) dg + scaleR (g x) df
  unfolding gderiv-def
  apply (rule FDERIV-subst)
  apply (erule (1) FDERIV-mult)
  apply (simp add: inner-add mult-ac)
  done

lemma GDERIV-inverse:
    ⟦GDERIV f x :> df; f x ≠ 0⟧
     ⟹ GDERIV (λx. inverse (f x)) x :> - (inverse (f x))² *R df
  apply (erule GDERIV-DERIV-compose)
  apply (erule DERIV-inverse [folded numeral-2-eq-2])
  done

lemma GDERIV-norm:
  assumes x ≠ 0 shows GDERIV (λx. norm x) x :> sgn x
proof −
  have 1: FDERIV (λx. inner x x) x :> (λh. inner x h + inner h x)
    by (intro inner.FDERIV FDERIV-ident)
  have 2: (λh. inner x h + inner h x) = (λh. inner h (scaleR 2 x))
    by (simp add: expand-fun-eq inner-commute)
  have 0 < inner x x using ‹x ≠ 0› by simp
  then have 3: DERIV sqrt (inner x x) :> (inverse (sqrt (inner x x)) / 2)
    by (rule DERIV-real-sqrt)
  have 4: (inverse (sqrt (inner x x)) / 2) *R 2 *R x = sgn x
    by (simp add: sgn-div-norm norm-eq-sqrt-inner)
  show ?thesis
    unfolding norm-eq-sqrt-inner
    apply (rule GDERIV-subst [OF - 4])
    apply (rule GDERIV-DERIV-compose [where g=sqrt and df=scaleR 2 x])
    apply (subst gderiv-def)
    apply (rule FDERIV-subst [OF - 2])
    apply (rule 1)
    apply (rule 3)
    done
qed

lemmas FDERIV-norm = GDERIV-norm [unfolded gderiv-def]

end
```

6 L2-Norm: Square root of sum of squares

theory *L2-Norm*
imports *NthRoot*
begin

definition
$setL2\ f\ A = sqrt\ (\sum i{\in}A.\ (f\ i)^2)$

lemma *setL2-cong*:
$[\![A = B;\ \bigwedge x.\ x \in B \Longrightarrow f\ x = g\ x]\!] \Longrightarrow setL2\ f\ A = setL2\ g\ B$
unfolding *setL2-def* **by** *simp*

lemma *strong-setL2-cong*:
$[\![A = B;\ \bigwedge x.\ x \in B =simp=> f\ x = g\ x]\!] \Longrightarrow setL2\ f\ A = setL2\ g\ B$
unfolding *setL2-def simp-implies-def* **by** *simp*

lemma *setL2-infinite* [*simp*]: $\neg\ finite\ A \Longrightarrow setL2\ f\ A = 0$
unfolding *setL2-def* **by** *simp*

lemma *setL2-empty* [*simp*]: $setL2\ f\ \{\} = 0$
unfolding *setL2-def* **by** *simp*

lemma *setL2-insert* [*simp*]:
$[\![finite\ F;\ a \notin F]\!] \Longrightarrow$
$setL2\ f\ (insert\ a\ F) = sqrt\ ((f\ a)^2 + (setL2\ f\ F)^2)$
unfolding *setL2-def* **by** (*simp add*: *setsum-nonneg*)

lemma *setL2-nonneg* [*simp*]: $0 \leq setL2\ f\ A$
unfolding *setL2-def* **by** (*simp add*: *setsum-nonneg*)

lemma *setL2-0'*: $\forall\ a{\in}A.\ f\ a = 0 \Longrightarrow setL2\ f\ A = 0$
unfolding *setL2-def* **by** *simp*

lemma *setL2-constant*: $setL2\ (\lambda x.\ y)\ A = sqrt\ (of\text{-}nat\ (card\ A)) * |y|$
unfolding *setL2-def* **by** (*simp add*: *real-sqrt-mult*)

lemma *setL2-mono*:
assumes $\bigwedge i.\ i \in K \Longrightarrow f\ i \leq g\ i$
assumes $\bigwedge i.\ i \in K \Longrightarrow 0 \leq f\ i$
shows $setL2\ f\ K \leq setL2\ g\ K$
unfolding *setL2-def*
by (*simp add*: *setsum-nonneg setsum-mono power-mono prems*)

lemma *setL2-strict-mono*:
assumes *finite* K **and** $K \neq \{\}$
assumes $\bigwedge i.\ i \in K \Longrightarrow f\ i < g\ i$
assumes $\bigwedge i.\ i \in K \Longrightarrow 0 \leq f\ i$
shows $setL2\ f\ K < setL2\ g\ K$

unfolding *setL2-def*
by (*simp add*: *setsum-strict-mono power-strict-mono assms*)

lemma *setL2-right-distrib*:
$0 \leq r \Longrightarrow r * setL2\ f\ A = setL2\ (\lambda x.\ r * f\ x)\ A$
unfolding *setL2-def*
apply (*simp add*: *power-mult-distrib*)
apply (*simp add*: *setsum-right-distrib* [*symmetric*])
apply (*simp add*: *real-sqrt-mult setsum-nonneg*)
done

lemma *setL2-left-distrib*:
$0 \leq r \Longrightarrow setL2\ f\ A * r = setL2\ (\lambda x.\ f\ x * r)\ A$
unfolding *setL2-def*
apply (*simp add*: *power-mult-distrib*)
apply (*simp add*: *setsum-left-distrib* [*symmetric*])
apply (*simp add*: *real-sqrt-mult setsum-nonneg*)
done

lemma *setsum-nonneg-eq-0-iff*:
fixes $f :: 'a \Rightarrow 'b$::*ordered-ab-group-add*
shows $[\![finite\ A;\ \forall x{\in}A.\ 0 \leq f\ x]\!] \Longrightarrow setsum\ f\ A = 0 \longleftrightarrow (\forall x{\in}A.\ f\ x = 0)$
apply (*induct set*: *finite*, *simp*)
apply (*simp add*: *add-nonneg-eq-0-iff setsum-nonneg*)
done

lemma *setL2-eq-0-iff*: *finite* $A \Longrightarrow setL2\ f\ A = 0 \longleftrightarrow (\forall x{\in}A.\ f\ x = 0)$
unfolding *setL2-def*
by (*simp add*: *setsum-nonneg setsum-nonneg-eq-0-iff*)

lemma *setL2-triangle-ineq*:
shows $setL2\ (\lambda i.\ f\ i + g\ i)\ A \leq setL2\ f\ A + setL2\ g\ A$
proof (*cases finite A*)
case *False*
thus *?thesis* **by** *simp*
next
case *True*
thus *?thesis*
proof (*induct set*: *finite*)
case *empty*
show *?case* **by** *simp*
next
case (*insert x F*)
hence $sqrt\ ((f\ x + g\ x)^2 + (setL2\ (\lambda i.\ f\ i + g\ i)\ F)^2) \leq$
$sqrt\ ((f\ x + g\ x)^2 + (setL2\ f\ F + setL2\ g\ F)^2)$
by (*intro real-sqrt-le-mono add-left-mono power-mono insert*
setL2-nonneg add-increasing zero-le-power2)
also have
$\ldots \leq sqrt\ ((f\ x)^2 + (setL2\ f\ F)^2) + sqrt\ ((g\ x)^2 + (setL2\ g\ F)^2)$

by (*rule real-sqrt-sum-squares-triangle-ineq*)
finally show *?case*
using *insert* **by** *simp*
qed
qed

lemma *sqrt-sum-squares-le-sum*:
$[\![0 \leq x; 0 \leq y]\!] \Longrightarrow sqrt\ (x^2 + y^2) \leq x + y$
apply (*rule power2-le-imp-le*)
apply (*simp add*: *power2-sum*)
apply (*simp add*: *mult-nonneg-nonneg*)
apply (*simp add*: *add-nonneg-nonneg*)
done

lemma *setL2-le-setsum* [*rule-format*]:
$(\forall i{\in}A.\ 0 \leq f\ i) \longrightarrow setL2\ f\ A \leq setsum\ f\ A$
apply (*cases finite A*)
apply (*induct set*: *finite*)
apply *simp*
apply *clarsimp*
apply (*erule order-trans* [*OF sqrt-sum-squares-le-sum*])
apply *simp*
apply *simp*
apply *simp*
done

lemma *sqrt-sum-squares-le-sum-abs*: $sqrt\ (x^2 + y^2) \leq |x| + |y|$
apply (*rule power2-le-imp-le*)
apply (*simp add*: *power2-sum*)
apply (*simp add*: *mult-nonneg-nonneg*)
apply (*simp add*: *add-nonneg-nonneg*)
done

lemma *setL2-le-setsum-abs*: $setL2\ f\ A \leq (\sum i{\in}A.\ |f\ i|)$
apply (*cases finite A*)
apply (*induct set*: *finite*)
apply *simp*
apply *simp*
apply (*rule order-trans* [*OF sqrt-sum-squares-le-sum-abs*])
apply *simp*
apply *simp*
done

lemma *setL2-mult-ineq-lemma*:
fixes *a b c d* :: *real*
shows $2 * (a * c) * (b * d) \leq a^2 * d^2 + b^2 * c^2$
proof −
have $0 \leq (a * d - b * c)^2$ **by** *simp*
also have $\ldots = a^2 * d^2 + b^2 * c^2 - 2 * (a * d) * (b * c)$

by (*simp only*: *power2-diff power-mult-distrib*)
also have $\ldots = a^2 * d^2 + b^2 * c^2 - 2 * (a * c) * (b * d)$
by *simp*
finally show $2 * (a * c) * (b * d) \leq a^2 * d^2 + b^2 * c^2$
by *simp*
qed

lemma *setL2-mult-ineq*: $(\sum i{\in}A.\ |f\ i| * |g\ i|) \leq$ *setL2 f A* $*$ *setL2 g A*
apply (*cases finite A*)
apply (*induct set*: *finite*)
apply *simp*
apply (*rule power2-le-imp-le*, *simp*)
apply (*rule order-trans*)
apply (*rule power-mono*)
apply (*erule add-left-mono*)
apply (*simp add*: *add-nonneg-nonneg mult-nonneg-nonneg setsum-nonneg*)
apply (*simp add*: *power2-sum*)
apply (*simp add*: *power-mult-distrib*)
apply (*simp add*: *right-distrib left-distrib*)
apply (*rule ord-le-eq-trans*)
apply (*rule setL2-mult-ineq-lemma*)
apply *simp*
apply (*intro mult-nonneg-nonneg setL2-nonneg*)
apply *simp*
done

lemma *member-le-setL2*: $[\![$*finite A*; $i \in A]\!] \Longrightarrow$ *f i* $\leq$ *setL2 f A*
apply (*rule-tac s*=*insert i* $(A - \{i\})$ **and** t=A **in** *subst*)
apply *fast*
apply (*subst setL2-insert*)
apply *simp*
apply *simp*
apply *simp*
done

end

7 Numeral-Type: Numeral Syntax for Types

theory *Numeral-Type*
imports *Main*
begin

7.1 Preliminary lemmas

lemma (**in** *type-definition*) *univ*:
UNIV = *Abs* ' *A*
proof

```
  show Abs ‘ A ⊆ UNIV by (rule subset-UNIV)
  show UNIV ⊆ Abs ‘ A
  proof
    fix x :: 'b
    have x = Abs (Rep x) by (rule Rep-inverse [symmetric])
    moreover have Rep x ∈ A by (rule Rep)
    ultimately show x ∈ Abs ‘ A by (rule image-eqI)
  qed
qed

lemma (in type-definition) card: card (UNIV :: 'b set) = card A
  by (simp add: univ card-image inj-on-def Abs-inject)
```

7.2 Cardinalities of types

```
syntax -type-card :: type => nat ((1CARD/(1'(-'))))

translations CARD('t) => CONST card (CONST UNIV :: 't set)

typed-print-translation ⟪
let
  fun card-univ-tr' show-sorts - [Const (@{const-syntax UNIV}, Type(-, [T, -]))]
=
    Syntax.const @{syntax-const -type-card} $ Syntax.term-of-typ show-sorts T;
in [(@{const-syntax card}, card-univ-tr')]
end
⟫

lemma card-unit [simp]: CARD(unit) = 1
  unfolding UNIV-unit by simp

lemma card-bool [simp]: CARD(bool) = 2
  unfolding UNIV-bool by simp

lemma card-prod [simp]: CARD('a × 'b) = CARD('a::finite) * CARD('b::finite)
 unfolding UNIV-Times-UNIV [symmetric] by (simp only: card-cartesian-product)

lemma card-sum [simp]: CARD('a + 'b) = CARD('a::finite) + CARD('b::finite)
  unfolding UNIV-Plus-UNIV [symmetric] by (simp only: finite card-Plus)

lemma card-option [simp]: CARD('a option) = Suc CARD('a::finite)
  unfolding UNIV-option-conv
  apply (subgoal-tac (None::'a option) ∉ range Some)
  apply (simp add: card-image)
  apply fast
  done

lemma card-set [simp]: CARD('a set) = 2 ^ CARD('a::finite)
  unfolding Pow-UNIV [symmetric]
```

by (*simp only*: *card-Pow finite numeral-2-eq-2*)

lemma *card-nat* [*simp*]: *CARD*(*nat*) = *0*
by (*simp add*: *infinite-UNIV-nat card-eq-0-iff*)

7.3 Classes with at least 1 and 2

Class finite already captures "at least 1"

lemma *zero-less-card-finite* [*simp*]: *0* < *CARD*('*a*::*finite*)
unfolding *neq0-conv* [*symmetric*] **by** *simp*

lemma *one-le-card-finite* [*simp*]: *Suc 0* ≤ *CARD*('*a*::*finite*)
by (*simp add*: *less-Suc-eq-le* [*symmetric*])

Class for cardinality "at least 2"

class *card2* = *finite* +
assumes *two-le-card*: *2* ≤ *CARD*('*a*)

lemma *one-less-card*: *Suc 0* < *CARD*('*a*::*card2*)
using *two-le-card* [**where** *'a*='*a*] **by** *simp*

lemma *one-less-int-card*: *1* < *int CARD*('*a*::*card2*)
using *one-less-card* [**where** *'a*='*a*] **by** *simp*

7.4 Numeral Types

typedef (**open**) *num0* = *UNIV* :: *nat set* **..**
typedef (**open**) *num1* = *UNIV* :: *unit set* **..**

typedef (**open**) *'a bit0* = {*0* ..< *2* ∗ *int CARD*('*a*::*finite*)}
proof
show *0* ∈ {*0* ..< *2* ∗ *int CARD*('*a*)}
by *simp*
qed

typedef (**open**) *'a bit1* = {*0* ..< *1* + *2* ∗ *int CARD*('*a*::*finite*)}
proof
show *0* ∈ {*0* ..< *1* + *2* ∗ *int CARD*('*a*)}
by *simp*
qed

lemma *card-num0* [*simp*]: *CARD* (*num0*) = *0*
unfolding *type-definition.card* [*OF type-definition-num0*]
by *simp*

lemma *card-num1* [*simp*]: *CARD*(*num1*) = *1*
unfolding *type-definition.card* [*OF type-definition-num1*]
by (*simp only*: *card-unit*)

lemma *card-bit0* [*simp*]: $CARD('a\ bit0) = 2 * CARD('a::finite)$
unfolding *type-definition.card* [*OF type-definition-bit0*]
by *simp*

lemma *card-bit1* [*simp*]: $CARD('a\ bit1) = Suc\ (2 * CARD('a::finite))$
unfolding *type-definition.card* [*OF type-definition-bit1*]
by *simp*

instance *num1* :: *finite*
proof
show *finite* (*UNIV*::*num1 set*)
unfolding *type-definition.univ* [*OF type-definition-num1*]
using *finite* **by** (*rule finite-imageI*)
qed

instance *bit0* :: (*finite*) *card2*
proof
show *finite* (*UNIV*::*'a bit0 set*)
unfolding *type-definition.univ* [*OF type-definition-bit0*]
by *simp*
show $2 \leq CARD('a\ bit0)$
by *simp*
qed

instance *bit1* :: (*finite*) *card2*
proof
show *finite* (*UNIV*::*'a bit1 set*)
unfolding *type-definition.univ* [*OF type-definition-bit1*]
by *simp*
show $2 \leq CARD('a\ bit1)$
by *simp*
qed

7.5 Locale for modular arithmetic subtypes

locale *mod-type* =
fixes *n* :: *int*
and *Rep* :: *'a*::{*zero*,*one*,*plus*,*times*,*uminus*,*minus*} ⇒ *int*
and *Abs* :: *int* ⇒ *'a*::{*zero*,*one*,*plus*,*times*,*uminus*,*minus*}
assumes *type*: *type-definition Rep Abs* {*0*..<*n*}
and *size1*: $1 < n$
and *zero-def*: $0 = Abs\ 0$
and *one-def*: $1 = Abs\ 1$
and *add-def*: $x + y = Abs\ ((Rep\ x + Rep\ y)\ mod\ n)$
and *mult-def*: $x * y = Abs\ ((Rep\ x * Rep\ y)\ mod\ n)$
and *diff-def*: $x - y = Abs\ ((Rep\ x - Rep\ y)\ mod\ n)$
and *minus-def*: $- x = Abs\ ((- Rep\ x)\ mod\ n)$
begin

lemma *size0*: $0 < n$
using *size1* **by** *simp*

lemmas *definitions* =
zero-def one-def add-def mult-def minus-def diff-def

lemma *Rep-less-n*: $Rep\ x < n$
by (*rule type-definition.Rep* [*OF type*, *simplified*, *THEN conjunct2*])

lemma *Rep-le-n*: $Rep\ x \leq n$
by (*rule Rep-less-n* [*THEN order-less-imp-le*])

lemma *Rep-inject-sym*: $x = y \longleftrightarrow Rep\ x = Rep\ y$
by (*rule type-definition.Rep-inject* [*OF type*, *symmetric*])

lemma *Rep-inverse*: $Abs\ (Rep\ x) = x$
by (*rule type-definition.Rep-inverse* [*OF type*])

lemma *Abs-inverse*: $m \in \{0..<n\} \Longrightarrow Rep\ (Abs\ m) = m$
by (*rule type-definition.Abs-inverse* [*OF type*])

lemma *Rep-Abs-mod*: $Rep\ (Abs\ (m\ mod\ n)) = m\ mod\ n$
by (*simp add*: *Abs-inverse pos-mod-conj* [*OF size0*])

lemma *Rep-Abs-0*: $Rep\ (Abs\ 0) = 0$
by (*simp add*: *Abs-inverse size0*)

lemma *Rep-0*: $Rep\ 0 = 0$
by (*simp add*: *zero-def Rep-Abs-0*)

lemma *Rep-Abs-1*: $Rep\ (Abs\ 1) = 1$
by (*simp add*: *Abs-inverse size1*)

lemma *Rep-1*: $Rep\ 1 = 1$
by (*simp add*: *one-def Rep-Abs-1*)

lemma *Rep-mod*: $Rep\ x\ mod\ n = Rep\ x$
apply (*rule-tac x*=*x* **in** *type-definition.Abs-cases* [*OF type*])
apply (*simp add*: *type-definition.Abs-inverse* [*OF type*])
apply (*simp add*: *mod-pos-pos-trivial*)
done

lemmas *Rep-simps* =
Rep-inject-sym Rep-inverse Rep-Abs-mod Rep-mod Rep-Abs-0 Rep-Abs-1

lemma *comm-ring-1*: *OFCLASS*('*a*, *comm-ring-1-class*)
apply (*intro-classes*, *unfold definitions*)
apply (*simp-all add*: *Rep-simps zmod-simps field-simps*)
done

end

locale *mod-ring* = *mod-type* +
  constrains *n* :: *int*
  and *Rep* :: *'a*::{*number-ring*} ⇒ *int*
  and *Abs* :: *int* ⇒ *'a*::{*number-ring*}
begin

lemma *of-nat-eq*: *of-nat k* = *Abs* (*int k mod n*)
apply (*induct k*)
apply (*simp add*: *zero-def*)
apply (*simp add*: *Rep-simps add-def one-def zmod-simps add-ac*)
done

lemma *of-int-eq*: *of-int z* = *Abs* (*z mod n*)
apply (*cases z rule*: *int-diff-cases*)
apply (*simp add*: *Rep-simps of-nat-eq diff-def zmod-simps*)
done

lemma *Rep-number-of*:
  Rep (*number-of w*) = *number-of w mod n*
by (*simp add*: *number-of-eq of-int-eq Rep-Abs-mod*)

lemma *iszero-number-of*:
  iszero (*number-of w*::*'a*) ⟷ *number-of w mod n* = *0*
by (*simp add*: *Rep-simps number-of-eq of-int-eq iszero-def zero-def*)

lemma *cases*:
  assumes *1*: ⋀*z*. ⟦(*x*::*'a*) = *of-int z*; *0* ≤ *z*; *z* < *n*⟧ ⟹ *P*
  shows *P*
apply (*cases x rule*: *type-definition.Abs-cases* [*OF type*])
apply (*rule-tac z*=*y* **in** *1*)
apply (*simp-all add*: *of-int-eq mod-pos-pos-trivial*)
done

lemma *induct*:
  (⋀*z*. ⟦*0* ≤ *z*; *z* < *n*⟧ ⟹ *P* (*of-int z*)) ⟹ *P* (*x*::*'a*)
by (*cases x rule*: *cases*) *simp*

end

7.6 Number ring instances

Unfortunately a number ring instance is not possible for *num1*, since 0 and 1 are not distinct.

instantiation *num1* :: {*comm-ring*,*comm-monoid-mult*,*number*}
begin

lemma *num1-eq-iff*: $(x::num1) = (y::num1) \longleftrightarrow True$
by (*induct x*, *induct y*) *simp*

instance proof
qed (*simp-all add*: *num1-eq-iff*)

end

instantiation
bit0 **and** *bit1* :: (*finite*) {*zero*,*one*,*plus*,*times*,*uminus*,*minus*}
begin

definition *Abs-bit0'* :: *int* ⇒ *'a bit0* **where**
Abs-bit0' x = *Abs-bit0* (*x mod int CARD*(*'a bit0*))

definition *Abs-bit1'* :: *int* ⇒ *'a bit1* **where**
Abs-bit1' x = *Abs-bit1* (*x mod int CARD*(*'a bit1*))

definition *0* = *Abs-bit0 0*
definition *1* = *Abs-bit0 1*
definition *x* + *y* = *Abs-bit0'* (*Rep-bit0 x* + *Rep-bit0 y*)
definition *x* ∗ *y* = *Abs-bit0'* (*Rep-bit0 x* ∗ *Rep-bit0 y*)
definition *x* − *y* = *Abs-bit0'* (*Rep-bit0 x* − *Rep-bit0 y*)
definition − *x* = *Abs-bit0'* (− *Rep-bit0 x*)

definition *0* = *Abs-bit1 0*
definition *1* = *Abs-bit1 1*
definition *x* + *y* = *Abs-bit1'* (*Rep-bit1 x* + *Rep-bit1 y*)
definition *x* ∗ *y* = *Abs-bit1'* (*Rep-bit1 x* ∗ *Rep-bit1 y*)
definition *x* − *y* = *Abs-bit1'* (*Rep-bit1 x* − *Rep-bit1 y*)
definition − *x* = *Abs-bit1'* (− *Rep-bit1 x*)

instance ..

end

interpretation *bit0*:
mod-type int CARD(*'a*::*finite bit0*)
Rep-bit0 :: *'a*::*finite bit0* ⇒ *int*
Abs-bit0 :: *int* ⇒ *'a*::*finite bit0*
apply (*rule mod-type.intro*)
apply (*simp add*: *int-mult type-definition-bit0*)
apply (*rule one-less-int-card*)
apply (*rule zero-bit0-def*)
apply (*rule one-bit0-def*)
apply (*rule plus-bit0-def* [*unfolded Abs-bit0'-def*])
apply (*rule times-bit0-def* [*unfolded Abs-bit0'-def*])
apply (*rule minus-bit0-def* [*unfolded Abs-bit0'-def*])
apply (*rule uminus-bit0-def* [*unfolded Abs-bit0'-def*])

done

interpretation *bit1*:
  mod-type int CARD('*a*::*finite bit1*)
    Rep-bit1 :: '*a*::*finite bit1* ⇒ *int*
    Abs-bit1 :: *int* ⇒ '*a*::*finite bit1*
apply (*rule mod-type.intro*)
apply (*simp add*: *int-mult type-definition-bit1*)
apply (*rule one-less-int-card*)
apply (*rule zero-bit1-def*)
apply (*rule one-bit1-def*)
apply (*rule plus-bit1-def* [*unfolded Abs-bit1'-def*])
apply (*rule times-bit1-def* [*unfolded Abs-bit1'-def*])
apply (*rule minus-bit1-def* [*unfolded Abs-bit1'-def*])
apply (*rule uminus-bit1-def* [*unfolded Abs-bit1'-def*])
done

instance *bit0* :: (*finite*) *comm-ring-1*
  by (*rule bit0.comm-ring-1*)+

instance *bit1* :: (*finite*) *comm-ring-1*
  by (*rule bit1.comm-ring-1*)+

instantiation *bit0* **and** *bit1* :: (*finite*) *number-ring*
begin

definition (*number-of w* :: - *bit0*) = *of-int w*

definition (*number-of w* :: - *bit1*) = *of-int w*

instance proof
qed (*rule number-of-bit0-def number-of-bit1-def*)+

end

interpretation *bit0*:
  mod-ring int CARD('*a*::*finite bit0*)
    Rep-bit0 :: '*a*::*finite bit0* ⇒ *int*
    Abs-bit0 :: *int* ⇒ '*a*::*finite bit0*
  ..

interpretation *bit1*:
  mod-ring int CARD('*a*::*finite bit1*)
    Rep-bit1 :: '*a*::*finite bit1* ⇒ *int*
    Abs-bit1 :: *int* ⇒ '*a*::*finite bit1*
  ..

Set up cases, induction, and arithmetic

lemmas *bit0-cases* [*case-names of-int*, *cases type*: *bit0*] = *bit0.cases*

lemmas *bit1-cases* [*case-names of-int*, *cases type*: *bit1*] = *bit1.cases*

lemmas *bit0-induct* [*case-names of-int*, *induct type*: *bit0*] = *bit0.induct*
lemmas *bit1-induct* [*case-names of-int*, *induct type*: *bit1*] = *bit1.induct*

lemmas *bit0-iszero-number-of* [*simp*] = *bit0.iszero-number-of*
lemmas *bit1-iszero-number-of* [*simp*] = *bit1.iszero-number-of*

7.7 Syntax

syntax
-NumeralType :: *num-const* => *type* (-)
-NumeralType0 :: *type* (*0*)
-NumeralType1 :: *type* (*1*)

translations
(*type*) *1* == (*type*) *num1*
(*type*) *0* == (*type*) *num0*

parse-translation ⟪
let
fun mk-bintype n =
let
fun mk-bit 0 = *Syntax.const* @{*type-syntax bit0*}
| *mk-bit 1* = *Syntax.const* @{*type-syntax bit1*};
fun bin-of n =
if n = *1 then Syntax.const* @{*type-syntax num1*}
else if n = *0 then Syntax.const* @{*type-syntax num0*}
else if n = ~*1 then raise TERM* (*negative type numeral*, [])
else
let val (*q*, *r*) = *Integer.div-mod n 2*;
in mk-bit r $ *bin-of q end*;
in bin-of n end;

fun numeral-tr (∗*-NumeralType*∗) [*Const* (*str*, -)] =
mk-bintype (*the* (*Int.fromString str*))
| *numeral-tr* (∗*-NumeralType*∗) *ts* = *raise TERM* (*numeral-tr*, *ts*);

in [(@{*syntax-const -NumeralType*}, *numeral-tr*)] *end*;
⟫

print-translation ⟪
let
fun int-of [] = *0*
| *int-of* (*b* :: *bs*) = *b* + *2* ∗ *int-of bs*;

fun bin-of (*Const* (@{*type-syntax num0*}, -)) = []
| *bin-of* (*Const* (@{*type-syntax num1*}, -)) = [*1*]
| *bin-of* (*Const* (@{*type-syntax bit0*}, -) $ *bs*) = *0* :: *bin-of bs*

```
  | bin-of (Const (@{type-syntax bit1}, -) $ bs) = 1 :: bin-of bs
  | bin-of t = raise TERM (bin-of, [t]);

fun bit-tr' b [t] =
      let
        val rev-digs = b :: bin-of t handle TERM - => raise Match
        val i = int-of rev-digs;
        val num = string-of-int (abs i);
      in
        Syntax.const @{syntax-const -NumeralType} $ Syntax.free num
      end
  | bit-tr' b - = raise Match;
in [(@{type-syntax bit0}, bit-tr' 0), (@{type-syntax bit1}, bit-tr' 1)] end;
⟫
```

7.8 Examples

lemma *CARD(0) = 0* **by** *simp*
lemma *CARD(17) = 17* **by** *simp*
lemma *8 * 11 ^ 3 - 6 = (2::5)* **by** *simp*

end

8 Finite-Cartesian-Product: Definition of finite Cartesian product types.

theory *Finite-Cartesian-Product*
imports *Inner-Product L2-Norm Numeral-Type*
begin

8.1 Finite Cartesian products, with indexing and lambdas.

typedef (**open** *Cart*)
  ('a, 'b) cart = UNIV :: (('b::finite) ⇒ 'a) set
  morphisms *Cart-nth Cart-lambda* **..**

notation
  Cart-nth (**infixl** $ *90*) **and**
  Cart-lambda (**binder** *χ 10*)

syntax *-finite-cart :: type ⇒ type ⇒ type* ((- ^/ -) [*15, 16*] *15*)

parse-translation ⟪
let
  fun cart t u = Syntax.const @{type-syntax cart} $ t $ u;
  fun finite-cart-tr [t, u as Free (x, -)] =
```
```

```
        if Syntax.is-tid x then
       cart t (Syntax.const @{syntax-const -ofsort} $ u $ Syntax.const @{class-syntax
finite})
        else cart t u
    | finite-cart-tr [t, u] = cart t u
in
  [(@{syntax-const -finite-cart}, finite-cart-tr)]
end
⟩⟩

lemma stupid-ext: (∀ x. f x = g x) ⟷ (f = g)
  by (auto intro: ext)

lemma Cart-eq: (x = y) ⟷ (∀ i. x$i = y$i)
  by (simp add: Cart-nth-inject [symmetric] expand-fun-eq)

lemma Cart-lambda-beta [simp]: Cart-lambda g $ i = g i
  by (simp add: Cart-lambda-inverse)

lemma Cart-lambda-unique: (∀ i. f$i = g i) ⟷ Cart-lambda g = f
  by (auto simp add: Cart-eq)

lemma Cart-lambda-eta: (χ i. (g$i)) = g
  by (simp add: Cart-eq)
```

8.2 Group operations and class instances

```
instantiation cart :: (zero,finite) zero
begin
  definition vector-zero-def : 0 ≡ (χ i. 0)
  instance ..
end

instantiation cart :: (plus,finite) plus
begin
  definition  vector-add-def : op + ≡ (λ x y.  (χ i. (x$i) + (y$i)))
  instance ..
end

instantiation cart :: (minus,finite) minus
begin
  definition vector-minus-def : op − ≡ (λ x y.  (χ i. (x$i) − (y$i)))
  instance ..
end

instantiation cart :: (uminus,finite) uminus
begin
  definition vector-uminus-def : uminus ≡ (λ x.  (χ i. − (x$i)))
  instance ..
```

end

lemma *zero-index* [*simp*]: *0* $ *i* = *0*
unfolding *vector-zero-def* **by** *simp*

lemma *vector-add-component* [*simp*]: $(x + y)\$i = x\$i + y\$i$
unfolding *vector-add-def* **by** *simp*

lemma *vector-minus-component* [*simp*]: $(x - y)\$i = x\$i - y\$i$
unfolding *vector-minus-def* **by** *simp*

lemma *vector-uminus-component* [*simp*]: $(- x)\$i = - (x\$i)$
unfolding *vector-uminus-def* **by** *simp*

instance *cart* :: (*semigroup-add*, *finite*) *semigroup-add*
by *default* (*simp add*: *Cart-eq add-assoc*)

instance *cart* :: (*ab-semigroup-add*, *finite*) *ab-semigroup-add*
by *default* (*simp add*: *Cart-eq add-commute*)

instance *cart* :: (*monoid-add*, *finite*) *monoid-add*
by *default* (*simp-all add*: *Cart-eq*)

instance *cart* :: (*comm-monoid-add*, *finite*) *comm-monoid-add*
by *default* (*simp add*: *Cart-eq*)

instance *cart* :: (*cancel-semigroup-add*, *finite*) *cancel-semigroup-add*
by *default* (*simp-all add*: *Cart-eq*)

instance *cart* :: (*cancel-ab-semigroup-add*, *finite*) *cancel-ab-semigroup-add*
by *default* (*simp add*: *Cart-eq*)

instance *cart* :: (*cancel-comm-monoid-add*, *finite*) *cancel-comm-monoid-add* **..**

instance *cart* :: (*group-add*, *finite*) *group-add*
by *default* (*simp-all add*: *Cart-eq diff-minus*)

instance *cart* :: (*ab-group-add*, *finite*) *ab-group-add*
by *default* (*simp-all add*: *Cart-eq*)

8.3 Real vector space

instantiation *cart* :: (*real-vector*, *finite*) *real-vector*
begin

definition *vector-scaleR-def*: *scaleR* = $(\lambda\ r\ x.\ (\chi\ i.\ scaleR\ r\ (x\$i)))$

lemma *vector-scaleR-component* [*simp*]: $(scaleR\ r\ x)\$i = scaleR\ r\ (x\$i)$
unfolding *vector-scaleR-def* **by** *simp*

instance
by *default* (*simp-all add*: *Cart-eq scaleR-left-distrib scaleR-right-distrib*)

end

8.4 Topological space

instantiation *cart* :: (*topological-space*, *finite*) *topological-space*
begin

definition *open-vector-def*:
open (*S* :: ('*a* ^ '*b*) *set*) ⟷
(∀ *x*∈*S*. ∃ *A*. (∀ *i*. *open* (*A i*) ∧ *x*$*i* ∈ *A i*) ∧
(∀ *y*. (∀ *i*. *y*$*i* ∈ *A i*) ⟶ *y* ∈ *S*))

instance proof
show *open* (*UNIV* :: ('*a* ^ '*b*) *set*)
unfolding *open-vector-def* **by** *auto*
next
fix *S T* :: ('*a* ^ '*b*) *set*
assume *open S open T* **thus** *open* (*S* ∩ *T*)
unfolding *open-vector-def*
apply *clarify*
apply (*drule* (*1*) *bspec*)+
apply (*clarify*, *rename-tac Sa Ta*)
apply (*rule-tac x*=λ*i*. *Sa i* ∩ *Ta i* **in** *exI*)
apply (*simp add*: *open-Int*)
done
next
fix *K* :: ('*a* ^ '*b*) *set set*
assume ∀ *S*∈*K*. *open S* **thus** *open* (⋃ *K*)
unfolding *open-vector-def*
apply *clarify*
apply (*drule* (*1*) *bspec*)
apply (*drule* (*1*) *bspec*)
apply *clarify*
apply (*rule-tac x*=*A* **in** *exI*)
apply *fast*
done
qed

end

lemma *open-vector-box*: ∀ *i*. *open* (*S i*) ⟹ *open* {*x*. ∀ *i*. *x* $ *i* ∈ *S i*}
unfolding *open-vector-def* **by** *auto*

lemma *open-vimage-Cart-nth*: *open S* ⟹ *open* ((λ*x*. *x* $ *i*) −‘ *S*)
unfolding *open-vector-def*

apply *clarify*
apply (*rule-tac x*=$\lambda k.$ *if* $k = i$ *then S else UNIV* **in** *exI*, *simp*)
done

lemma *closed-vimage-Cart-nth*: *closed S* $\Longrightarrow$ *closed* $((\lambda x.\ x\ \$\ i) -` S)$
unfolding *closed-open vimage-Compl* [*symmetric*]
by (*rule open-vimage-Cart-nth*)

lemma *closed-vector-box*: $\forall i.$ *closed* $(S\ i) \Longrightarrow$ *closed* $\{x.\ \forall i.\ x\ \$\ i \in S\ i\}$
proof −
  have $\{x.\ \forall i.\ x\ \$\ i \in S\ i\} = (\bigcap i.\ (\lambda x.\ x\ \$\ i) -` S\ i)$ **by** *auto*
  thus $\forall i.$ *closed* $(S\ i) \Longrightarrow$ *closed* $\{x.\ \forall i.\ x\ \$\ i \in S\ i\}$
    by (*simp add*: *closed-INT closed-vimage-Cart-nth*)
qed

lemma *tendsto-Cart-nth* [*tendsto-intros*]:
  assumes $((\lambda x.\ f\ x) \longrightarrow a)$ *net*
  shows $((\lambda x.\ f\ x\ \$\ i) \longrightarrow a\ \$\ i)$ *net*
proof (*rule topological-tendstoI*)
  fix *S* **assume** *open S a* $\$\ i \in S$
  then have *open* $((\lambda y.\ y\ \$\ i) -` S)\ a \in ((\lambda y.\ y\ \$\ i) -` S)$
    by (*simp-all add*: *open-vimage-Cart-nth*)
  with *assms* **have** *eventually* $(\lambda x.\ f\ x \in (\lambda y.\ y\ \$\ i) -` S)$ *net*
    by (*rule topological-tendstoD*)
  then show *eventually* $(\lambda x.\ f\ x\ \$\ i \in S)$ *net*
    by *simp*
qed

lemma *eventually-Ball-finite*:
  assumes *finite A* **and** $\forall y{\in}A.$ *eventually* $(\lambda x.\ P\ x\ y)$ *net*
  shows *eventually* $(\lambda x.\ \forall y{\in}A.\ P\ x\ y)$ *net*
using *assms* **by** (*induct set*: *finite*, *simp*, *simp add*: *eventually-conj*)

lemma *eventually-all-finite*:
  fixes $P :: {'a} \Rightarrow {'b}$::*finite* $\Rightarrow$ *bool*
  assumes $\bigwedge y.$ *eventually* $(\lambda x.\ P\ x\ y)$ *net*
  shows *eventually* $(\lambda x.\ \forall y.\ P\ x\ y)$ *net*
using *eventually-Ball-finite* [*of UNIV P*] *assms* **by** *simp*

lemma *tendsto-vector*:
  assumes $\bigwedge i.$ $((\lambda x.\ f\ x\ \$\ i) \longrightarrow a\ \$\ i)$ *net*
  shows $((\lambda x.\ f\ x) \longrightarrow a)$ *net*
proof (*rule topological-tendstoI*)
  fix *S* **assume** *open S* **and** $a \in S$
  then obtain *A* **where** *A*: $\bigwedge i.$ *open* $(A\ i)$ $\bigwedge i.$ $a\ \$\ i \in A\ i$
    and *S*: $\bigwedge y.\ \forall i.\ y\ \$\ i \in A\ i \Longrightarrow y \in S$
    unfolding *open-vector-def* **by** *metis*
  have $\bigwedge i.$ *eventually* $(\lambda x.\ f\ x\ \$\ i \in A\ i)$ *net*
    using *assms A* **by** (*rule topological-tendstoD*)

hence *eventually* $(\lambda x.\ \forall i.\ f\ x\ \$\ i \in A\ i)$ *net*
by (*rule eventually-all-finite*)
thus *eventually* $(\lambda x.\ f\ x \in S)$ *net*
by (*rule eventually-elim1*, *simp add*: *S*)
qed

lemma *tendsto-Cart-lambda* [*tendsto-intros*]:
assumes $\bigwedge i.\ ((\lambda x.\ f\ x\ i) \longrightarrow a\ i)$ *net*
shows $((\lambda x.\ \chi\ i.\ f\ x\ i) \longrightarrow (\chi\ i.\ a\ i))$ *net*
using *assms* **by** (*simp add*: *tendsto-vector*)

8.5 Metric

lemma *finite-choice*: *finite A* $\Longrightarrow \forall x{\in}A.\ \exists y.\ P\ x\ y \Longrightarrow \exists f.\ \forall x{\in}A.\ P\ x\ (f\ x)$
apply (*induct set*: *finite*, *simp-all*)
apply (*clarify*, *rename-tac y*)
apply (*rule-tac x*=*f*(*x*:=*y*) **in** *exI*, *simp*)
done

instantiation *cart* :: (*metric-space*, *finite*) *metric-space*
begin

definition *dist-vector-def*:
dist x y = *setL2* $(\lambda i.$ *dist* $(x\$i)$ $(y\$i))$ *UNIV*

lemma *dist-nth-le*: *dist* $(x\ \$\ i)$ $(y\ \$\ i) \leq$ *dist x y*
unfolding *dist-vector-def*
by (*rule member-le-setL2*) *simp-all*

instance proof
fix *x y* :: *'a* ^ *'b*
show *dist x y* = *0* $\longleftrightarrow$ *x* = *y*
unfolding *dist-vector-def*
by (*simp add*: *setL2-eq-0-iff Cart-eq*)
next
fix *x y z* :: *'a* ^ *'b*
show *dist x y* $\leq$ *dist x z* + *dist y z*
unfolding *dist-vector-def*
apply (*rule order-trans* [*OF - setL2-triangle-ineq*])
apply (*simp add*: *setL2-mono dist-triangle2*)
done
next

fix *S* :: (*'a* ^ *'b*) *set*
show *open S* $\longleftrightarrow$ $(\forall x{\in}S.\ \exists e{>}0.\ \forall y.$ *dist y x* < *e* $\longrightarrow y \in S)$
unfolding *open-vector-def open-dist*
apply *safe*
apply (*drule* (*1*) *bspec*)
apply *clarify*

```
    apply (subgoal-tac ∃ e>0. ∀ i y. dist y (x$i) < e ⟶ y ∈ A i)
     apply clarify
     apply (rule-tac x=e in exI, clarify)
     apply (drule spec, erule mp, clarify)
     apply (drule spec, drule spec, erule mp)
     apply (erule le-less-trans [OF dist-nth-le])
    apply (subgoal-tac ∀ i∈UNIV. ∃ e>0. ∀ y. dist y (x$i) < e ⟶ y ∈ A i)
     apply (drule finite-choice [OF finite], clarify)
     apply (rule-tac x=Min (range f) in exI, simp)
    apply clarify
    apply (drule-tac x=i in spec, clarify)
    apply (erule (1) bspec)
   apply (drule (1) bspec, clarify)
   apply (subgoal-tac ∃ r. (∀ i::'b. 0 < r i) ∧ e = setL2 r UNIV)
    apply clarify
    apply (rule-tac x=λi. {y. dist y (x$i) < r i} in exI)
    apply (rule conjI)
     apply clarify
     apply (rule conjI)
      apply (clarify, rename-tac y)
      apply (rule-tac x=r i - dist y (x$i) in exI, rule conjI, simp)
      apply clarify
      apply (simp only: less-diff-eq)
      apply (erule le-less-trans [OF dist-triangle])
     apply simp
    apply clarify
    apply (drule spec, erule mp)
    apply (simp add: dist-vector-def setL2-strict-mono)
   apply (rule-tac x=λi. e / sqrt (of-nat CARD('b)) in exI)
   apply (simp add: divide-pos-pos setL2-constant)
   done
qed

end

lemma Cauchy-Cart-nth:
  Cauchy (λn. X n) ⟹ Cauchy (λn. X n $ i)
unfolding Cauchy-def by (fast intro: le-less-trans [OF dist-nth-le])

lemma Cauchy-vector:
  fixes X :: nat ⇒ 'a::metric-space ^ 'n
  assumes X: ⋀i. Cauchy (λn. X n $ i)
  shows Cauchy (λn. X n)
proof (rule metric-CauchyI)
  fix r :: real assume 0 < r
  then have 0 < r / of-nat CARD('n) (is 0 < ?s)
    by (simp add: divide-pos-pos)
  def N ≡ λi. LEAST N. ∀ m≥N. ∀ n≥N. dist (X m $ i) (X n $ i) < ?s
  def M ≡ Max (range N)
```

```
  have ⋀i. ∃N. ∀m≥N. ∀n≥N. dist (X m $ i) (X n $ i) < ?s
    using X ‹0 < ?s› by (rule metric-CauchyD)
  hence ⋀i. ∀m≥N i. ∀n≥N i. dist (X m $ i) (X n $ i) < ?s
    unfolding N-def by (rule LeastI-ex)
  hence M: ⋀i. ∀m≥M. ∀n≥M. dist (X m $ i) (X n $ i) < ?s
    unfolding M-def by simp
  {
    fix m n :: nat
    assume M ≤ m M ≤ n
    have dist (X m) (X n) = setL2 (λi. dist (X m $ i) (X n $ i)) UNIV
      unfolding dist-vector-def ..
    also have ... ≤ setsum (λi. dist (X m $ i) (X n $ i)) UNIV
      by (rule setL2-le-setsum [OF zero-le-dist])
    also have ... < setsum (λi::'n. ?s) UNIV
      by (rule setsum-strict-mono, simp-all add: M ‹M ≤ m› ‹M ≤ n›)
    also have ... = r
      by simp
    finally have dist (X m) (X n) < r .
  }
  hence ∀m≥M. ∀n≥M. dist (X m) (X n) < r
    by simp
  then show ∃M. ∀m≥M. ∀n≥M. dist (X m) (X n) < r ..
qed

instance cart :: (complete-space, finite) complete-space
proof
  fix X :: nat ⇒ 'a ^ 'b assume Cauchy X
  have ⋀i. (λn. X n $ i) ----> lim (λn. X n $ i)
    using Cauchy-Cart-nth [OF ‹Cauchy X›]
    by (simp add: Cauchy-convergent-iff convergent-LIMSEQ-iff)
  hence X ----> Cart-lambda (λi. lim (λn. X n $ i))
    by (simp add: tendsto-vector)
  then show convergent X
    by (rule convergentI)
qed
```

8.6 Normed vector space

```
instantiation cart :: (real-normed-vector, finite) real-normed-vector
begin

definition norm-vector-def:
  norm x = setL2 (λi. norm (x$i)) UNIV

definition vector-sgn-def:
  sgn (x::'a^'b) = scaleR (inverse (norm x)) x

instance proof
  fix a :: real and x y :: 'a ^ 'b
```

show $0 \leq norm\ x$
unfolding *norm-vector-def*
by (*rule setL2-nonneg*)
show $norm\ x = 0 \longleftrightarrow x = 0$
unfolding *norm-vector-def*
by (*simp add*: *setL2-eq-0-iff Cart-eq*)
show $norm\ (x + y) \leq norm\ x + norm\ y$
unfolding *norm-vector-def*
apply (*rule order-trans* [*OF - setL2-triangle-ineq*])
apply (*simp add*: *setL2-mono norm-triangle-ineq*)
done
show $norm\ (scaleR\ a\ x) = |a| * norm\ x$
unfolding *norm-vector-def*
by (*simp add*: *setL2-right-distrib*)
show $sgn\ x = scaleR\ (inverse\ (norm\ x))\ x$
by (*rule vector-sgn-def*)
show $dist\ x\ y = norm\ (x - y)$
unfolding *dist-vector-def norm-vector-def*
by (*simp add*: *dist-norm*)
qed

end

lemma *norm-nth-le*: $norm\ (x\ \$\ i) \leq norm\ x$
unfolding *norm-vector-def*
by (*rule member-le-setL2*) *simp-all*

interpretation *Cart-nth*: *bounded-linear* $\lambda x.\ x\ \$\ i$
apply *default*
apply (*rule vector-add-component*)
apply (*rule vector-scaleR-component*)
apply (*rule-tac x=1* **in** *exI*, *simp add*: *norm-nth-le*)
done

instance *cart* :: (*banach*, *finite*) *banach* **..**

8.7 Inner product space

instantiation *cart* :: (*real-inner*, *finite*) *real-inner*
begin

definition *inner-vector-def*:
$inner\ x\ y = setsum\ (\lambda i.\ inner\ (x\$i)\ (y\$i))\ UNIV$

instance proof
fix r :: *real* **and** $x\ y\ z$ :: $'a\ \hat{}\ 'b$
show $inner\ x\ y = inner\ y\ x$
unfolding *inner-vector-def*
by (*simp add*: *inner-commute*)

show *inner* $(x + y)$ z = *inner* x z + *inner* y z
unfolding *inner-vector-def*
by (*simp add*: *inner-add-left setsum-addf*)
show *inner* (*scaleR* r x) y = r * *inner* x y
unfolding *inner-vector-def*
by (*simp add*: *setsum-right-distrib*)
show $0 \leq$ *inner* x x
unfolding *inner-vector-def*
by (*simp add*: *setsum-nonneg*)
show *inner* x x = $0 \longleftrightarrow$ x = 0
unfolding *inner-vector-def*
by (*simp add*: *Cart-eq setsum-nonneg-eq-0-iff*)
show *norm* x = *sqrt* (*inner* x x)
unfolding *inner-vector-def norm-vector-def setL2-def*
by (*simp add*: *power2-norm-eq-inner*)
qed

end

end

9 Infinite-Set: Infinite Sets and Related Concepts

theory *Infinite-Set*
imports *Main*
begin

9.1 Infinite Sets

Some elementary facts about infinite sets, mostly by Stefan Merz. Beware! Because "infinite" merely abbreviates a negation, these lemmas may not work well with *blast*.

abbreviation
infinite :: $'a$ *set* $\Rightarrow$ *bool* **where**
infinite S == $\neg$ *finite* S

Infinite sets are non-empty, and if we remove some elements from an infinite set, the result is still infinite.

lemma *infinite-imp-nonempty*: *infinite* S ==> $S \neq \{\}$
by *auto*

lemma *infinite-remove*:
infinite $S \Longrightarrow$ *infinite* $(S - \{a\})$
by *simp*

lemma *Diff-infinite-finite*:
assumes T: *finite* T **and** S: *infinite* S

```
  shows infinite (S − T)
  using T
proof induct
  from S
  show infinite (S − {}) by auto
next
  fix T x
  assume ih: infinite (S − T)
  have S − (insert x T) = (S − T) − {x}
    by (rule Diff-insert)
  with ih
  show infinite (S − (insert x T))
    by (simp add: infinite-remove)
qed

lemma Un-infinite: infinite S ⟹ infinite (S ∪ T)
  by simp

lemma infinite-Un: infinite (S ∪ T) ⟷ infinite S ∨ infinite T
  by simp

lemma infinite-super:
  assumes T: S ⊆ T and S: infinite S
  shows infinite T
proof
  assume finite T
  with T have finite S by (simp add: finite-subset)
  with S show False by simp
qed
```

As a concrete example, we prove that the set of natural numbers is infinite.

```
lemma finite-nat-bounded:
  assumes S: finite (S::nat set)
  shows ∃ k. S ⊆ {..<k}  (is ∃ k. ?bounded S k)
using S
proof induct
  have ?bounded {} 0 by simp
  then show ∃ k. ?bounded {} k ..
next
  fix S x
  assume ∃ k. ?bounded S k
  then obtain k where k: ?bounded S k ..
  show ∃ k. ?bounded (insert x S) k
  proof (cases x < k)
    case True
    with k show ?thesis by auto
  next
    case False
    with k have ?bounded S (Suc x) by auto
```

```
    then show ?thesis by auto
  qed
qed

lemma finite-nat-iff-bounded:
  finite (S::nat set) = (∃ k. S ⊆ {..<k})  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs by (rule finite-nat-bounded)
next
  assume ?rhs
  then obtain k where S ⊆ {..<k} ..
  then show finite S
    by (rule finite-subset) simp
qed

lemma finite-nat-iff-bounded-le:
  finite (S::nat set) = (∃ k. S ⊆ {..k})  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain k where S ⊆ {..<k}
    by (blast dest: finite-nat-bounded)
  then have S ⊆ {..k} by auto
  then show ?rhs ..
next
  assume ?rhs
  then obtain k where S ⊆ {..k} ..
  then show finite S
    by (rule finite-subset) simp
qed

lemma infinite-nat-iff-unbounded:
  infinite (S::nat set) = (∀ m. ∃ n. m<n ∧ n∈S)
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  show ?rhs
  proof (rule ccontr)
    assume ¬ ?rhs
    then obtain m where m: ∀ n. m<n ⟶ n∉S by blast
    then have S ⊆ {..m}
      by (auto simp add: sym [OF linorder-not-less])
    with ‹?lhs› show False
      by (simp add: finite-nat-iff-bounded-le)
  qed
next
  assume ?rhs
  show ?lhs
  proof
```

```
    assume finite S
    then obtain m where S ⊆ {..m}
      by (auto simp add: finite-nat-iff-bounded-le)
    then have ∀ n. m<n ⟶ n∉S by auto
    with ‹?rhs› show False by blast
  qed
qed

lemma infinite-nat-iff-unbounded-le:
  infinite (S::nat set) = (∀ m. ∃ n. m≤n ∧ n∈S)
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  show ?rhs
  proof
    fix m
    from ‹?lhs› obtain n where m<n ∧ n∈S
      by (auto simp add: infinite-nat-iff-unbounded)
    then have m≤n ∧ n∈S by simp
    then show ∃ n. m ≤ n ∧ n ∈ S ..
  qed
next
  assume ?rhs
  show ?lhs
  proof (auto simp add: infinite-nat-iff-unbounded)
    fix m
    from ‹?rhs› obtain n where Suc m ≤ n ∧ n∈S
      by blast
    then have m<n ∧ n∈S by simp
    then show ∃ n. m < n ∧ n ∈ S ..
  qed
qed
```

For a set of natural numbers to be infinite, it is enough to know that for any number larger than some k, there is some larger number that is an element of the set.

```
lemma unbounded-k-infinite:
  assumes k: ∀ m. k<m ⟶ (∃ n. m<n ∧ n∈S)
  shows infinite (S::nat set)
proof −
  {
    fix m have ∃ n. m<n ∧ n∈S
    proof (cases k<m)
      case True
      with k show ?thesis by blast
    next
      case False
      from k obtain n where Suc k < n ∧ n∈S by auto
      with False have m<n ∧ n∈S by auto
```

```
    then show ?thesis ..
  qed
 }
 then show ?thesis
  by (auto simp add: infinite-nat-iff-unbounded)
qed


lemma nat-infinite: infinite (UNIV :: nat set)
 by (auto simp add: infinite-nat-iff-unbounded)

lemma nat-not-finite: finite (UNIV::nat set) ⟹ R
 by simp
```

Every infinite set contains a countable subset. More precisely we show that a set S is infinite if and only if there exists an injective function from the naturals into S.

```
lemma range-inj-infinite:
 inj (f::nat ⇒ 'a) ⟹ infinite (range f)
proof
 assume finite (range f) and inj f
 then have finite (UNIV::nat set)
  by (rule finite-imageD)
 then show False by simp
qed

lemma int-infinite [simp]:
 shows infinite (UNIV::int set)
proof −
 from inj-int have infinite (range int) by (rule range-inj-infinite)
 moreover
 have range int ⊆ (UNIV::int set) by simp
 ultimately show infinite (UNIV::int set) by (simp add: infinite-super)
qed
```

The "only if" direction is harder because it requires the construction of a sequence of pairwise different elements of an infinite set S. The idea is to construct a sequence of non-empty and infinite subsets of S obtained by successively removing elements of S.

```
lemma linorder-injI:
 assumes hyp: !!x y. x < (y::'a::linorder) ==> f x ≠ f y
 shows inj f
proof (rule inj-onI)
 fix x y
 assume f-eq: f x = f y
 show x = y
 proof (rule linorder-cases)
  assume x < y
```

```
    with hyp have f x ≠ f y by blast
    with f-eq show ?thesis by simp
  next
    assume x = y
    then show ?thesis .
  next
    assume y < x
    with hyp have f y ≠ f x by blast
    with f-eq show ?thesis by simp
  qed
qed

lemma infinite-countable-subset:
  assumes inf: infinite (S::'a set)
  shows ∃f. inj (f::nat ⇒ 'a) ∧ range f ⊆ S
proof −
  def Sseq ≡ nat-rec S (λn T. T − {SOME e. e ∈ T})
  def pick ≡ λn. (SOME e. e ∈ Sseq n)
  have Sseq-inf: ⋀n. infinite (Sseq n)
  proof −
    fix n
    show infinite (Sseq n)
    proof (induct n)
      from inf show infinite (Sseq 0)
        by (simp add: Sseq-def)
    next
      fix n
      assume infinite (Sseq n) then show infinite (Sseq (Suc n))
        by (simp add: Sseq-def infinite-remove)
    qed
  qed
  have Sseq-S: ⋀n. Sseq n ⊆ S
  proof −
    fix n
    show Sseq n ⊆ S
      by (induct n) (auto simp add: Sseq-def)
  qed
  have Sseq-pick: ⋀n. pick n ∈ Sseq n
  proof −
    fix n
    show pick n ∈ Sseq n
    proof (unfold pick-def, rule someI-ex)
      from Sseq-inf have infinite (Sseq n) .
      then have Sseq n ≠ {} by auto
      then show ∃x. x ∈ Sseq n by auto
    qed
  qed
  with Sseq-S have rng: range pick ⊆ S
    by auto
```

```
  have pick-Sseq-gt: ⋀n m. pick n ∉ Sseq (n + Suc m)
  proof −
    fix n m
    show pick n ∉ Sseq (n + Suc m)
      by (induct m) (auto simp add: Sseq-def pick-def)
  qed
  have pick-pick: ⋀n m. pick n ≠ pick (n + Suc m)
  proof −
    fix n m
    from Sseq-pick have pick (n + Suc m) ∈ Sseq (n + Suc m) .
    moreover from pick-Sseq-gt
    have pick n ∉ Sseq (n + Suc m) .
    ultimately show pick n ≠ pick (n + Suc m)
      by auto
  qed
  have inj: inj pick
  proof (rule linorder-injI)
    fix i j :: nat
    assume i < j
    show pick i ≠ pick j
    proof
      assume eq: pick i = pick j
      from ‹i < j› obtain k where j = i + Suc k
        by (auto simp add: less-iff-Suc-add)
      with pick-pick have pick i ≠ pick j by simp
      with eq show False by simp
    qed
  qed
  from rng inj show ?thesis by auto
qed

lemma infinite-iff-countable-subset:
    infinite S = (∃f. inj (f::nat ⇒ 'a) ∧ range f ⊆ S)
  by (auto simp add: infinite-countable-subset range-inj-infinite infinite-super)
```

For any function with infinite domain and finite range there is some element that is the image of infinitely many domain elements. In particular, any infinite sequence of elements from a finite set contains some element that occurs infinitely often.

```
lemma inf-img-fin-dom:
  assumes img: finite (f‘A) and dom: infinite A
  shows ∃ y ∈ f‘A. infinite (f −‘ {y})
proof (rule ccontr)
  assume ¬ ?thesis
  with img have finite (UN y:f‘A. f −‘ {y}) by (blast intro: finite-UN-I)
  moreover have A ⊆ (UN y:f‘A. f −‘ {y}) by auto
  moreover note dom
  ultimately show False by (simp add: infinite-super)
qed
```

lemma *inf-img-fin-domE*:
assumes *finite* (*f*`*A*) **and** *infinite A*
obtains *y* **where** *y* ∈ *f*`*A* **and** *infinite* (*f* −` {*y*})
using *assms* **by** (*blast dest*: *inf-img-fin-dom*)

9.2 Infinitely Many and Almost All

We often need to reason about the existence of infinitely many (resp., all but finitely many) objects satisfying some predicate, so we introduce corresponding binders and their proof rules.

definition
Inf-many :: ('*a* ⇒ *bool*) ⇒ *bool* (**binder** *INFM* *10*) **where**
Inf-many P = *infinite* {*x*. *P x*}

definition
Alm-all :: ('*a* ⇒ *bool*) ⇒ *bool* (**binder** *MOST* *10*) **where**
Alm-all P = (¬ (*INFM x*. ¬ *P x*))

notation (*xsymbols*)
Inf-many (**binder** ∃∞ *10*) **and**
Alm-all (**binder** ∀∞ *10*)

notation (*HTML* **output**)
Inf-many (**binder** ∃∞ *10*) **and**
Alm-all (**binder** ∀∞ *10*)

lemma *INFM-iff-infinite*: (*INFM x*. *P x*) ⟷ *infinite* {*x*. *P x*}
unfolding *Inf-many-def* **..**

lemma *MOST-iff-cofinite*: (*MOST x*. *P x*) ⟷ *finite* {*x*. ¬ *P x*}
unfolding *Alm-all-def Inf-many-def* **by** *simp*

lemmas *MOST-iff-finiteNeg* = *MOST-iff-cofinite*

lemma *not-INFM* [*simp*]: ¬ (*INFM x*. *P x*) ⟷ (*MOST x*. ¬ *P x*)
unfolding *Alm-all-def not-not* **..**

lemma *not-MOST* [*simp*]: ¬ (*MOST x*. *P x*) ⟷ (*INFM x*. ¬ *P x*)
unfolding *Alm-all-def not-not* **..**

lemma *INFM-const* [*simp*]: (*INFM x*::'*a*. *P*) ⟷ *P* ∧ *infinite* (*UNIV*::'*a set*)
unfolding *Inf-many-def* **by** *simp*

lemma *MOST-const* [*simp*]: (*MOST x*::'*a*. *P*) ⟷ *P* ∨ *finite* (*UNIV*::'*a set*)
unfolding *Alm-all-def* **by** *simp*

lemma *INFM-EX*: (∃∞*x*. *P x*) ⟹ (∃ *x*. *P x*)

by (*erule contrapos-pp*, *simp*)

lemma *ALL-MOST*: $\forall x.\ P\ x \Longrightarrow \forall_\infty x.\ P\ x$
by *simp*

lemma *INFM-E*: **assumes** *INFM x. P x* **obtains** x **where** $P\ x$
using *INFM-EX* [*OF assms*] **by** (*rule exE*)

lemma *MOST-I*: **assumes** $\bigwedge x.\ P\ x$ **shows** *MOST x. P x*
using *assms* **by** *simp*

lemma *INFM-mono*:
assumes *inf*: $\exists_\infty x.\ P\ x$ **and** *q*: $\bigwedge x.\ P\ x \Longrightarrow Q\ x$
shows $\exists_\infty x.\ Q\ x$
proof −
from *inf* **have** *infinite* $\{x.\ P\ x\}$ **unfolding** *Inf-many-def* **.**
moreover from *q* **have** $\{x.\ P\ x\} \subseteq \{x.\ Q\ x\}$ **by** *auto*
ultimately show *?thesis*
by (*simp add*: *Inf-many-def infinite-super*)
qed

lemma *MOST-mono*: $\forall_\infty x.\ P\ x \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow \forall_\infty x.\ Q\ x$
unfolding *Alm-all-def* **by** (*blast intro*: *INFM-mono*)

lemma *INFM-disj-distrib*:
$(\exists_\infty x.\ P\ x \vee Q\ x) \longleftrightarrow (\exists_\infty x.\ P\ x) \vee (\exists_\infty x.\ Q\ x)$
unfolding *Inf-many-def* **by** (*simp add*: *Collect-disj-eq*)

lemma *INFM-imp-distrib*:
$(INFM\ x.\ P\ x \longrightarrow Q\ x) \longleftrightarrow ((MOST\ x.\ P\ x) \longrightarrow (INFM\ x.\ Q\ x))$
by (*simp only*: *imp-conv-disj INFM-disj-distrib not-MOST*)

lemma *MOST-conj-distrib*:
$(\forall_\infty x.\ P\ x \wedge Q\ x) \longleftrightarrow (\forall_\infty x.\ P\ x) \wedge (\forall_\infty x.\ Q\ x)$
unfolding *Alm-all-def* **by** (*simp add*: *INFM-disj-distrib del*: *disj-not1*)

lemma *MOST-conjI*:
$MOST\ x.\ P\ x \Longrightarrow MOST\ x.\ Q\ x \Longrightarrow MOST\ x.\ P\ x \wedge Q\ x$
by (*simp add*: *MOST-conj-distrib*)

lemma *INFM-conjI*:
$INFM\ x.\ P\ x \Longrightarrow MOST\ x.\ Q\ x \Longrightarrow INFM\ x.\ P\ x \wedge Q\ x$
unfolding *MOST-iff-cofinite INFM-iff-infinite*
apply (*drule* (*1*) *Diff-infinite-finite*)
apply (*simp add*: *Collect-conj-eq Collect-neg-eq*)
done

lemma *MOST-rev-mp*:
assumes $\forall_\infty x.\ P\ x$ **and** $\forall_\infty x.\ P\ x \longrightarrow Q\ x$

shows $\forall_\infty x.\ Q\ x$
proof −
have $\forall_\infty x.\ P\ x \wedge (P\ x \longrightarrow Q\ x)$
using *assms* **by** (*rule MOST-conjI*)
thus *?thesis* **by** (*rule MOST-mono*) *simp*
qed

lemma *MOST-imp-iff*:
assumes $MOST\ x.\ P\ x$
shows $(MOST\ x.\ P\ x \longrightarrow Q\ x) \longleftrightarrow (MOST\ x.\ Q\ x)$
proof
assume $MOST\ x.\ P\ x \longrightarrow Q\ x$
with *assms* **show** $MOST\ x.\ Q\ x$ **by** (*rule MOST-rev-mp*)
next
assume $MOST\ x.\ Q\ x$
then show $MOST\ x.\ P\ x \longrightarrow Q\ x$ **by** (*rule MOST-mono*) *simp*
qed

lemma *INFM-MOST-simps* [*simp*]:
$\bigwedge P\ Q.\ (INFM\ x.\ P\ x \wedge Q) \longleftrightarrow (INFM\ x.\ P\ x) \wedge Q$
$\bigwedge P\ Q.\ (INFM\ x.\ P \wedge Q\ x) \longleftrightarrow P \wedge (INFM\ x.\ Q\ x)$
$\bigwedge P\ Q.\ (MOST\ x.\ P\ x \vee Q) \longleftrightarrow (MOST\ x.\ P\ x) \vee Q$
$\bigwedge P\ Q.\ (MOST\ x.\ P \vee Q\ x) \longleftrightarrow P \vee (MOST\ x.\ Q\ x)$
$\bigwedge P\ Q.\ (MOST\ x.\ P\ x \longrightarrow Q) \longleftrightarrow ((INFM\ x.\ P\ x) \longrightarrow Q)$
$\bigwedge P\ Q.\ (MOST\ x.\ P \longrightarrow Q\ x) \longleftrightarrow (P \longrightarrow (MOST\ x.\ Q\ x))$
unfolding *Alm-all-def Inf-many-def*
by (*simp-all add*: *Collect-conj-eq*)

Properties of quantifiers with injective functions.

lemma *INFM-inj*:
$INFM\ x.\ P\ (f\ x) \Longrightarrow inj\ f \Longrightarrow INFM\ x.\ P\ x$
unfolding *INFM-iff-infinite*
by (*clarify*, *drule* (*1*) *finite-vimageI*, *simp*)

lemma *MOST-inj*:
$MOST\ x.\ P\ x \Longrightarrow inj\ f \Longrightarrow MOST\ x.\ P\ (f\ x)$
unfolding *MOST-iff-cofinite*
by (*drule* (*1*) *finite-vimageI*, *simp*)

Properties of quantifiers with singletons.

lemma *not-INFM-eq* [*simp*]:
$\neg\ (INFM\ x.\ x = a)$
$\neg\ (INFM\ x.\ a = x)$
unfolding *INFM-iff-infinite* **by** *simp-all*

lemma *MOST-neq* [*simp*]:
$MOST\ x.\ x \neq a$
$MOST\ x.\ a \neq x$
unfolding *MOST-iff-cofinite* **by** *simp-all*

lemma *INFM-neq* [*simp*]:
(*INFM x*::'*a*. *x* ≠ *a*) ⟷ *infinite* (*UNIV*::'*a set*)
(*INFM x*::'*a*. *a* ≠ *x*) ⟷ *infinite* (*UNIV*::'*a set*)
unfolding *INFM-iff-infinite* **by** *simp-all*

lemma *MOST-eq* [*simp*]:
(*MOST x*::'*a*. *x* = *a*) ⟷ *finite* (*UNIV*::'*a set*)
(*MOST x*::'*a*. *a* = *x*) ⟷ *finite* (*UNIV*::'*a set*)
unfolding *MOST-iff-cofinite* **by** *simp-all*

lemma *MOST-eq-imp*:
MOST x. *x* = *a* ⟶ *P x*
MOST x. *a* = *x* ⟶ *P x*
unfolding *MOST-iff-cofinite* **by** *simp-all*

Properties of quantifiers over the naturals.

lemma *INFM-nat*: (∃ ∞*n*. *P* (*n*::*nat*)) = (∀ *m*. ∃ *n*. *m*<*n* ∧ *P n*)
by (*simp add*: *Inf-many-def infinite-nat-iff-unbounded*)

lemma *INFM-nat-le*: (∃ ∞*n*. *P* (*n*::*nat*)) = (∀ *m*. ∃ *n*. *m*≤*n* ∧ *P n*)
by (*simp add*: *Inf-many-def infinite-nat-iff-unbounded-le*)

lemma *MOST-nat*: (∀ ∞*n*. *P* (*n*::*nat*)) = (∃ *m*. ∀ *n*. *m*<*n* ⟶ *P n*)
by (*simp add*: *Alm-all-def INFM-nat*)

lemma *MOST-nat-le*: (∀ ∞*n*. *P* (*n*::*nat*)) = (∃ *m*. ∀ *n*. *m*≤*n* ⟶ *P n*)
by (*simp add*: *Alm-all-def INFM-nat-le*)

9.3 Enumeration of an Infinite Set

The set's element type must be wellordered (e.g. the natural numbers).

primrec (**in** *wellorder*) *enumerate* :: '*a set* ⇒ *nat* ⇒ '*a* **where**
enumerate-0: *enumerate S 0* = (*LEAST n*. *n* ∈ *S*)
| *enumerate-Suc*: *enumerate S* (*Suc n*) = *enumerate* (*S* − {*LEAST n*. *n* ∈ *S*}) *n*

lemma *enumerate-Suc'*:
enumerate S (*Suc n*) = *enumerate* (*S* − {*enumerate S 0*}) *n*
by *simp*

lemma *enumerate-in-set*: *infinite S* ⟹ *enumerate S n* : *S*
apply (*induct n arbitrary*: *S*)
apply (*fastsimp intro*: *LeastI dest*!: *infinite-imp-nonempty*)
apply *simp*
apply (*metis Collect-def Collect-mem-eq DiffE infinite-remove*)
done

declare *enumerate-0* [*simp del*] *enumerate-Suc* [*simp del*]

lemma *enumerate-step*: *infinite S* $\Longrightarrow$ *enumerate S n* $<$ *enumerate S* (*Suc n*)
apply (*induct n arbitrary*: *S*)
apply (*rule order-le-neq-trans*)
apply (*simp add*: *enumerate-0 Least-le enumerate-in-set*)
apply (*simp only*: *enumerate-Suc'*)
apply (*subgoal-tac enumerate* (*S* $-$ {*enumerate S 0*}) *0* $:$ *S* $-$ {*enumerate S 0*})
apply (*blast intro*: *sym*)
apply (*simp add*: *enumerate-in-set del*: *Diff-iff*)
apply (*simp add*: *enumerate-Suc'*)
done

lemma *enumerate-mono*: *m*$<$*n* $\Longrightarrow$ *infinite S* $\Longrightarrow$ *enumerate S m* $<$ *enumerate S n*
apply (*erule less-Suc-induct*)
apply (*auto intro*: *enumerate-step*)
done

9.4 Miscellaneous

A few trivial lemmas about sets that contain at most one element. These simplify the reasoning about deterministic automata.

definition
atmost-one :: *'a set* $\Rightarrow$ *bool* **where**
atmost-one S = ($\forall$ *x y*. *x*$\in$*S* $\wedge$ *y*$\in$*S* $\longrightarrow$ *x*=*y*)

lemma *atmost-one-empty*: *S* = {} $\Longrightarrow$ *atmost-one S*
by (*simp add*: *atmost-one-def*)

lemma *atmost-one-singleton*: *S* = {*x*} $\Longrightarrow$ *atmost-one S*
by (*simp add*: *atmost-one-def*)

lemma *atmost-one-unique* [*elim*]: *atmost-one S* $\Longrightarrow$ *x* $\in$ *S* $\Longrightarrow$ *y* $\in$ *S* $\Longrightarrow$ *y* = *x*
by (*simp add*: *atmost-one-def*)

end

10 Product-plus: Additive group operations on product types

theory *Product-plus*
imports *Main*
begin

10.1 Operations

instantiation $*$ :: (*zero*, *zero*) *zero*
begin

definition *zero-prod-def*: $0 = (0, 0)$

instance ..
end

instantiation $*$ :: (*plus*, *plus*) *plus*
begin

definition *plus-prod-def*:
$x + y = (fst\ x + fst\ y, snd\ x + snd\ y)$

instance ..
end

instantiation $*$ :: (*minus*, *minus*) *minus*
begin

definition *minus-prod-def*:
$x - y = (fst\ x - fst\ y, snd\ x - snd\ y)$

instance ..
end

instantiation $*$ :: (*uminus*, *uminus*) *uminus*
begin

definition *uminus-prod-def*:
$- x = (- fst\ x, - snd\ x)$

instance ..
end

lemma *fst-zero* [*simp*]: $fst\ 0 = 0$
unfolding *zero-prod-def* **by** *simp*

lemma *snd-zero* [*simp*]: $snd\ 0 = 0$
unfolding *zero-prod-def* **by** *simp*

lemma *fst-add* [*simp*]: $fst\ (x + y) = fst\ x + fst\ y$
unfolding *plus-prod-def* **by** *simp*

lemma *snd-add* [*simp*]: $snd\ (x + y) = snd\ x + snd\ y$
unfolding *plus-prod-def* **by** *simp*

lemma *fst-diff* [*simp*]: $fst\ (x - y) = fst\ x - fst\ y$

unfolding *minus-prod-def* **by** *simp*

lemma *snd-diff* [*simp*]: *snd* $(x - y) =$ *snd* $x -$ *snd* y
unfolding *minus-prod-def* **by** *simp*

lemma *fst-uminus* [*simp*]: *fst* $(- x) = -$ *fst* x
unfolding *uminus-prod-def* **by** *simp*

lemma *snd-uminus* [*simp*]: *snd* $(- x) = -$ *snd* x
unfolding *uminus-prod-def* **by** *simp*

lemma *add-Pair* [*simp*]: $(a, b) + (c, d) = (a + c, b + d)$
unfolding *plus-prod-def* **by** *simp*

lemma *diff-Pair* [*simp*]: $(a, b) - (c, d) = (a - c, b - d)$
unfolding *minus-prod-def* **by** *simp*

lemma *uminus-Pair* [*simp*, *code*]: $- (a, b) = (- a, - b)$
unfolding *uminus-prod-def* **by** *simp*

lemmas *expand-prod-eq* = *Pair-fst-snd-eq*

10.2 Class instances

instance $*$ :: (*semigroup-add*, *semigroup-add*) *semigroup-add*
by *default* (*simp add*: *expand-prod-eq add-assoc*)

instance $*$ :: (*ab-semigroup-add*, *ab-semigroup-add*) *ab-semigroup-add*
by *default* (*simp add*: *expand-prod-eq add-commute*)

instance $*$ :: (*monoid-add*, *monoid-add*) *monoid-add*
by *default* (*simp-all add*: *expand-prod-eq*)

instance $*$ :: (*comm-monoid-add*, *comm-monoid-add*) *comm-monoid-add*
by *default* (*simp add*: *expand-prod-eq*)

instance $*$ ::
(*cancel-semigroup-add*, *cancel-semigroup-add*) *cancel-semigroup-add*
by *default* (*simp-all add*: *expand-prod-eq*)

instance $*$ ::
(*cancel-ab-semigroup-add*, *cancel-ab-semigroup-add*) *cancel-ab-semigroup-add*
by *default* (*simp add*: *expand-prod-eq*)

instance $*$ ::
(*cancel-comm-monoid-add*, *cancel-comm-monoid-add*) *cancel-comm-monoid-add*
..

instance $*$ :: (*group-add*, *group-add*) *group-add*

by *default* (*simp-all add*: *expand-prod-eq diff-minus*)

instance $*$ :: (*ab-group-add*, *ab-group-add*) *ab-group-add*
by *default* (*simp-all add*: *expand-prod-eq*)

lemma *fst-setsum*: *fst* $(\sum x{\in}A.\ f\ x) = (\sum x{\in}A.\ fst\ (f\ x))$
by (*cases finite A*, *induct set*: *finite*, *simp-all*)

lemma *snd-setsum*: *snd* $(\sum x{\in}A.\ f\ x) = (\sum x{\in}A.\ snd\ (f\ x))$
by (*cases finite A*, *induct set*: *finite*, *simp-all*)

end

11 Product-Vector: Cartesian Products as Vector Spaces

theory *Product-Vector*
imports *Inner-Product Product-plus*
begin

11.1 Product is a real vector space

instantiation $*$ :: (*real-vector*, *real-vector*) *real-vector*
begin

definition *scaleR-prod-def*:
scaleR r A = (*scaleR r* (*fst A*), *scaleR r* (*snd A*))

lemma *fst-scaleR* [*simp*]: *fst* (*scaleR r A*) = *scaleR r* (*fst A*)
unfolding *scaleR-prod-def* **by** *simp*

lemma *snd-scaleR* [*simp*]: *snd* (*scaleR r A*) = *scaleR r* (*snd A*)
unfolding *scaleR-prod-def* **by** *simp*

lemma *scaleR-Pair* [*simp*]: *scaleR r* (*a*, *b*) = (*scaleR r a*, *scaleR r b*)
unfolding *scaleR-prod-def* **by** *simp*

instance proof
fix *a b* :: *real* **and** *x y* :: $'a \times 'b$
show *scaleR a* (*x* + *y*) = *scaleR a x* + *scaleR a y*
by (*simp add*: *expand-prod-eq scaleR-right-distrib*)
show *scaleR* (*a* + *b*) *x* = *scaleR a x* + *scaleR b x*
by (*simp add*: *expand-prod-eq scaleR-left-distrib*)
show *scaleR a* (*scaleR b x*) = *scaleR* (*a* $*$ *b*) *x*
by (*simp add*: *expand-prod-eq*)
show *scaleR 1 x* = *x*
by (*simp add*: *expand-prod-eq*)

qed

end

11.2 Product is a topological space

instantiation
$*$ :: (*topological-space*, *topological-space*) *topological-space*
begin

definition *open-prod-def*:
open (S :: ($'a \times 'b$) *set*) $\longleftrightarrow$
($\forall x{\in}S.\ \exists A\ B.$ *open* $A \wedge$ *open* $B \wedge x \in A \times B \wedge A \times B \subseteq S$)

lemma *open-prod-elim*:
assumes *open* S **and** $x \in S$
obtains $A\ B$ **where** *open* A **and** *open* B **and** $x \in A \times B$ **and** $A \times B \subseteq S$
using *assms* **unfolding** *open-prod-def* **by** *fast*

lemma *open-prod-intro*:
assumes $\bigwedge x.\ x \in S \Longrightarrow \exists A\ B.$ *open* $A \wedge$ *open* $B \wedge x \in A \times B \wedge A \times B \subseteq S$
shows *open* S
using *assms* **unfolding** *open-prod-def* **by** *fast*

instance proof
show *open* (*UNIV* :: ($'a \times 'b$) *set*)
unfolding *open-prod-def* **by** *auto*
next
fix $S\ T$ :: ($'a \times 'b$) *set*
assume *open* S *open* T
show *open* ($S \cap T$)
proof (*rule open-prod-intro*)
fix x **assume** x: $x \in S \cap T$
from x **have** $x \in S$ **by** *simp*
obtain *Sa Sb* **where** A: *open Sa open Sb* $x \in Sa \times Sb$ $Sa \times Sb \subseteq S$
using ‹*open* S› **and** ‹$x \in S$› **by** (*rule open-prod-elim*)
from x **have** $x \in T$ **by** *simp*
obtain *Ta Tb* **where** B: *open Ta open Tb* $x \in Ta \times Tb$ $Ta \times Tb \subseteq T$
using ‹*open* T› **and** ‹$x \in T$› **by** (*rule open-prod-elim*)
let *?A* $= Sa \cap Ta$ **and** *?B* $= Sb \cap Tb$
have *open ?A* $\wedge$ *open ?B* $\wedge\ x \in$ *?A* $\times$ *?B* $\wedge$ *?A* $\times$ *?B* $\subseteq S \cap T$
using $A\ B$ **by** (*auto simp add*: *open-Int*)
thus $\exists A\ B.$ *open* $A \wedge$ *open* $B \wedge x \in A \times B \wedge A \times B \subseteq S \cap T$
by *fast*
qed
next
fix K :: ($'a \times 'b$) *set set*
assume $\forall S{\in}K.$ *open* S **thus** *open* ($\bigcup K$)
unfolding *open-prod-def* **by** *fast*

qed

end

lemma *open-Times*: *open S* $\Longrightarrow$ *open T* $\Longrightarrow$ *open* (*S* $\times$ *T*)
unfolding *open-prod-def* **by** *auto*

lemma *fst-vimage-eq-Times*: *fst* $-`$ *S* = *S* $\times$ *UNIV*
by *auto*

lemma *snd-vimage-eq-Times*: *snd* $-`$ *S* = *UNIV* $\times$ *S*
by *auto*

lemma *open-vimage-fst*: *open S* $\Longrightarrow$ *open* (*fst* $-`$ *S*)
by (*simp add*: *fst-vimage-eq-Times open-Times*)

lemma *open-vimage-snd*: *open S* $\Longrightarrow$ *open* (*snd* $-`$ *S*)
by (*simp add*: *snd-vimage-eq-Times open-Times*)

lemma *closed-vimage-fst*: *closed S* $\Longrightarrow$ *closed* (*fst* $-`$ *S*)
unfolding *closed-open vimage-Compl* [*symmetric*]
by (*rule open-vimage-fst*)

lemma *closed-vimage-snd*: *closed S* $\Longrightarrow$ *closed* (*snd* $-`$ *S*)
unfolding *closed-open vimage-Compl* [*symmetric*]
by (*rule open-vimage-snd*)

lemma *closed-Times*: *closed S* $\Longrightarrow$ *closed T* $\Longrightarrow$ *closed* (*S* $\times$ *T*)
proof −
  have *S* $\times$ *T* = (*fst* $-`$ *S*) $\cap$ (*snd* $-`$ *T*) **by** *auto*
  thus *closed S* $\Longrightarrow$ *closed T* $\Longrightarrow$ *closed* (*S* $\times$ *T*)
    by (*simp add*: *closed-vimage-fst closed-vimage-snd closed-Int*)
qed

lemma *openI*:
  assumes $\bigwedge$*x*. *x* $\in$ *S* $\Longrightarrow$ $\exists$ *T*. *open T* $\wedge$ *x* $\in$ *T* $\wedge$ *T* $\subseteq$ *S*
  shows *open S*
proof −
  have *open* ($\bigcup$ {*T*. *open T* $\wedge$ *T* $\subseteq$ *S*}) **by** *auto*
  moreover have $\bigcup$ {*T*. *open T* $\wedge$ *T* $\subseteq$ *S*} = *S* **by** (*auto dest*!: *assms*)
  ultimately show *open S* **by** *simp*
qed

lemma *subset-fst-imageI*: *A* $\times$ *B* $\subseteq$ *S* $\Longrightarrow$ *y* $\in$ *B* $\Longrightarrow$ *A* $\subseteq$ *fst* ` *S*
  unfolding *image-def subset-eq* **by** *force*

lemma *subset-snd-imageI*: *A* $\times$ *B* $\subseteq$ *S* $\Longrightarrow$ *x* $\in$ *A* $\Longrightarrow$ *B* $\subseteq$ *snd* ` *S*
  unfolding *image-def subset-eq* **by** *force*

lemma *open-image-fst*: **assumes** *open S* **shows** *open* (*fst ' S*)
proof (*rule openI*)
  fix *x* **assume** *x* ∈ *fst ' S*
  then obtain *y* **where** (*x*, *y*) ∈ *S* **by** *auto*
  then obtain *A B* **where** *open A open B x* ∈ *A y* ∈ *B A* × *B* ⊆ *S*
    using ‹*open S*› **unfolding** *open-prod-def* **by** *auto*
  from ‹*A* × *B* ⊆ *S*› ‹*y* ∈ *B*› **have** *A* ⊆ *fst ' S* **by** (*rule subset-fst-imageI*)
  with ‹*open A*› ‹*x* ∈ *A*› **have** *open A* ∧ *x* ∈ *A* ∧ *A* ⊆ *fst ' S* **by** *simp*
  then show ∃ *T*. *open T* ∧ *x* ∈ *T* ∧ *T* ⊆ *fst ' S* **by** − (*rule exI*)
qed

lemma *open-image-snd*: **assumes** *open S* **shows** *open* (*snd ' S*)
proof (*rule openI*)
  fix *y* **assume** *y* ∈ *snd ' S*
  then obtain *x* **where** (*x*, *y*) ∈ *S* **by** *auto*
  then obtain *A B* **where** *open A open B x* ∈ *A y* ∈ *B A* × *B* ⊆ *S*
    using ‹*open S*› **unfolding** *open-prod-def* **by** *auto*
  from ‹*A* × *B* ⊆ *S*› ‹*x* ∈ *A*› **have** *B* ⊆ *snd ' S* **by** (*rule subset-snd-imageI*)
  with ‹*open B*› ‹*y* ∈ *B*› **have** *open B* ∧ *y* ∈ *B* ∧ *B* ⊆ *snd ' S* **by** *simp*
  then show ∃ *T*. *open T* ∧ *y* ∈ *T* ∧ *T* ⊆ *snd ' S* **by** − (*rule exI*)
qed

11.3 Product is a metric space

instantiation
  ∗ :: (*metric-space*, *metric-space*) *metric-space*
begin

definition *dist-prod-def*:
  dist (*x*::*'a* × *'b*) *y* = *sqrt* ((*dist* (*fst x*) (*fst y*))^2 + (*dist* (*snd x*) (*snd y*))^2)

lemma *dist-Pair-Pair*: *dist* (*a*, *b*) (*c*, *d*) = *sqrt* ((*dist a c*)^2 + (*dist b d*)^2)
  unfolding *dist-prod-def* **by** *simp*

lemma *dist-fst-le*: *dist* (*fst x*) (*fst y*) ≤ *dist x y*
unfolding *dist-prod-def* **by** (*rule real-sqrt-sum-squares-ge1*)

lemma *dist-snd-le*: *dist* (*snd x*) (*snd y*) ≤ *dist x y*
unfolding *dist-prod-def* **by** (*rule real-sqrt-sum-squares-ge2*)

instance proof
  fix *x y* :: *'a* × *'b*
  show *dist x y* = *0* ⟷ *x* = *y*
    unfolding *dist-prod-def expand-prod-eq* **by** *simp*
next
  fix *x y z* :: *'a* × *'b*
  show *dist x y* ≤ *dist x z* + *dist y z*
    unfolding *dist-prod-def*
    by (*intro order-trans* [*OF - real-sqrt-sum-squares-triangle-ineq*]

real-sqrt-le-mono add-mono power-mono dist-triangle2 zero-le-dist)
next

fix $S :: ('a \times 'b)$ *set*
show *open* $S \longleftrightarrow (\forall x{\in}S.\ \exists e{>}0.\ \forall y.\ dist\ y\ x < e \longrightarrow y \in S)$
proof
assume *open* S **show** $\forall x{\in}S.\ \exists e{>}0.\ \forall y.\ dist\ y\ x < e \longrightarrow y \in S$
proof
fix x **assume** $x \in S$
obtain $A\ B$ **where** *open* A *open* B $x \in A \times B$ $A \times B \subseteq S$
using ‹*open* S› **and** ‹$x \in S$› **by** (*rule open-prod-elim*)
obtain r **where** $r{:}\ 0 < r\ \forall y.\ dist\ y\ (fst\ x) < r \longrightarrow y \in A$
using ‹*open* A› **and** ‹$x \in A \times B$› **unfolding** *open-dist* **by** *auto*
obtain s **where** $s{:}\ 0 < s\ \forall y.\ dist\ y\ (snd\ x) < s \longrightarrow y \in B$
using ‹*open* B› **and** ‹$x \in A \times B$› **unfolding** *open-dist* **by** *auto*
let *?e* = *min* $r\ s$
have $0 < ?e \wedge (\forall y.\ dist\ y\ x < ?e \longrightarrow y \in S)$
proof (*intro allI impI conjI*)
show $0 < min\ r\ s$ **by** (*simp add*: $r(1)\ s(1)$)
next
fix y **assume** $dist\ y\ x < min\ r\ s$
hence $dist\ y\ x < r$ **and** $dist\ y\ x < s$
by *simp-all*
hence $dist\ (fst\ y)\ (fst\ x) < r$ **and** $dist\ (snd\ y)\ (snd\ x) < s$
by (*auto intro*: *le-less-trans dist-fst-le dist-snd-le*)
hence $fst\ y \in A$ **and** $snd\ y \in B$
by (*simp-all add*: $r(2)\ s(2)$)
hence $y \in A \times B$ **by** (*induct y*, *simp*)
with ‹$A \times B \subseteq S$› **show** $y \in S$ **..**
qed
thus $\exists e{>}0.\ \forall y.\ dist\ y\ x < e \longrightarrow y \in S$ **..**
qed
next
assume $\forall x{\in}S.\ \exists e{>}0.\ \forall y.\ dist\ y\ x < e \longrightarrow y \in S$ **thus** *open* S
unfolding *open-prod-def open-dist*
apply *safe*
apply (*drule* (*1*) *bspec*)
apply *clarify*
apply (*subgoal-tac* $\exists r{>}0.\ \exists s{>}0.\ e = sqrt\ (r^2 + s^2)$)
apply *clarify*
apply (*rule-tac* $x{=}\{y.\ dist\ y\ a < r\}$ **in** *exI*)
apply (*rule-tac* $x{=}\{y.\ dist\ y\ b < s\}$ **in** *exI*)
apply (*rule conjI*)
apply *clarify*
apply (*rule-tac* $x{=}r - dist\ x\ a$ **in** *exI*, *rule conjI*, *simp*)
apply *clarify*
apply (*simp add*: *less-diff-eq*)

```
    apply (erule le-less-trans [OF dist-triangle])
    apply (rule conjI)
    apply clarify
    apply (rule-tac x=s - dist x b in exI, rule conjI, simp)
    apply clarify
    apply (simp add: less-diff-eq)
    apply (erule le-less-trans [OF dist-triangle])
    apply (rule conjI)
    apply simp
    apply (clarify, rename-tac c d)
    apply (drule spec, erule mp)
    apply (simp add: dist-Pair-Pair add-strict-mono power-strict-mono)
    apply (rule-tac x=e / sqrt 2 in exI, simp add: divide-pos-pos)
    apply (rule-tac x=e / sqrt 2 in exI, simp add: divide-pos-pos)
    apply (simp add: power-divide)
    done
  qed
qed

end
```

11.4 Continuity of operations

```
lemma tendsto-fst [tendsto-intros]:
  assumes (f ---> a) net
  shows ((λx. fst (f x)) ---> fst a) net
proof (rule topological-tendstoI)
  fix S assume open S fst a ∈ S
  then have open (fst -` S) a ∈ fst -` S
    unfolding open-prod-def
    apply simp-all
    apply clarify
    apply (rule exI, erule conjI)
    apply (rule exI, rule conjI [OF open-UNIV])
    apply auto
    done
  with assms have eventually (λx. f x ∈ fst -` S) net
    by (rule topological-tendstoD)
  then show eventually (λx. fst (f x) ∈ S) net
    by simp
qed

lemma tendsto-snd [tendsto-intros]:
  assumes (f ---> a) net
  shows ((λx. snd (f x)) ---> snd a) net
proof (rule topological-tendstoI)
  fix S assume open S snd a ∈ S
  then have open (snd -` S) a ∈ snd -` S
    unfolding open-prod-def
```

```
    apply simp-all
    apply clarify
    apply (rule exI, rule conjI [OF open-UNIV])
    apply (rule exI, erule conjI)
    apply auto
    done
  with assms have eventually (λx. f x ∈ snd −‘ S) net
    by (rule topological-tendstoD)
  then show eventually (λx. snd (f x) ∈ S) net
    by simp
qed

lemma tendsto-Pair [tendsto-intros]:
  assumes (f ———> a) net and (g ———> b) net
  shows ((λx. (f x, g x)) ———> (a, b)) net
proof (rule topological-tendstoI)
  fix S assume open S (a, b) ∈ S
  then obtain A B where open A open B a ∈ A b ∈ B A × B ⊆ S
    unfolding open-prod-def by auto
  have eventually (λx. f x ∈ A) net
    using ‹(f ———> a) net› ‹open A› ‹a ∈ A›
    by (rule topological-tendstoD)
  moreover
  have eventually (λx. g x ∈ B) net
    using ‹(g ———> b) net› ‹open B› ‹b ∈ B›
    by (rule topological-tendstoD)
  ultimately
  show eventually (λx. (f x, g x) ∈ S) net
    by (rule eventually-elim2)
       (simp add: subsetD [OF ‹A × B ⊆ S›])
qed

lemma Cauchy-fst: Cauchy X ⟹ Cauchy (λn. fst (X n))
unfolding Cauchy-def by (fast elim: le-less-trans [OF dist-fst-le])

lemma Cauchy-snd: Cauchy X ⟹ Cauchy (λn. snd (X n))
unfolding Cauchy-def by (fast elim: le-less-trans [OF dist-snd-le])

lemma Cauchy-Pair:
  assumes Cauchy X and Cauchy Y
  shows Cauchy (λn. (X n, Y n))
proof (rule metric-CauchyI)
  fix r :: real assume 0 < r
  then have 0 < r / sqrt 2 (is 0 < ?s)
    by (simp add: divide-pos-pos)
  obtain M where M: ∀ m≥M. ∀ n≥M. dist (X m) (X n) < ?s
    using metric-CauchyD [OF ‹Cauchy X› ‹0 < ?s›] ..
  obtain N where N: ∀ m≥N. ∀ n≥N. dist (Y m) (Y n) < ?s
    using metric-CauchyD [OF ‹Cauchy Y› ‹0 < ?s›] ..
```

have $\forall\, m{\geq}max\ M\ N.\ \forall\, n{\geq}max\ M\ N.\ dist\ (X\ m,\ Y\ m)\ (X\ n,\ Y\ n) < r$
using *M N* **by** (*simp add*: *real-sqrt-sum-squares-less dist-Pair-Pair*)
then show $\exists\, n0.\ \forall\, m{\geq}n0.\ \forall\, n{\geq}n0.\ dist\ (X\ m,\ Y\ m)\ (X\ n,\ Y\ n) < r$ **..**
qed

lemma *isCont-Pair* [*simp*]:
$[\![isCont\ f\ x;\ isCont\ g\ x]\!] \Longrightarrow isCont\ (\lambda x.\ (f\ x,\ g\ x))\ x$
unfolding *isCont-def* **by** (*rule tendsto-Pair*)

11.5 Product is a complete metric space

instance $*$ :: (*complete-space*, *complete-space*) *complete-space*
proof
fix *X* :: $nat \Rightarrow 'a \times 'b$ **assume** *Cauchy X*
have *1*: $(\lambda n.\ fst\ (X\ n)) \longrightarrow lim\ (\lambda n.\ fst\ (X\ n))$
using *Cauchy-fst* [*OF* ‹*Cauchy X*›]
by (*simp add*: *Cauchy-convergent-iff convergent-LIMSEQ-iff*)
have *2*: $(\lambda n.\ snd\ (X\ n)) \longrightarrow lim\ (\lambda n.\ snd\ (X\ n))$
using *Cauchy-snd* [*OF* ‹*Cauchy X*›]
by (*simp add*: *Cauchy-convergent-iff convergent-LIMSEQ-iff*)
have $X \longrightarrow (lim\ (\lambda n.\ fst\ (X\ n)),\ lim\ (\lambda n.\ snd\ (X\ n)))$
using *tendsto-Pair* [*OF 1 2*] **by** *simp*
then show *convergent X*
by (*rule convergentI*)
qed

11.6 Product is a normed vector space

instantiation
$*$ :: (*real-normed-vector*, *real-normed-vector*) *real-normed-vector*
begin

definition *norm-prod-def*:
$norm\ x = sqrt\ ((norm\ (fst\ x))^2 + (norm\ (snd\ x))^2)$

definition *sgn-prod-def*:
$sgn\ (x{::}'a \times 'b) = scaleR\ (inverse\ (norm\ x))\ x$

lemma *norm-Pair*: $norm\ (a,\ b) = sqrt\ ((norm\ a)^2 + (norm\ b)^2)$
unfolding *norm-prod-def* **by** *simp*

instance proof
fix *r* :: *real* **and** *x y* :: $'a \times 'b$
show $0 \leq norm\ x$
unfolding *norm-prod-def* **by** *simp*
show $norm\ x = 0 \longleftrightarrow x = 0$
unfolding *norm-prod-def*
by (*simp add*: *expand-prod-eq*)
show $norm\ (x + y) \leq norm\ x + norm\ y$
unfolding *norm-prod-def*

```
    apply (rule order-trans [OF - real-sqrt-sum-squares-triangle-ineq])
    apply (simp add: add-mono power-mono norm-triangle-ineq)
    done
  show norm (scaleR r x) = |r| * norm x
    unfolding norm-prod-def
    apply (simp add: power-mult-distrib)
    apply (simp add: right-distrib [symmetric])
    apply (simp add: real-sqrt-mult-distrib)
    done
  show sgn x = scaleR (inverse (norm x)) x
    by (rule sgn-prod-def)
  show dist x y = norm (x - y)
    unfolding dist-prod-def norm-prod-def
    by (simp add: dist-norm)
qed

end

instance * :: (banach, banach) banach ..
```

11.7 Product is an inner product space

```
instantiation * :: (real-inner, real-inner) real-inner
begin

definition inner-prod-def:
  inner x y = inner (fst x) (fst y) + inner (snd x) (snd y)

lemma inner-Pair [simp]: inner (a, b) (c, d) = inner a c + inner b d
  unfolding inner-prod-def by simp

instance proof
  fix r :: real
  fix x y z :: 'a::real-inner * 'b::real-inner
  show inner x y = inner y x
    unfolding inner-prod-def
    by (simp add: inner-commute)
  show inner (x + y) z = inner x z + inner y z
    unfolding inner-prod-def
    by (simp add: inner-add-left)
  show inner (scaleR r x) y = r * inner x y
    unfolding inner-prod-def
    by (simp add: right-distrib)
  show 0 ≤ inner x x
    unfolding inner-prod-def
    by (intro add-nonneg-nonneg inner-ge-zero)
  show inner x x = 0 ⟷ x = 0
    unfolding inner-prod-def expand-prod-eq
    by (simp add: add-nonneg-eq-0-iff)
```

show *norm x = sqrt (inner x x)*
unfolding *norm-prod-def inner-prod-def*
by (*simp add*: *power2-norm-eq-inner*)
qed

end

11.8 Pair operations are linear

interpretation *fst*: *bounded-linear fst*
apply (*unfold-locales*)
apply (*rule fst-add*)
apply (*rule fst-scaleR*)
apply (*rule-tac x=1* **in** *exI*, *simp add*: *norm-Pair*)
done

interpretation *snd*: *bounded-linear snd*
apply (*unfold-locales*)
apply (*rule snd-add*)
apply (*rule snd-scaleR*)
apply (*rule-tac x=1* **in** *exI*, *simp add*: *norm-Pair*)
done

TODO: move to NthRoot

lemma *sqrt-add-le-add-sqrt*:
assumes *x*: $0 \leq x$ **and** *y*: $0 \leq y$
shows $sqrt\ (x + y) \leq sqrt\ x + sqrt\ y$
apply (*rule power2-le-imp-le*)
apply (*simp add*: *real-sum-squared-expand add-nonneg-nonneg x y*)
apply (*simp add*: *mult-nonneg-nonneg x y*)
apply (*simp add*: *add-nonneg-nonneg x y*)
done

lemma *bounded-linear-Pair*:
assumes *f*: *bounded-linear f*
assumes *g*: *bounded-linear g*
shows *bounded-linear* $(\lambda x.\ (f\ x,\ g\ x))$
proof
interpret *f*: *bounded-linear f* **by** *fact*
interpret *g*: *bounded-linear g* **by** *fact*
fix *x y* **and** *r* :: *real*
show $(f\ (x + y),\ g\ (x + y)) = (f\ x,\ g\ x) + (f\ y,\ g\ y)$
by (*simp add*: *f.add g.add*)
show $(f\ (r *_R x),\ g\ (r *_R x)) = r *_R (f\ x,\ g\ x)$
by (*simp add*: *f.scaleR g.scaleR*)
obtain *Kf* **where** $0 < Kf$ **and** *norm-f*: $\bigwedge x.\ norm\ (f\ x) \leq norm\ x * Kf$
using *f.pos-bounded* **by** *fast*
obtain *Kg* **where** $0 < Kg$ **and** *norm-g*: $\bigwedge x.\ norm\ (g\ x) \leq norm\ x * Kg$
using *g.pos-bounded* **by** *fast*

have $\forall x.\ norm\ (f\ x,\ g\ x) \leq norm\ x * (Kf + Kg)$
apply (*rule allI*)
apply (*simp add*: *norm-Pair*)
apply (*rule order-trans* [*OF sqrt-add-le-add-sqrt*], *simp*, *simp*)
apply (*simp add*: *right-distrib*)
apply (*rule add-mono* [*OF norm-f norm-g*])
done
then show $\exists K.\ \forall x.\ norm\ (f\ x,\ g\ x) \leq norm\ x * K$ **..**
qed

11.9 Frechet derivatives involving pairs

lemma *FDERIV-Pair*:
assumes *f*: *FDERIV f x :> f'* **and** *g*: *FDERIV g x :> g'*
shows *FDERIV* $(\lambda x.\ (f\ x,\ g\ x))\ x :> (\lambda h.\ (f'\ h,\ g'\ h))$
apply (*rule FDERIV-I*)
apply (*rule bounded-linear-Pair*)
apply (*rule FDERIV-bounded-linear* [*OF f*])
apply (*rule FDERIV-bounded-linear* [*OF g*])
apply (*simp add*: *norm-Pair*)
apply (*rule real-LIM-sandwich-zero*)
apply (*rule LIM-add-zero*)
apply (*rule FDERIV-D* [*OF f*])
apply (*rule FDERIV-D* [*OF g*])
apply (*rename-tac h*)
apply (*simp add*: *divide-nonneg-pos*)
apply (*rename-tac h*)
apply (*subst add-divide-distrib* [*symmetric*])
apply (*rule divide-right-mono* [*OF - norm-ge-zero*])
apply (*rule order-trans* [*OF sqrt-add-le-add-sqrt*])
apply *simp*
apply *simp*
apply *simp*
done

end

12 Convex: Convexity in real vector spaces

theory *Convex*
imports *Product-Vector*
begin

12.1 Convexity.

definition
convex :: *'a*::*real-vector set* ⇒ *bool* **where**

convex s $\longleftrightarrow$ $(\forall x{\in}s.\ \forall y{\in}s.\ \forall u{\geq}0.\ \forall v{\geq}0.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in s)$

lemma *convex-alt*:
convex s $\longleftrightarrow$ $(\forall x{\in}s.\ \forall y{\in}s.\ \forall u.\ 0 \leq u \wedge u \leq 1 \longrightarrow ((1 - u) *_R x + u *_R y) \in s)$
(**is** - $\longleftrightarrow$ *?alt*)
proof
assume *alt*[*rule-format*]: *?alt*
{ fix *x y* **and** *u v* :: *real* **assume** *mem*: $x \in s$ $y \in s$
assume $0 \leq u$ $0 \leq v$ $u + v = 1$
moreover hence $u = 1 - v$ **by** *auto*
ultimately have $u *_R x + v *_R y \in s$ **using** *alt*[*OF mem*] **by** *auto* **}**
thus *convex s* **unfolding** *convex-def* **by** *auto*
qed (*auto simp*: *convex-def*)

lemma *mem-convex*:
assumes *convex s* $a \in s$ $b \in s$ $0 \leq u$ $u \leq 1$
shows $((1 - u) *_R a + u *_R b) \in s$
using *assms* **unfolding** *convex-alt* **by** *auto*

lemma *convex-empty*[*intro*]: *convex* {}
unfolding *convex-def* **by** *simp*

lemma *convex-singleton*[*intro*]: *convex* {*a*}
unfolding *convex-def* **by** (*auto simp*: *scaleR-left-distrib*[*symmetric*])

lemma *convex-UNIV*[*intro*]: *convex UNIV*
unfolding *convex-def* **by** *auto*

lemma *convex-Inter*: $(\forall s{\in}f.\ convex\ s) ==> convex(\bigcap f)$
unfolding *convex-def* **by** *auto*

lemma *convex-Int*: *convex s* $\Longrightarrow$ *convex t* $\Longrightarrow$ *convex* $(s \cap t)$
unfolding *convex-def* **by** *auto*

lemma *convex-halfspace-le*: *convex* $\{x.\ inner\ a\ x \leq b\}$
unfolding *convex-def*
by (*auto simp*: *inner-add inner-scaleR intro*!: *convex-bound-le*)

lemma *convex-halfspace-ge*: *convex* $\{x.\ inner\ a\ x \geq b\}$
proof −
have $*$:$\{x.\ inner\ a\ x \geq b\} = \{x.\ inner\ (-a)\ x \leq -b\}$ **by** *auto*
show *?thesis* **unfolding** $*$ **using** *convex-halfspace-le*[*of* $-a$ $-b$] **by** *auto*
qed

lemma *convex-hyperplane*: *convex* $\{x.\ inner\ a\ x = b\}$
proof−
have $*$:$\{x.\ inner\ a\ x = b\} = \{x.\ inner\ a\ x \leq b\} \cap \{x.\ inner\ a\ x \geq b\}$ **by** *auto*

show *?thesis* **using** *convex-halfspace-le convex-halfspace-ge*
by (*auto intro*!: *convex-Int simp*: ∗)
qed

lemma *convex-halfspace-lt*: *convex* {*x*. *inner a x* < *b*}
unfolding *convex-def*
by (*auto simp*: *convex-bound-lt inner-add*)

lemma *convex-halfspace-gt*: *convex* {*x*. *inner a x* > *b*}
using *convex-halfspace-lt*[*of* −*a* −*b*] **by** *auto*

lemma *convex-real-interval*:
fixes *a b* :: *real*
shows *convex* {*a*..} **and** *convex* {..*b*}
and *convex* {*a*<..} **and** *convex* {..<*b*}
and *convex* {*a*..*b*} **and** *convex* {*a*<..*b*}
and *convex* {*a*..<*b*} **and** *convex* {*a*<..<*b*}
proof −
have {*a*..} = {*x*. *a* ≤ *inner 1 x*} **by** *auto*
thus *1*: *convex* {*a*..} **by** (*simp only*: *convex-halfspace-ge*)
have {..*b*} = {*x*. *inner 1 x* ≤ *b*} **by** *auto*
thus *2*: *convex* {..*b*} **by** (*simp only*: *convex-halfspace-le*)
have {*a*<..} = {*x*. *a* < *inner 1 x*} **by** *auto*
thus *3*: *convex* {*a*<..} **by** (*simp only*: *convex-halfspace-gt*)
have {..<*b*} = {*x*. *inner 1 x* < *b*} **by** *auto*
thus *4*: *convex* {..<*b*} **by** (*simp only*: *convex-halfspace-lt*)
have {*a*..*b*} = {*a*..} ∩ {..*b*} **by** *auto*
thus *convex* {*a*..*b*} **by** (*simp only*: *convex-Int 1 2*)
have {*a*<..*b*} = {*a*<..} ∩ {..*b*} **by** *auto*
thus *convex* {*a*<..*b*} **by** (*simp only*: *convex-Int 3 2*)
have {*a*..<*b*} = {*a*..} ∩ {..<*b*} **by** *auto*
thus *convex* {*a*..<*b*} **by** (*simp only*: *convex-Int 1 4*)
have {*a*<..<*b*} = {*a*<..} ∩ {..<*b*} **by** *auto*
thus *convex* {*a*<..<*b*} **by** (*simp only*: *convex-Int 3 4*)
qed

12.2 Explicit expressions for convexity in terms of arbitrary sums.

lemma *convex-setsum*:
fixes *C* :: *'a*::*real-vector set*
assumes *finite s* **and** *convex C* **and** (∑ *i* ∈ *s*. *a i*) = *1*
assumes ⋀ *i*. *i* ∈ *s* ⟹ *a i* ≥ *0* **and** ⋀ *i*. *i* ∈ *s* ⟹ *y i* ∈ *C*
shows (∑ *j* ∈ *s*. *a j* ∗*R* *y j*) ∈ *C*
using *assms*
proof (*induct s arbitrary*:*a rule*:*finite-induct*)
case *empty* **thus** *?case* **by** *auto*
next
case (*insert i s*) **note** *asms* = *this*

```
  { assume a i = 1
    hence (∑ j ∈ s. a j) = 0
      using asms by auto
    hence ⋀ j. j ∈ s ⟹ a j = 0
      using setsum-nonneg-0[where 'b=real] asms by fastsimp
    hence ?case using asms by auto }
  moreover
  { assume asm: a i ≠ 1
    from asms have yai: y i ∈ C a i ≥ 0 by auto
    have fis: finite (insert i s) using asms by auto
    hence ai1: a i ≤ 1 using setsum-nonneg-leq-bound[of insert i s a 1] asms by
simp
    hence a i < 1 using asm by auto
    hence i0: 1 − a i > 0 by auto
    let ?a j = a j / (1 − a i)
    { fix j assume j ∈ s
      hence ?a j ≥ 0
        using i0 asms divide-nonneg-pos
        by fastsimp } note a-nonneg = this
    have (∑ j ∈ insert i s. a j) = 1 using asms by auto
    hence (∑ j ∈ s. a j) = 1 − a i using setsum.insert asms by fastsimp
    hence (∑ j ∈ s. a j) / (1 − a i) = 1 using i0 by auto
    hence a1: (∑ j ∈ s. ?a j) = 1 unfolding divide.setsum by simp
    from this asms
    have (∑j∈s. ?a j *R y j) ∈ C using a-nonneg by fastsimp
    hence a i *R y i + (1 − a i) *R (∑ j ∈ s. ?a j *R y j) ∈ C
      using asms[unfolded convex-def, rule-format] yai ai1 by auto
    hence a i *R y i + (∑ j ∈ s. (1 − a i) *R (?a j *R y j)) ∈ C
      using scaleR-right.setsum[of (1 − a i) λ j. ?a j *R y j s] by auto
    hence a i *R y i + (∑ j ∈ s. a j *R y j) ∈ C using i0 by auto
    hence ?case using setsum.insert asms by auto }
  ultimately show ?case by auto
qed

lemma convex:
  shows convex s ⟷ (∀ (k::nat) u x. (∀ i. 1≤i ∧ i≤k ⟶ 0 ≤ u i ∧ x i ∈s) ∧
(setsum u {1..k} = 1)
        ⟶ setsum (λi. u i *R x i) {1..k} ∈ s)
proof safe
  fix k :: nat fix u :: nat ⇒ real fix x
  assume convex s
    ∀ i. 1 ≤ i ∧ i ≤ k ⟶ 0 ≤ u i ∧ x i ∈ s
    setsum u {1..k} = 1
  from this convex-setsum[of {1 .. k} s]
  show (∑j∈{1 .. k}. u j *R x j) ∈ s by auto
next
  assume asm: ∀ k u x. (∀ i :: nat. 1 ≤ i ∧ i ≤ k ⟶ 0 ≤ u i ∧ x i ∈ s) ∧
setsum u {1..k} = 1
    ⟶ (∑ i = 1..k. u i *R (x i :: 'a)) ∈ s
```

{ fix μ :: *real* **fix** x y :: $'a$ **assume** *xy*: $x \in s$ $y \in s$ **assume** *mu*: $\mu \geq 0$ $\mu \leq 1$
let *?u i* = *if* $(i :: nat) = 1$ *then* μ *else* $1 - \mu$
let *?x i* = *if* $(i :: nat) = 1$ *then x else y*
have $\{1 :: nat .. 2\} \cap - \{x.\ x = 1\} = \{2\}$ **by** *auto*
hence *card*: *card* $(\{1 :: nat .. 2\} \cap - \{x.\ x = 1\}) = 1$ **by** *simp*
hence *setsum ?u* $\{1 .. 2\} = 1$
using *setsum-cases*[*of* $\{(1 :: nat) .. 2\}$ $\lambda\ x.\ x = 1$ $\lambda\ x.\ \mu$ $\lambda\ x.\ 1 - \mu$]
by *auto*
from *this asm*[*rule-format*, *of 2 ?u ?x*]
have *s*: $(\sum j \in \{1..2\}.\ ?u\ j *_R ?x\ j) \in s$
using *mu xy* **by** *auto*
have *grarr*: $(\sum j \in \{Suc\ (Suc\ 0)..2\}.\ ?u\ j *_R ?x\ j) = (1 - \mu) *_R y$
using *setsum-head-Suc*[*of Suc* (*Suc 0*) *2* $\lambda\ j.\ (1 - \mu) *_R y$] **by** *auto*
from *setsum-head-Suc*[*of Suc 0 2* $\lambda\ j.\ ?u\ j *_R ?x\ j$, *simplified this*]
have $(\sum j \in \{1..2\}.\ ?u\ j *_R ?x\ j) = \mu *_R x + (1 - \mu) *_R y$ **by** *auto*
hence $(1 - \mu) *_R y + \mu *_R x \in s$ **using** *s* **by** (*auto simp*:*add-commute*) **}**
thus *convex s* **unfolding** *convex-alt* **by** *auto*
qed

lemma *convex-explicit*:
fixes *s* :: $'a$::*real-vector set*
shows *convex s* $\longleftrightarrow$
$(\forall\, t\ u.$ *finite* $t \wedge t \subseteq s \wedge (\forall\, x \in t.\ 0 \leq u\ x) \wedge$ *setsum* $u\ t = 1 \longrightarrow$ *setsum* $(\lambda x.\ u\ x *_R x)\ t \in s)$
proof *safe*
fix *t* **fix** *u* :: $'a \Rightarrow$ *real*
assume *convex s finite t*
$t \subseteq s\ \forall\, x \in t.\ 0 \leq u\ x$ *setsum* $u\ t = 1$
thus $(\sum x \in t.\ u\ x *_R x) \in s$
using *convex-setsum*[*of t s u* $\lambda\ x.\ x$] **by** *auto*
next
assume *asm0*: $\forall\, t.\ \forall\ u.$ *finite* $t \wedge t \subseteq s \wedge (\forall\, x \in t.\ 0 \leq u\ x)$
$\wedge$ *setsum* $u\ t = 1 \longrightarrow (\sum x \in t.\ u\ x *_R x) \in s$
show *convex s*
unfolding *convex-alt*
proof *safe*
fix *x y* **fix** μ :: *real*
assume *asm*: $x \in s$ $y \in s$ $0 \leq \mu$ $\mu \leq 1$
{ assume $x \neq y$
hence $(1 - \mu) *_R x + \mu *_R y \in s$
using *asm0*[*rule-format*, *of* $\{x, y\}$ $\lambda\ z.$ *if* $z = x$ *then* $1 - \mu$ *else* μ]
asm **by** *auto* **}**
moreover
{ assume $x = y$
hence $(1 - \mu) *_R x + \mu *_R y \in s$
using *asm0*[*rule-format*, *of* $\{x, y\}$ $\lambda\ z.\ 1$]
asm **by** (*auto simp*:*field-simps real-vector.scale-left-diff-distrib*) **}**
ultimately show $(1 - \mu) *_R x + \mu *_R y \in s$ **by** *blast*

qed
qed

lemma *convex-finite*: **assumes** *finite s*
shows *convex s* $\longleftrightarrow$ $(\forall u.\ (\forall x{\in}s.\ 0 \le u\ x) \wedge setsum\ u\ s = 1 \longrightarrow setsum\ (\lambda x.\ u\ x *_R x)\ s \in s)$
unfolding *convex-explicit*
proof (*safe elim*!: *conjE*)
fix *t u* **assume** *sum*: $\forall u.\ (\forall x{\in}s.\ 0 \le u\ x) \wedge setsum\ u\ s = 1 \longrightarrow (\sum x{\in}s.\ u\ x *_R x) \in s$
and *as*: *finite t* $t \subseteq s$ $\forall x{\in}t.\ 0 \le u\ x$ *setsum u t* = (*1*::*real*)
have $*$:$s \cap t = t$ **using** *as*(*2*) **by** *auto*
have *if-distrib-arg*: $\bigwedge P\ f\ g\ x.\ (if\ P\ then\ f\ else\ g)\ x = (if\ P\ then\ f\ x\ else\ g\ x)$ **by** *simp*
show $(\sum x{\in}t.\ u\ x *_R x) \in s$
using *sum*[*THEN spec*[**where** $x{=}\lambda x.\ if\ x{\in}t\ then\ u\ x\ else\ 0$]] *as* $*$
by (*auto simp*: *assms setsum-cases if-distrib if-distrib-arg*)
qed (*erule-tac x*=*s* **in** *allE*, *erule-tac x*=*u* **in** *allE*, *auto*)

definition
convex-on :: '*a*::*real-vector set* $\Rightarrow$ ('*a* $\Rightarrow$ *real*) $\Rightarrow$ *bool* **where**
convex-on s f $\longleftrightarrow$
$(\forall x{\in}s.\ \forall y{\in}s.\ \forall u{\ge}0.\ \forall v{\ge}0.\ u + v = 1 \longrightarrow f\ (u *_R x + v *_R y) \le u * f\ x + v * f\ y)$

lemma *convex-on-subset*: *convex-on t f* $\Longrightarrow$ $s \subseteq t$ $\Longrightarrow$ *convex-on s f*
unfolding *convex-on-def* **by** *auto*

lemma *convex-add*[*intro*]:
assumes *convex-on s f convex-on s g*
shows *convex-on s* $(\lambda x.\ f\ x + g\ x)$
proof−
{ fix *x y* **assume** $x{\in}s$ $y{\in}s$ **moreover**
fix *u v* ::*real* **assume** $0 \le u$ $0 \le v$ $u + v = 1$
ultimately have $f\ (u *_R x + v *_R y) + g\ (u *_R x + v *_R y) \le (u * f\ x + v * f\ y) + (u * g\ x + v * g\ y)$
using *assms* **unfolding** *convex-on-def* **by** (*auto simp add*:*add-mono*)
hence $f\ (u *_R x + v *_R y) + g\ (u *_R x + v *_R y) \le u * (f\ x + g\ x) + v * (f\ y + g\ y)$ **by** (*simp add*: *field-simps*) **}**
thus *?thesis* **unfolding** *convex-on-def* **by** *auto*
qed

lemma *convex-cmul*[*intro*]:
assumes $0 \le$ (*c*::*real*) *convex-on s f*
shows *convex-on s* $(\lambda x.\ c * f\ x)$
proof−
have $*$:$\bigwedge u\ c\ fx\ v\ fy$ ::*real*. $u * (c * fx) + v * (c * fy) = c * (u * fx + v * fy)$
by (*simp add*: *field-simps*)
show *?thesis* **using** *assms*(*2*) **and** *mult-mono1*[*OF - assms*(*1*)] **unfolding** *convex-on-def*

and $*$ **by** *auto*
qed

lemma *convex-lower*:
assumes *convex-on s f* $x \in s$ $y \in s$ $0 \le u$ $0 \le v$ $u + v = 1$
shows $f\ (u *_R x + v *_R y) \le max\ (f\ x)\ (f\ y)$
proof−
let *?m* $= max\ (f\ x)\ (f\ y)$
have $u * f\ x + v * f\ y \le u * max\ (f\ x)\ (f\ y) + v * max\ (f\ x)\ (f\ y)$
using *assms*(*4*,*5*) **by**(*auto simp add*: *mult-mono1 add-mono*)
also have $\ldots = max\ (f\ x)\ (f\ y)$ **using** *assms*(*6*) **unfolding** *distrib*[*THEN sym*]
by *auto*
finally show *?thesis*
using *assms* **unfolding** *convex-on-def* **by** *fastsimp*
qed

lemma *convex-distance*[*intro*]:
fixes *s* :: *'a*::*real-normed-vector set*
shows *convex-on s* $(\lambda x.\ dist\ a\ x)$
proof(*auto simp add*: *convex-on-def dist-norm*)
fix *x y* **assume** $x \in s$ $y \in s$
fix *u v* ::*real* **assume** $0 \le u$ $0 \le v$ $u + v = 1$
have $a = u *_R a + v *_R a$ **unfolding** *scaleR-left-distrib*[*THEN sym*] **and**
‹*u*+*v*=*1*› **by** *simp*
hence $*$:$a - (u *_R x + v *_R y) = (u *_R (a - x)) + (v *_R (a - y))$
by (*auto simp add*: *algebra-simps*)
show $norm\ (a - (u *_R x + v *_R y)) \le u * norm\ (a - x) + v * norm\ (a - y)$
unfolding $*$ **using** *norm-triangle-ineq*[*of u* $*_R$ $(a - x)$ *v* $*_R$ $(a - y)$]
using ‹*0* $\le$ *u*› ‹*0* $\le$ *v*› **by** *auto*
qed

12.3 Arithmetic operations on sets preserve convexity.

lemma *convex-scaling*:
assumes *convex s*
shows*convex* $((\lambda x.\ c *_R x)$ ‘ *s*)
using *assms* **unfolding** *convex-def image-iff*
proof *safe*
fix *x xa y xb* :: *'a*::*real-vector* **fix** *u v* :: *real*
assume *asm*: $\forall x \in s.\ \forall y \in s.\ \forall u \ge 0.\ \forall v \ge 0.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in s$
xa $\in$ *s xb* $\in$ *s* $0 \le u$ $0 \le v$ $u + v = 1$
show $\exists x \in s.\ u *_R c *_R xa + v *_R c *_R xb = c *_R x$
using *bexI*[*of* - $u *_R xa + v *_R xb$] *asm* **by** (*auto simp add*: *algebra-simps*)
qed

lemma *convex-negations*: *convex s* $\Longrightarrow$ *convex* $((\lambda x.\ -x)$‘ *s*)
using *assms* **unfolding** *convex-def image-iff*
proof *safe*
fix *x xa y xb* :: *'a*::*real-vector* **fix** *u v* :: *real*

assume *asm*: $\forall x{\in}s.\ \forall y{\in}s.\ \forall u{\geq}0.\ \forall v{\geq}0.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in s$
xa $\in s$ *xb* $\in s$ $0 \leq u$ $0 \leq v$ $u + v = 1$
show $\exists x{\in}s.\ u *_R - xa + v *_R - xb = - x$
using *bexI*[*of* - $u *_R xa + v *_R xb$] *asm* **by** *auto*
qed

lemma *convex-sums*:
assumes *convex s convex t*
shows *convex* $\{x + y |\ x\ y.\ x \in s \wedge y \in t\}$
using *assms* **unfolding** *convex-def image-iff*
proof *safe*
fix *xa xb ya yb* **assume** *xy*:*xa*$\in s$ *xb*$\in s$ *ya*$\in t$ *yb*$\in t$
fix *u v* ::*real* **assume** *uv*:$0 \leq u$ $0 \leq v$ $u + v = 1$
show $\exists x\ y.\ u *_R (xa + ya) + v *_R (xb + yb) = x + y \wedge x \in s \wedge y \in t$
using *exI*[*of* - $u *_R xa + v *_R xb$] *exI*[*of* - $u *_R ya + v *_R yb$]
assms[*unfolded convex-def*] *uv xy* **by** (*auto simp add*:*scaleR-right-distrib*)
qed

lemma *convex-differences*:
assumes *convex s convex t*
shows *convex* $\{x - y |\ x\ y.\ x \in s \wedge y \in t\}$
proof −
have $\{x - y |\ x\ y.\ x \in s \wedge y \in t\} = \{x + y\ | x\ y.\ x \in s \wedge y \in uminus\ `\ t\}$
proof *safe*
fix *x x' y* **assume** $x' \in s$ $y \in t$
thus $\exists x\ y'.\ x' - y = x + y' \wedge x \in s \wedge y' \in uminus\ `\ t$
using *exI*[*of* - x'] *exI*[*of* - $-y$] **by** *auto*
next
fix *x x' y y'* **assume** $x' \in s$ $y' \in t$
thus $\exists x\ y.\ x' + - y' = x - y \wedge x \in s \wedge y \in t$
using *exI*[*of* - x'] *exI*[*of* - y'] **by** *auto*
qed
thus *?thesis* **using** *convex-sums*[*OF assms*(*1*) *convex-negations*[*OF assms*(*2*)]]
by *auto*
qed

lemma *convex-translation*: **assumes** *convex s* **shows** *convex* $((\lambda x.\ a + x)\ `\ s)$
proof− **have** $\{a + y |\ y.\ y \in s\} = (\lambda x.\ a + x)\ `\ s$ **by** *auto*
thus *?thesis* **using** *convex-sums*[*OF convex-singleton*[*of a*] *assms*] **by** *auto* **qed**

lemma *convex-affinity*: **assumes** *convex s* **shows** *convex* $((\lambda x.\ a + c *_R x)\ `\ s)$
proof− **have** $(\lambda x.\ a + c *_R x)\ `\ s = op + a\ `\ op *_R c\ `\ s$ **by** *auto*
thus *?thesis* **using** *convex-translation*[*OF convex-scaling*[*OF assms*], *of a c*] **by** *auto* **qed**

lemma *convex-linear-image*:
assumes *c*:*convex s* **and** *l*:*bounded-linear f*
shows *convex*(*f* ` *s*)
proof(*auto simp add*: *convex-def*)

```
  interpret f: bounded-linear f by fact
  fix x y assume xy:x ∈ s y ∈ s
  fix u v ::real assume uv:0 ≤ u 0 ≤ v u + v = 1
  show u *R f x + v *R f y ∈ f ' s unfolding image-iff
    using bexI[of - u *R x + v *R y] f.add f.scaleR
      c[unfolded convex-def] xy uv by auto
qed


lemma pos-is-convex:
  shows convex {0 :: real <..}
unfolding convex-alt
proof safe
  fix y x μ :: real
  assume asms: y > 0 x > 0 μ ≥ 0 μ ≤ 1
  { assume μ = 0
    hence μ *R x + (1 − μ) *R y = y by simp
    hence μ *R x + (1 − μ) *R y > 0 using asms by simp }
  moreover
  { assume μ = 1
    hence μ *R x + (1 − μ) *R y > 0 using asms by simp }
  moreover
  { assume μ ≠ 1 μ ≠ 0
    hence μ > 0 (1 − μ) > 0 using asms by auto
    hence μ *R x + (1 − μ) *R y > 0 using asms
      by (auto simp add: add-pos-pos mult-pos-pos) }
  ultimately show (1 − μ) *R y + μ *R x > 0 using assms by fastsimp
qed

lemma convex-on-setsum:
  fixes a :: 'a ⇒ real
  fixes y :: 'a ⇒ 'b::real-vector
  fixes f :: 'b ⇒ real
  assumes finite s s ≠ {}
  assumes convex-on C f
  assumes convex C
  assumes (∑ i ∈ s. a i) = 1
  assumes ⋀ i. i ∈ s ⟹ a i ≥ 0
  assumes ⋀ i. i ∈ s ⟹ y i ∈ C
  shows f (∑ i ∈ s. a i *R y i) ≤ (∑ i ∈ s. a i * f (y i))
using assms
proof (induct s arbitrary:a rule:finite-ne-induct)
  case (singleton i)
  hence ai: a i = 1 by auto
  thus ?case by auto
next
  case (insert i s) note asms = this
  hence convex-on C f by simp
  from this[unfolded convex-on-def, rule-format]
```

```
  have conv: ⋀ x y μ. ⟦x ∈ C; y ∈ C; 0 ≤ μ; μ ≤ 1⟧
  ⟹ f (μ *R x + (1 − μ) *R y) ≤ μ * f x + (1 − μ) * f y
    by simp
  { assume a i = 1
    hence (∑ j ∈ s. a j) = 0
      using asms by auto
    hence ⋀ j. j ∈ s ⟹ a j = 0
      using setsum-nonneg-0[where 'b=real] asms by fastsimp
    hence ?case using asms by auto }
  moreover
  { assume asm: a i ≠ 1
    from asms have yai: y i ∈ C a i ≥ 0 by auto
    have fis: finite (insert i s) using asms by auto
     hence ai1: a i ≤ 1 using setsum-nonneg-leq-bound[of insert i s a] asms by
simp
    hence a i < 1 using asm by auto
    hence i0: 1 − a i > 0 by auto
    let ?a j = a j / (1 − a i)
    { fix j assume j ∈ s
      hence ?a j ≥ 0
        using i0 asms divide-nonneg-pos
        by fastsimp } note a-nonneg = this
    have (∑ j ∈ insert i s. a j) = 1 using asms by auto
    hence (∑ j ∈ s. a j) = 1 − a i using setsum.insert asms by fastsimp
    hence (∑ j ∈ s. a j) / (1 − a i) = 1 using i0 by auto
    hence a1: (∑ j ∈ s. ?a j) = 1 unfolding divide.setsum by simp
    have convex C using asms by auto
    hence asum: (∑ j ∈ s. ?a j *R y j) ∈ C
      using asms convex-setsum[OF ‹finite s›
        ‹convex C› a1 a-nonneg] by auto
    have asum-le: f (∑ j ∈ s. ?a j *R y j) ≤ (∑ j ∈ s. ?a j * f (y j))
      using a-nonneg a1 asms by blast
   have f (∑ j ∈ insert i s. a j *R y j) = f ((∑ j ∈ s. a j *R y j) + a i *R y i)
      using setsum.insert[of s i λ j. a j *R y j, OF ‹finite s› ‹i ∉ s›] asms
      by (auto simp only:add-commute)
    also have ... = f (((1 − a i) * inverse (1 − a i)) *R (∑ j ∈ s. a j *R y j)
+ a i *R y i)
      using i0 by auto
    also have ... = f ((1 − a i) *R (∑ j ∈ s. (a j * inverse (1 − a i)) *R y j)
+ a i *R y i)
      using scaleR-right.setsum[of inverse (1 − a i) λ j. a j *R y j s, symmetric]
by (auto simp:algebra-simps)
    also have ... = f ((1 − a i) *R (∑ j ∈ s. ?a j *R y j) + a i *R y i)
      by (auto simp: divide-inverse)
    also have ... ≤ (1 − a i) *R f ((∑ j ∈ s. ?a j *R y j)) + a i * f (y i)
      using conv[of y i (∑ j ∈ s. ?a j *R y j) a i, OF yai(1) asum yai(2) ai1]
      by (auto simp add:add-commute)
    also have ... ≤ (1 − a i) * (∑ j ∈ s. ?a j * f (y j)) + a i * f (y i)
      using add-right-mono[OF mult-left-mono[of - - 1 − a i,
```

```
      OF asum-le less-imp-le[OF i0]], of a i * f (y i)] by simp
   also have ... = (∑ j ∈ s. (1 - a i) * ?a j * f (y j)) + a i * f (y i)
    unfolding mult-right.setsum[of 1 - a i λ j. ?a j * f (y j)] using i0 by auto
   also have ... = (∑ j ∈ s. a j * f (y j)) + a i * f (y i) using i0 by auto
   also have ... = (∑ j ∈ insert i s. a j * f (y j)) using asms by auto
   finally have f (∑ j ∈ insert i s. a j *R y j) ≤ (∑ j ∈ insert i s. a j * f (y
j))
    by simp }
  ultimately show ?case by auto
qed

lemma convex-on-alt:
  fixes C :: 'a::real-vector set
  assumes convex C
  shows convex-on C f =
  (∀ x ∈ C. ∀ y ∈ C. ∀ μ :: real. μ ≥ 0 ∧ μ ≤ 1
     ⟶ f (μ *R x + (1 - μ) *R y) ≤ μ * f x + (1 - μ) * f y)
proof safe
  fix x y fix μ :: real
  assume asms: convex-on C f x ∈ C y ∈ C 0 ≤ μ μ ≤ 1
  from this[unfolded convex-on-def, rule-format]
  have ⋀ u v. ⟦0 ≤ u; 0 ≤ v; u + v = 1⟧ ⟹ f (u *R x + v *R y) ≤ u * f x +
v * f y by auto
  from this[of μ 1 - μ, simplified] asms
  show f (μ *R x + (1 - μ) *R y)
        ≤ μ * f x + (1 - μ) * f y by auto
next
  assume asm: ∀ x∈C. ∀ y∈C. ∀ μ. 0 ≤ μ ∧ μ ≤ 1 ⟶ f (μ *R x + (1 - μ) *R
y) ≤ μ * f x + (1 - μ) * f y
  {fix x y fix u v :: real
   assume lasm: x ∈ C y ∈ C u ≥ 0 v ≥ 0 u + v = 1
   hence[simp]: 1 - u = v by auto
   from asm[rule-format, of x y u]
   have f (u *R x + v *R y) ≤ u * f x + v * f y using lasm by auto }
  thus convex-on C f unfolding convex-on-def by auto
qed


lemma pos-convex-function:
  fixes f :: real ⇒ real
  assumes convex C
  assumes leq: ⋀ x y. ⟦x ∈ C ; y ∈ C⟧ ⟹ f' x * (y - x) ≤ f y - f x
  shows convex-on C f
unfolding convex-on-alt[OF assms(1)]
using assms
proof safe
  fix x y μ :: real
  let ?x = μ *R x + (1 - μ) *R y
  assume asm: convex C x ∈ C y ∈ C μ ≥ 0 μ ≤ 1
```

```
  hence 1 − μ ≥ 0 by auto
  hence xpos: ?x ∈ C using asm unfolding convex-alt by fastsimp
  have geq: μ * (f x − f ?x) + (1 − μ) * (f y − f ?x)
          ≥ μ * f' ?x * (x − ?x) + (1 − μ) * f' ?x * (y − ?x)
    using add-mono[OF mult-mono1[OF leq[OF xpos asm(2)] ‹μ ≥ 0›]
      mult-mono1[OF leq[OF xpos asm(3)] ‹1 − μ ≥ 0›]] by auto
  hence μ * f x + (1 − μ) * f y − f ?x ≥ 0
    by (auto simp add:field-simps)
  thus f (μ *R x + (1 − μ) *R y) ≤ μ * f x + (1 − μ) * f y
    using convex-on-alt by auto
qed

lemma atMostAtLeast-subset-convex:
  fixes C :: real set
  assumes convex C
  assumes x ∈ C y ∈ C x < y
  shows {x .. y} ⊆ C
proof safe
  fix z assume zasm: z ∈ {x .. y}
  { assume asm: x < z z < y
    let ?μ = (y − z) / (y − x)
    have 0 ≤ ?μ ?μ ≤ 1 using assms asm by (auto simp add:field-simps)
    hence comb: ?μ * x + (1 − ?μ) * y ∈ C
        using assms iffD1[OF convex-alt, rule-format, of C y x ?μ] by (simp
add:algebra-simps)
    have ?μ * x + (1 − ?μ) * y = (y − z) * x / (y − x) + (1 − (y − z) / (y −
x)) * y
      by (auto simp add:field-simps)
    also have ... = ((y − z) * x + (y − x − (y − z)) * y) / (y − x)
      using assms unfolding add-divide-distrib by (auto simp:field-simps)
    also have ... = z
      using assms by (auto simp:field-simps)
    finally have z ∈ C
      using comb by auto } note less = this
  show z ∈ C using zasm less assms
    unfolding atLeastAtMost-iff le-less by auto
qed

lemma f''-imp-f':
  fixes f :: real ⇒ real
  assumes convex C
  assumes f': ⋀ x. x ∈ C ⟹ DERIV f x :> (f' x)
  assumes f'': ⋀ x. x ∈ C ⟹ DERIV f' x :> (f'' x)
  assumes pos: ⋀ x. x ∈ C ⟹ f'' x ≥ 0
  assumes x ∈ C y ∈ C
  shows f' x * (y − x) ≤ f y − f x
using assms
proof −
  { fix x y :: real assume asm: x ∈ C y ∈ C y > x
```

hence *ge*: $y - x > 0$ $y - x \geq 0$ **by** *auto*
from *asm* **have** *le*: $x - y < 0$ $x - y \leq 0$ **by** *auto*
then obtain *z1* **where** *z1*: $z1 > x$ $z1 < y$ $f\, y - f\, x = (y - x) * f'\, z1$
using *subsetD*[*OF atMostAtLeast-subset-convex*[*OF* ‹*convex C*› ‹$x \in C$› ‹$y \in C$› ‹$x < y$›],
THEN f', *THEN MVT2*[*OF* ‹$x < y$›, *rule-format*, *unfolded atLeastAtMost-iff*[*symmetric*]]]
by *auto*
hence *z1* $\in$ *C* **using** *atMostAtLeast-subset-convex*
‹*convex C*› ‹$x \in C$› ‹$y \in C$› ‹$x < y$› **by** *fastsimp*
from *z1* **have** *z1'*: $f\, x - f\, y = (x - y) * f'\, z1$
by (*simp add*:*field-simps*)
obtain *z2* **where** *z2*: $z2 > x$ $z2 < z1$ $f'\, z1 - f'\, x = (z1 - x) * f''\, z2$
using *subsetD*[*OF atMostAtLeast-subset-convex*[*OF* ‹*convex C*› ‹$x \in C$› ‹*z1*
$\in C$› ‹$x < z1$›],
THEN f'', *THEN MVT2*[*OF* ‹$x < z1$›, *rule-format*, *unfolded atLeastAtMost-iff*[*symmetric*]]]
z1
by *auto*
obtain *z3* **where** *z3*: $z3 > z1$ $z3 < y$ $f'\, y - f'\, z1 = (y - z1) * f''\, z3$
using *subsetD*[*OF atMostAtLeast-subset-convex*[*OF* ‹*convex C*› ‹*z1* $\in C$› ‹*y*
$\in C$› ‹$z1 < y$›],
THEN f'', *THEN MVT2*[*OF* ‹$z1 < y$›, *rule-format*, *unfolded atLeastAtMost-iff*[*symmetric*]]]
z1
by *auto*
have $f'\, y - (f\, x - f\, y) \,/\, (x - y) = f'\, y - f'\, z1$
using *asm z1'* **by** *auto*
also have $\ldots = (y - z1) * f''\, z3$ **using** *z3* **by** *auto*
finally have *cool'*: $f'\, y - (f\, x - f\, y) \,/\, (x - y) = (y - z1) * f''\, z3$ **by** *simp*
have *A'*: $y - z1 \geq 0$ **using** *z1* **by** *auto*
have *z3* $\in$ *C* **using** *z3 asm atMostAtLeast-subset-convex*
‹*convex C*› ‹$x \in C$› ‹*z1* $\in C$› ‹$x < z1$› **by** *fastsimp*
hence *B'*: $f''\, z3 \geq 0$ **using** *assms* **by** *auto*
from *A' B'* **have** $(y - z1) * f''\, z3 \geq 0$ **using** *mult-nonneg-nonneg* **by** *auto*
from *cool' this* **have** $f'\, y - (f\, x - f\, y) \,/\, (x - y) \geq 0$ **by** *auto*
from *mult-right-mono-neg*[*OF this le*(*2*)]
have $f'\, y * (x - y) - (f\, x - f\, y) \,/\, (x - y) * (x - y) \leq 0 * (x - y)$
by (*simp add*: *algebra-simps*)
hence $f'\, y * (x - y) - (f\, x - f\, y) \leq 0$ **using** *le* **by** *auto*
hence *res*: $f'\, y * (x - y) \leq f\, x - f\, y$ **by** *auto*
have $(f\, y - f\, x) \,/\, (y - x) - f'\, x = f'\, z1 - f'\, x$
using *asm z1* **by** *auto*
also have $\ldots = (z1 - x) * f''\, z2$ **using** *z2* **by** *auto*
finally have *cool*: $(f\, y - f\, x) \,/\, (y - x) - f'\, x = (z1 - x) * f''\, z2$ **by** *simp*
have *A*: $z1 - x \geq 0$ **using** *z1* **by** *auto*
have *z2* $\in$ *C* **using** *z2 z1 asm atMostAtLeast-subset-convex*
‹*convex C*› ‹*z1* $\in C$› ‹*y* $\in C$› ‹$z1 < y$› **by** *fastsimp*
hence *B*: $f''\, z2 \geq 0$ **using** *assms* **by** *auto*
from *A B* **have** $(z1 - x) * f''\, z2 \geq 0$ **using** *mult-nonneg-nonneg* **by** *auto*
from *cool this* **have** $(f\, y - f\, x) \,/\, (y - x) - f'\, x \geq 0$ **by** *auto*
from *mult-right-mono*[*OF this ge*(*2*)]

```
    have (f y − f x) / (y − x) ∗ (y − x) − f′ x ∗ (y − x) ≥ 0 ∗ (y − x)
      by (simp add: algebra-simps)
    hence f y − f x − f′ x ∗ (y − x) ≥ 0 using ge by auto
    hence f y − f x ≥ f′ x ∗ (y − x) f′ y ∗ (x − y) ≤ f x − f y
      using res by auto } note less-imp = this
  { fix x y :: real assume x ∈ C y ∈ C x ≠ y
    hencef y − f x ≥ f′ x ∗ (y − x)
    unfolding neq-iff using less-imp by auto } note neq-imp = this
  moreover
  { fix x y :: real assume asm: x ∈ C y ∈ C x = y
    hence f y − f x ≥ f′ x ∗ (y − x) by auto }
  ultimately show ?thesis using assms by blast
qed

lemma f′′-ge0-imp-convex:
  fixes f :: real ⇒ real
  assumes conv: convex C
  assumes f′: ⋀ x. x ∈ C ⟹ DERIV f x :> (f′ x)
  assumes f′′: ⋀ x. x ∈ C ⟹ DERIV f′ x :> (f′′ x)
  assumes pos: ⋀ x. x ∈ C ⟹ f′′ x ≥ 0
  shows convex-on C f
using f′′-imp-f′[OF conv f′ f′′ pos] assms pos-convex-function by fastsimp

lemma minus-log-convex:
  fixes b :: real
  assumes b > 1
  shows convex-on {0 <..} (λ x. − log b x)
proof −
  have ⋀ z. z > 0 ⟹ DERIV (log b) z :> 1 / (ln b ∗ z) using DERIV-log by
auto
  hence f′: ⋀ z. z > 0 ⟹ DERIV (λ z. − log b z) z :> − 1 / (ln b ∗ z)
    using DERIV-minus by auto
  have ⋀ z :: real. z > 0 ⟹ DERIV inverse z :> − (inverse z ^ Suc (Suc 0))
    using less-imp-neq[THEN not-sym, THEN DERIV-inverse] by auto
  from this[THEN DERIV-cmult, of - − 1 / ln b]
  have ⋀ z :: real. z > 0 ⟹ DERIV (λ z. (− 1 / ln b) ∗ inverse z) z :> (− 1
/ ln b) ∗ (− (inverse z ^ Suc (Suc 0)))
    by auto
  hence f′′0: ⋀ z :: real. z > 0 ⟹ DERIV (λ z. − 1 / (ln b ∗ z)) z :> 1 / (ln
b ∗ z ∗ z)
    unfolding inverse-eq-divide by (auto simp add: mult-assoc)
  have f′′-ge0: ⋀ z :: real. z > 0 ⟹ 1 / (ln b ∗ z ∗ z) ≥ 0
   using ‹b > 1› by (auto intro!:less-imp-le simp add:divide-pos-pos[of 1] mult-pos-pos)
  from f′′-ge0-imp-convex[OF pos-is-convex,
    unfolded greaterThan-iff, OF f′ f′′0 f′′-ge0]
  show ?thesis by auto
qed

end
```

13 Euclidean-Space: (Real) Vectors in Euclidean space, and elementary linear algebra.

theory *Euclidean-Space*
imports
Complex-Main ~~/*src*/*HOL*/*Decision-Procs*/*Dense-Linear-Order*
Finite-Cartesian-Product Infinite-Set Numeral-Type
Inner-Product L2-Norm Convex
uses *positivstellensatz.ML* (*normarith.ML*)
begin

13.1 Basic componentwise operations on vectors.

instantiation *cart* :: (*times*,*finite*) *times*
begin
definition *vector-mult-def* : *op* ∗ ≡ (λ *x y*. (χ *i*. (*x*$*i*) ∗ (*y*$*i*)))
instance ..
end

instantiation *cart* :: (*one*,*finite*) *one*
begin
definition *vector-one-def* : *1* ≡ (χ *i*. *1*)
instance ..
end

instantiation *cart* :: (*ord*,*finite*) *ord*
begin
definition *vector-le-def*:
less-eq (*x* :: *'a* ^*'b*) *y* = (*ALL i*. *x*$*i* <= *y*$*i*)
definition *vector-less-def*: *less* (*x* :: *'a* ^*'b*) *y* = (*ALL i*. *x*$*i* < *y*$*i*)
instance by (*intro-classes*)
end

The ordering on one-dimensional vectors is linear.

class *cart-one* = **assumes** *UNIV-one*: *card* (*UNIV* :: *'a set*) = *Suc 0*
begin
subclass *finite*
proof from *UNIV-one* **show** *finite* (*UNIV* :: *'a set*)
by (*auto intro*!: *card-ge-0-finite*) **qed**
end

instantiation *cart* :: (*linorder*,*cart-one*) *linorder* **begin**
instance proof
guess *a B* **using** *UNIV-one*[**where** *'a*=*'b*] **unfolding** *card-Suc-eq* **apply**− **by**(*erule exE*)+
hence ∗:*UNIV* = {*a*} **by** *auto*

have $\bigwedge P.\ (\forall\, i{\in}UNIV.\ P\ i) \longleftrightarrow P\ a$ **unfolding** $*$ **by** *auto* **hence** *all*:$\bigwedge P.\ (\forall\, i.\ P\ i) \longleftrightarrow P\ a$ **by** *auto*
fix *x y z*::'a^'b::*cart-one* **note** $*$ = *vector-le-def vector-less-def all Cart-eq*
show $x{\leq}x\ (x < y) = (x \leq y \wedge \neg\ y \leq x)\ x{\leq}y \vee y{\leq}x$ **unfolding** $*$ **by**(*auto simp only*:*field-simps*)
{ assume $x{\leq}y\ y{\leq}z$ **thus** $x{\leq}z$ **unfolding** $*$ **by**(*auto simp only*:*field-simps*) **}**
{ assume $x{\leq}y\ y{\leq}x$ **thus** $x{=}y$ **unfolding** $*$ **by**(*auto simp only*:*field-simps*) **}**
qed end

Also the scalar-vector multiplication.

definition *vector-scalar-mult*:: 'a::*times* $\Rightarrow$ 'a ^ 'n $\Rightarrow$ 'a ^ 'n (**infixl** $*s$ *70*)
where $c *s\ x = (\chi\ i.\ c * (x\$i))$

Constant Vectors

definition *vec* $x = (\chi\ i.\ x)$

13.2 A naive proof procedure to lift really trivial arithmetic stuff from the basis of the vector space.

method-setup *vector* = ⟪
let
val ss1 = HOL-basic-ss addsimps [@{*thm setsum-addf*} *RS sym*,
@{*thm setsum-subtractf*} *RS sym*, @{*thm setsum-right-distrib*},
@{*thm setsum-left-distrib*}, @{*thm setsum-negf*} *RS sym*]
val ss2 = @{simpset} addsimps
[@{*thm vector-add-def*}, @{*thm vector-mult-def*},
@{*thm vector-minus-def*}, @{*thm vector-uminus-def*},
@{*thm vector-one-def*}, @{*thm vector-zero-def*}, @{*thm vec-def*},
@{*thm vector-scaleR-def*},
@{*thm Cart-lambda-beta*}, @{*thm vector-scalar-mult-def*}]
fun vector-arith-tac ths =
simp-tac ss1
THEN' (fn i => rtac @{thm setsum-cong2} i
ORELSE rtac @{thm setsum-0'} i
ORELSE simp-tac (HOL-basic-ss addsimps [@{thm Cart-eq}]) i)
(THEN' TRY o clarify-tac HOL-cs THEN' (TRY o rtac @{thm iffI}) *)*
THEN' asm-full-simp-tac (ss2 addsimps ths)
in
Attrib.thms >> (fn ths => K (SIMPLE-METHOD' (vector-arith-tac ths)))
end
⟫ *Lifts trivial vector statements to real arith statements*

lemma *vec-0*[*simp*]: *vec 0 = 0* **by** (*vector vector-zero-def*)
lemma *vec-1*[*simp*]: *vec 1 = 1* **by** (*vector vector-one-def*)

Obvious "component-pushing".

lemma *vec-component* [*simp*]: *vec x $ i = x*
by (*vector vec-def*)

lemma *vector-mult-component* [*simp*]: $(x * y)\$i = x\$i * y\$i$
by *vector*

lemma *vector-smult-component* [*simp*]: $(c *s y)\$i = c * (y\$i)$
by *vector*

lemma *cond-component*: $(if\ b\ then\ x\ else\ y)\$i = (if\ b\ then\ x\$i\ else\ y\$i)$ **by** *vector*

lemmas *vector-component* =
vec-component vector-add-component vector-mult-component
vector-smult-component vector-minus-component vector-uminus-component
vector-scaleR-component cond-component

13.3 Some frequently useful arithmetic lemmas over vectors.

instance *cart* :: (*semigroup-mult*,*finite*) *semigroup-mult*
apply (*intro-classes*) **by** (*vector mult-assoc*)

instance *cart* :: (*monoid-mult*,*finite*) *monoid-mult*
apply (*intro-classes*) **by** *vector+*

instance *cart* :: (*ab-semigroup-mult*,*finite*) *ab-semigroup-mult*
apply (*intro-classes*) **by** (*vector mult-commute*)

instance *cart* :: (*ab-semigroup-idem-mult*,*finite*) *ab-semigroup-idem-mult*
apply (*intro-classes*) **by** (*vector mult-idem*)

instance *cart* :: (*comm-monoid-mult*,*finite*) *comm-monoid-mult*
apply (*intro-classes*) **by** *vector*

instance *cart* :: (*semiring*,*finite*) *semiring*
apply (*intro-classes*) **by** (*vector field-simps*)+

instance *cart* :: (*semiring-0*,*finite*) *semiring-0*
apply (*intro-classes*) **by** (*vector field-simps*)+
instance *cart* :: (*semiring-1*,*finite*) *semiring-1*
apply (*intro-classes*) **by** *vector*
instance *cart* :: (*comm-semiring*,*finite*) *comm-semiring*
apply (*intro-classes*) **by** (*vector field-simps*)+

instance *cart* :: (*comm-semiring-0*,*finite*) *comm-semiring-0* **by** (*intro-classes*)
instance *cart* :: (*cancel-comm-monoid-add*, *finite*) *cancel-comm-monoid-add* **..**
instance *cart* :: (*semiring-0-cancel*,*finite*) *semiring-0-cancel* **by** (*intro-classes*)
instance *cart* :: (*comm-semiring-0-cancel*,*finite*) *comm-semiring-0-cancel* **by** (*intro-classes*)
instance *cart* :: (*ring*,*finite*) *ring* **by** (*intro-classes*)
instance *cart* :: (*semiring-1-cancel*,*finite*) *semiring-1-cancel* **by** (*intro-classes*)
instance *cart* :: (*comm-semiring-1*,*finite*) *comm-semiring-1* **by** (*intro-classes*)

instance *cart* :: (*ring-1*,*finite*) *ring-1* **..**

instance *cart* :: (*real-algebra*,*finite*) *real-algebra*
  apply *intro-classes*
  apply (*simp-all add*: *vector-scaleR-def field-simps*)
  apply *vector*
  apply *vector*
  done

instance *cart* :: (*real-algebra-1*,*finite*) *real-algebra-1* **..**

lemma *of-nat-index*:
  (*of-nat n* :: *'a*::*semiring-1* ^ *'n*)$*i* = *of-nat n*
  apply (*induct n*)
  apply *vector*
  apply *vector*
  done

lemma *one-index*[*simp*]:
  (*1* :: *'a*::*one* ^ *'n*)$*i* = *1* **by** *vector*

instance *cart* :: (*semiring-char-0*,*finite*) *semiring-char-0*
proof (*intro-classes*)
  fix *m n* ::*nat*
  show (*of-nat m* :: *'a* ^ *'b*) = *of-nat n* ⟷ *m* = *n*
    by (*simp add*: *Cart-eq of-nat-index*)
qed

instance *cart* :: (*comm-ring-1*,*finite*) *comm-ring-1* **by** *intro-classes*
instance *cart* :: (*ring-char-0*,*finite*) *ring-char-0* **by** *intro-classes*

lemma *vector-smult-assoc*: *a* ∗*s* (*b* ∗*s* *x*) = ((*a*::*'a*::*semigroup-mult*) ∗ *b*) ∗*s* *x*
  by (*vector mult-assoc*)
lemma *vector-sadd-rdistrib*: ((*a*::*'a*::*semiring*) + *b*) ∗*s* *x* = *a* ∗*s* *x* + *b* ∗*s* *x*
  by (*vector field-simps*)
lemma *vector-add-ldistrib*: (*c*::*'a*::*semiring*) ∗*s* (*x* + *y*) = *c* ∗*s* *x* + *c* ∗*s* *y*
  by (*vector field-simps*)
lemma *vector-smult-lzero*[*simp*]: (*0*::*'a*::*mult-zero*) ∗*s* *x* = *0* **by** *vector*
lemma *vector-smult-lid*[*simp*]: (*1*::*'a*::*monoid-mult*) ∗*s* *x* = *x* **by** *vector*
lemma *vector-ssub-ldistrib*: (*c*::*'a*::*ring*) ∗*s* (*x* − *y*) = *c* ∗*s* *x* − *c* ∗*s* *y*
  by (*vector field-simps*)
lemma *vector-smult-rneg*: (*c*::*'a*::*ring*) ∗*s* −*x* = −(*c* ∗*s* *x*) **by** *vector*
lemma *vector-smult-lneg*: − (*c*::*'a*::*ring*) ∗*s* *x* = −(*c* ∗*s* *x*) **by** *vector*
lemma *vector-sneg-minus1*: −*x* = (− (*1*::*'a*::*ring-1*)) ∗*s* *x* **by** *vector*
lemma *vector-smult-rzero*[*simp*]: *c* ∗*s* *0* = (*0*::*'a*::*mult-zero* ^ *'n*) **by** *vector*
lemma *vector-sub-rdistrib*: ((*a*::*'a*::*ring*) − *b*) ∗*s* *x* = *a* ∗*s* *x* − *b* ∗*s* *x*
  by (*vector field-simps*)

lemma *vec-eq*[*simp*]: (*vec m* = *vec n*) ⟷ (*m* = *n*)
  by (*simp add*: *Cart-eq*)

abbreviation *inner-bullet* (**infix** · *70*) **where** $x \cdot y \equiv inner\ x\ y$

13.4 A connectedness or intermediate value lemma with several applications.

lemma *connected-real-lemma*:
fixes $f :: real \Rightarrow 'a::metric\text{-}space$
assumes *ab*: $a \leq b$ **and** *fa*: $f\ a \in e1$ **and** *fb*: $f\ b \in e2$
and *dst*: $\bigwedge e\ x.\ a <= x \Longrightarrow x <= b \Longrightarrow 0 < e ==> \exists\, d > 0.\ \forall\, y.\ abs(y - x) < d \longrightarrow dist(f\ y)\ (f\ x) < e$
and *e1*: $\forall\, y \in e1.\ \exists\, e > 0.\ \forall\, y'.\ dist\ y'\ y < e \longrightarrow y' \in e1$
and *e2*: $\forall\, y \in e2.\ \exists\, e > 0.\ \forall\, y'.\ dist\ y'\ y < e \longrightarrow y' \in e2$
and *e12*: $\sim(\exists\, x \geq a.\ x <= b \wedge f\ x \in e1 \wedge f\ x \in e2)$
shows $\exists\, x \geq a.\ x <= b \wedge f\ x \notin e1 \wedge f\ x \notin e2$ (**is** $\exists\ x.\ ?P\ x$)
proof−
let *?S* = $\{c.\ \forall\, x \geq a.\ x <= c \longrightarrow f\ x \in e1\}$
have *Se*: $\exists\, x.\ x \in ?S$ **apply** (*rule exI*[**where** x=a]) **by** (*auto simp add*: *fa*)
have *Sub*: $\exists\, y.\ isUb\ UNIV\ ?S\ y$
apply (*rule exI*[**where** x= b])
using *ab fb e12* **by** (*auto simp add*: *isUb-def setle-def*)
from *reals-complete*[*OF Se Sub*] **obtain** *l* **where**
l: *isLub UNIV ?S l***by** *blast*
have *alb*: $a \leq l\ l \leq b$ **using** *l ab fa fb e12*
apply (*auto simp add*: *isLub-def leastP-def isUb-def setle-def setge-def*)
by (*metis linorder-linear*)
have *ale1*: $\forall\, z \geq a.\ z < l \longrightarrow f\ z \in e1$ **using** *l*
apply (*auto simp add*: *isLub-def leastP-def isUb-def setle-def setge-def*)
by (*metis linorder-linear not-le*)
have *th1*: $\bigwedge z\ x\ e\ d :: real.\ z <= x + e \Longrightarrow e < d ==> z < x \vee abs(z - x) < d$ **by** *arith*
have *th2*: $\bigwedge e\ x :: real.\ 0 < e ==> \sim(x + e <= x)$ **by** *arith*
have *th3*: $\bigwedge d::real.\ d > 0 \Longrightarrow \exists\, e > 0.\ e < d$ **by** *dlo*
{assume *le2*: $f\ l \in e2$
from *le2 fa fb e12 alb* **have** *la*: $l \neq a$ **by** *metis*
hence *lap*: $l - a > 0$ **using** *alb* **by** *arith*
from *e2*[*rule-format*, *OF le2*] **obtain** *e* **where**
e: $e > 0\ \forall\, y.\ dist\ y\ (f\ l) < e \longrightarrow y \in e2$ **by** *metis*
from *dst*[*OF alb e*(*1*)] **obtain** *d* **where**
d: $d > 0\ \forall\, y.\ |y - l| < d \longrightarrow dist\ (f\ y)\ (f\ l) < e$ **by** *metis*
have $\exists\, d'.\ d' < d \wedge d' > 0 \wedge l - d' > a$ **using** *lap d*(*1*)
apply *ferrack* **by** *arith*
then obtain d' **where** d': $d' > 0\ d' < d\ l - d' > a$ **by** *metis*
from *d e* **have** *th0*: $\forall\, y.\ |y - l| < d \longrightarrow f\ y \in e2$ **by** *metis*
from *th0*[*rule-format*, *of* $l - d'$] d' **have** $f\ (l - d') \in e2$ **by** *auto*
moreover
have $f\ (l - d') \in e1$ **using** *ale1*[*rule-format*, *of* $l - d'$] d' **by** *auto*
ultimately have *False* **using** *e12 alb* d' **by** *auto*}
moreover

```
    {assume le1: f l ∈ e1
    from le1 fa fb e12 alb have lb: l ≠ b by metis
      hence blp: b − l > 0 using alb by arith
      from e1[rule-format, OF le1] obtain e where
        e: e > 0 ∀ y. dist y (f l) < e ⟶ y ∈ e1 by metis
      from dst[OF alb e(1)] obtain d where
        d: d > 0 ∀ y. |y − l| < d ⟶ dist (f y) (f l) < e by metis
      have ∃ d'. d' < d ∧ d' >0 using d(1) by dlo
      then obtain d' where d': d' > 0 d' < d by metis
      from d e have th0: ∀ y. |y − l| < d ⟶ f y ∈ e1 by auto
      hence ∀ y. l ≤ y ∧ y ≤ l + d' ⟶ f y ∈ e1 using d' by auto
      with ale1 have ∀ y. a ≤ y ∧ y ≤ l + d' ⟶ f y ∈ e1 by auto
      with l d' have False
        by (auto simp add: isLub-def isUb-def setle-def setge-def leastP-def) }
    ultimately show ?thesis using alb by metis
qed
```

One immediately useful corollary is the existence of square roots! — Should help to get rid of all the development of square-root for reals as a special case

```
lemma square-bound-lemma: (x::real) < (1 + x) * (1 + x)
proof−
  have (x + 1/2) ^2 + 3/4 > 0 using zero-le-power2[of x+1/2] by arith
  thus ?thesis by (simp add: field-simps power2-eq-square)
qed

lemma square-continuous: 0 < (e::real) ==> ∃ d. 0 < d ∧ (∀ y. abs(y − x) < d
⟶ abs(y * y − x * x) < e)
 using isCont-power[OF isCont-ident, of 2, unfolded isCont-def LIM-eq, rule-format,
of e x] apply (auto simp add: power2-eq-square)
  apply (rule-tac x=s in exI)
  apply auto
  apply (erule-tac x=y in allE)
  apply auto
  done

lemma real-le-lsqrt: 0 <= x ⟹ 0 <= y ⟹ x <= y ^2 ==> sqrt x <= y
  using real-sqrt-le-iff[of x y ^2] by simp

lemma real-le-rsqrt: x ^2 ≤ y ⟹ x ≤ sqrt y
  using real-sqrt-le-mono[of x ^2 y] by simp

lemma real-less-rsqrt: x ^2 < y ⟹ x < sqrt y
  using real-sqrt-less-mono[of x ^2 y] by simp

lemma sqrt-even-pow2: assumes n: even n
  shows sqrt(2 ^ n) = 2 ^ (n div 2)
proof−
  from n obtain m where m: n = 2*m unfolding even-mult-two-ex ..
```

from *m* **have** *sqrt*(*2* ^ *n*) = *sqrt* ((*2* ^ *m*) ^ *2*)
by (*simp only*: *power-mult*[*symmetric*] *mult-commute*)
then show *?thesis* **using** *m* **by** *simp*
qed

lemma *real-div-sqrt*: *0* <= *x* ==> *x* / *sqrt*(*x*) = *sqrt*(*x*)
apply (*cases x = 0*, *simp-all*)
using *sqrt-divide-self-eq*[*of x*]
apply (*simp add*: *inverse-eq-divide field-simps*)
done

Hence derive more interesting properties of the norm.

lemma *norm-mul*[*simp*]: *norm*(*a* *s *x*) = *abs*(*a*) * *norm x*
by (*simp add*: *norm-vector-def setL2-right-distrib abs-mult*)

lemma *norm-eq-0-dot*: (*norm x = 0*) ⟷ (*inner x x* = (*0*::*real*))
by (*simp add*: *norm-vector-def setL2-def power2-eq-square*)
lemma *norm-eq-0-imp*: *norm x = 0* ==> *x* = (*0*::*real* ^'*n*) **by** (*metis norm-eq-zero*)
lemma *vector-mul-eq-0*[*simp*]: (*a* *s *x = 0*) ⟷ *a* = (*0*::'*a*::*idom*) ∨ *x = 0*
by *vector*
lemma *vector-mul-lcancel*[*simp*]: *a* *s *x* = *a* *s *y* ⟷ *a* = (*0*::*real*) ∨ *x* = *y*
by (*metis eq-iff-diff-eq-0 vector-mul-eq-0 vector-ssub-ldistrib*)
lemma *vector-mul-rcancel*[*simp*]: *a* *s *x* = *b* *s *x* ⟷ (*a*::*real*) = *b* ∨ *x = 0*
by (*metis eq-iff-diff-eq-0 vector-mul-eq-0 vector-sub-rdistrib*)
lemma *vector-mul-lcancel-imp*: *a* ≠ (*0*::*real*) ==> *a* *s *x* = *a* *s *y* ==> (*x* = *y*)
by (*metis vector-mul-lcancel*)
lemma *vector-mul-rcancel-imp*: *x* ≠ *0* ⟹ (*a*::*real*) *s *x* = *b* *s *x* ==> *a* = *b*
by (*metis vector-mul-rcancel*)

lemma *norm-cauchy-schwarz*:
shows *inner x y* <= *norm x* * *norm y*
using *Cauchy-Schwarz-ineq2*[*of x y*] **by** *auto*

lemma *norm-cauchy-schwarz-abs*:
shows |*inner x y*| ≤ *norm x* * *norm y*
by (*rule Cauchy-Schwarz-ineq2*)

lemma *norm-triangle-sub*:
fixes *x y* :: '*a*::*real-normed-vector*
shows *norm x* ≤ *norm y* + *norm* (*x* − *y*)
using *norm-triangle-ineq*[*of y x* − *y*] **by** (*simp add*: *field-simps*)

lemma *component-le-norm*: |*x*$*i*| <= *norm x*
apply (*simp add*: *norm-vector-def*)
apply (*rule member-le-setL2*, *simp-all*)
done

lemma *norm-bound-component-le*: *norm x* <= *e* ==> |*x*$*i*| <= *e*

by (*metis component-le-norm order-trans*)

lemma *norm-bound-component-lt*: *norm x < e ==> |x$i| < e*
by (*metis component-le-norm basic-trans-rules(21)*)

lemma *norm-le-l1*: *norm x <= setsum(λi. |x$i|) UNIV*
by (*simp add*: *norm-vector-def setL2-le-setsum*)

lemma *real-abs-norm*: *|norm x| = norm x*
by (*rule abs-norm-cancel*)
lemma *real-abs-sub-norm*: *|norm x − norm y| <= norm(x − y)*
by (*rule norm-triangle-ineq3*)
lemma *norm-le*: *norm(x) <= norm(y) ⟷ x · x <= y · y*
by (*simp add*: *norm-eq-sqrt-inner*)
lemma *norm-lt*: *norm(x) < norm(y) ⟷ x · x < y · y*
by (*simp add*: *norm-eq-sqrt-inner*)
lemma *norm-eq*: *norm(x) = norm (y) ⟷ x · x = y · y*
apply(*subst order-eq-iff*) **unfolding** *norm-le* **by** *auto*
lemma *norm-eq-1*: *norm(x) = 1 ⟷ x · x = 1*
unfolding *norm-eq-sqrt-inner* **by** *auto*

Squaring equations and inequalities involving norms.

lemma *dot-square-norm*: *x · x = norm(x) ^2*
by (*simp add*: *norm-eq-sqrt-inner*)

lemma *norm-eq-square*: *norm(x) = a ⟷ 0 <= a ∧ x · x = a^2*
by (*auto simp add*: *norm-eq-sqrt-inner*)

lemma *real-abs-le-square-iff*: *|x| ≤ |y| ⟷ (x::real) ^2 ≤ y ^2*
proof
assume $|x| \leq |y|$
then have $|x|^2 \leq |y|^2$ **by** (*rule power-mono*, *simp*)
then show $x^2 \leq y^2$ **by** *simp*
next
assume $x^2 \leq y^2$
then have *sqrt* $(x^2) \leq$ *sqrt* (y^2) **by** (*rule real-sqrt-le-mono*)
then show $|x| \leq |y|$ **by** *simp*
qed

lemma *norm-le-square*: *norm(x) <= a ⟷ 0 <= a ∧ x · x <= a^2*
apply (*simp add*: *dot-square-norm real-abs-le-square-iff*[*symmetric*])
using *norm-ge-zero*[*of x*]
apply *arith*
done

lemma *norm-ge-square*: *norm(x) >= a ⟷ a <= 0 ∨ x · x >= a ^ 2*
apply (*simp add*: *dot-square-norm real-abs-le-square-iff*[*symmetric*])
using *norm-ge-zero*[*of x*]
apply *arith*

done

lemma *norm-lt-square*: $norm(x) < a \longleftrightarrow 0 < a \wedge x \cdot x < a\hat{}2$
by (*metis not-le norm-ge-square*)
lemma *norm-gt-square*: $norm(x) > a \longleftrightarrow a < 0 \vee x \cdot x > a\hat{}2$
by (*metis norm-le-square not-less*)

Dot product in terms of the norm rather than conversely.

lemmas *inner-simps* = *inner.add-left inner.add-right inner.diff-right inner.diff-left*

inner.scaleR-left inner.scaleR-right

lemma *dot-norm*: $x \cdot y = (norm(x + y)\ \hat{}\ 2 - norm\ x\ \hat{}\ 2 - norm\ y\ \hat{}\ 2) / 2$
unfolding *power2-norm-eq-inner inner-simps inner-commute* **by** *auto*

lemma *dot-norm-neg*: $x \cdot y = ((norm\ x\ \hat{}\ 2 + norm\ y\ \hat{}\ 2) - norm(x - y)\ \hat{}\ 2)$
$/ 2$
unfolding *power2-norm-eq-inner inner-simps inner-commute* **by**(*auto simp add*:*algebra-simps*)

Equality of vectors in terms of $op \cdot$ products.

lemma *vector-eq*: $x = y \longleftrightarrow x \cdot x = x \cdot y \wedge y \cdot y = x \cdot x$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *?lhs* **then show** *?rhs* **by** *simp*
next
assume *?rhs*
then have $x \cdot x - x \cdot y = 0 \wedge x \cdot y - y \cdot y = 0$ **by** *simp*
hence $x \cdot (x - y) = 0 \wedge y \cdot (x - y) = 0$ **by** (*simp add*: *inner-simps inner-commute*)
then have $(x - y) \cdot (x - y) = 0$ **by** (*simp add*: *field-simps inner-simps*
inner-commute)
then show $x = y$ **by** (*simp*)
qed

13.5 General linear decision procedure for normed spaces.

lemma *norm-cmul-rule-thm*:
fixes $x :: 'a$::*real-normed-vector*
shows $b >= norm(x) ==> |c| * b >= norm(scaleR\ c\ x)$
unfolding *norm-scaleR*
apply (*erule mult-mono1*)
apply *simp*
done

lemma *norm-add-rule-thm*:
fixes *x1 x2* $:: 'a$::*real-normed-vector*
shows $norm\ x1 \leq b1 \Longrightarrow norm\ x2 \leq b2 \Longrightarrow norm\ (x1 + x2) \leq b1 + b2$
by (*rule order-trans* [*OF norm-triangle-ineq add-mono*])

lemma *ge-iff-diff-ge-0*: $(a::'a$::*linordered-ring*$) \geq b == a - b \geq 0$

by (*simp add*: *field-simps*)

lemma *pth-1*:
fixes *x* :: *'a*::*real-normed-vector*
shows *x* == *scaleR 1 x* **by** *simp*

lemma *pth-2*:
fixes *x* :: *'a*::*real-normed-vector*
shows *x* − *y* == *x* + −*y* **by** (*atomize* (*full*)) *simp*

lemma *pth-3*:
fixes *x* :: *'a*::*real-normed-vector*
shows − *x* == *scaleR* (−*1*) *x* **by** *simp*

lemma *pth-4*:
fixes *x* :: *'a*::*real-normed-vector*
shows *scaleR 0 x* == *0* **and** *scaleR c 0* = (*0*::*'a*) **by** *simp-all*

lemma *pth-5*:
fixes *x* :: *'a*::*real-normed-vector*
shows *scaleR c* (*scaleR d x*) == *scaleR* (*c* ∗ *d*) *x* **by** *simp*

lemma *pth-6*:
fixes *x* :: *'a*::*real-normed-vector*
shows *scaleR c* (*x* + *y*) == *scaleR c x* + *scaleR c y*
by (*simp add*: *scaleR-right-distrib*)

lemma *pth-7*:
fixes *x* :: *'a*::*real-normed-vector*
shows *0* + *x* == *x* **and** *x* + *0* == *x* **by** *simp-all*

lemma *pth-8*:
fixes *x* :: *'a*::*real-normed-vector*
shows *scaleR c x* + *scaleR d x* == *scaleR* (*c* + *d*) *x*
by (*simp add*: *scaleR-left-distrib*)

lemma *pth-9*:
fixes *x* :: *'a*::*real-normed-vector* **shows**
(*scaleR c x* + *z*) + *scaleR d x* == *scaleR* (*c* + *d*) *x* + *z*
scaleR c x + (*scaleR d x* + *z*) == *scaleR* (*c* + *d*) *x* + *z*
(*scaleR c x* + *w*) + (*scaleR d x* + *z*) == *scaleR* (*c* + *d*) *x* + (*w* + *z*)
by (*simp-all add*: *algebra-simps*)

lemma *pth-a*:
fixes *x* :: *'a*::*real-normed-vector*
shows *scaleR 0 x* + *y* == *y* **by** *simp*

lemma *pth-b*:
fixes *x* :: *'a*::*real-normed-vector* **shows**

scaleR c x + scaleR d y == scaleR c x + scaleR d y
(scaleR c x + z) + scaleR d y == scaleR c x + (z + scaleR d y)
scaleR c x + (scaleR d y + z) == scaleR c x + (scaleR d y + z)
(scaleR c x + w) + (scaleR d y + z) == scaleR c x + (w + (scaleR d y + z))
by (*simp-all add*: *algebra-simps*)

lemma *pth-c*:
fixes *x* :: *'a*::*real-normed-vector* **shows**
scaleR c x + scaleR d y == scaleR d y + scaleR c x
(scaleR c x + z) + scaleR d y == scaleR d y + (scaleR c x + z)
scaleR c x + (scaleR d y + z) == scaleR d y + (scaleR c x + z)
(scaleR c x + w) + (scaleR d y + z) == scaleR d y + ((scaleR c x + w) + z)
by (*simp-all add*: *algebra-simps*)

lemma *pth-d*:
fixes *x* :: *'a*::*real-normed-vector*
shows *x + 0 == x* **by** *simp*

lemma *norm-imp-pos-and-ge*:
fixes *x* :: *'a*::*real-normed-vector*
shows *norm x == n ⟹ norm x ≥ 0 ∧ n ≥ norm x*
by *atomize auto*

lemma *real-eq-0-iff-le-ge-0*: *(x*::*real) = 0 == x ≥ 0 ∧ −x ≥ 0* **by** *arith*

lemma *norm-pths*:
fixes *x* :: *'a*::*real-normed-vector* **shows**
x = y ⟷ norm (x − y) ≤ 0
x ≠ y ⟷ ¬ (norm (x − y) ≤ 0)
using *norm-ge-zero*[*of x − y*] **by** *auto*

use *normarith.ML*

method-setup *norm* = ⟪ *Scan.succeed* (*SIMPLE-METHOD' o NormArith.norm-arith-tac*)
⟫ *Proves simple linear statements about vector norms*

Hence more metric properties.

lemma *dist-triangle-alt*:
fixes *x y z* :: *'a*::*metric-space*
shows *dist y z <= dist x y + dist x z*
by (*rule dist-triangle3*)

lemma *dist-pos-lt*:
fixes *x y* :: *'a*::*metric-space*
shows *x ≠ y ==> 0 < dist x y*
by (*simp add*: *zero-less-dist-iff*)

lemma *dist-nz*:
fixes *x y* :: *'a*::*metric-space*

shows $x \neq y \longleftrightarrow 0 < dist\ x\ y$
by (*simp add*: *zero-less-dist-iff*)

lemma *dist-triangle-le*:
fixes $x\ y\ z :: 'a$::*metric-space*
shows $dist\ x\ z + dist\ y\ z <= e \Longrightarrow dist\ x\ y <= e$
by (*rule order-trans* [*OF dist-triangle2*])

lemma *dist-triangle-lt*:
fixes $x\ y\ z :: 'a$::*metric-space*
shows $dist\ x\ z + dist\ y\ z < e ==> dist\ x\ y < e$
by (*rule le-less-trans* [*OF dist-triangle2*])

lemma *dist-triangle-half-l*:
fixes *x1 x2 y* :: $'a$::*metric-space*
shows $dist\ x1\ y < e\ /\ 2 \Longrightarrow dist\ x2\ y < e\ /\ 2 \Longrightarrow dist\ x1\ x2 < e$
by (*rule dist-triangle-lt* [**where** *z*=*y*], *simp*)

lemma *dist-triangle-half-r*:
fixes *x1 x2 y* :: $'a$::*metric-space*
shows $dist\ y\ x1 < e\ /\ 2 \Longrightarrow dist\ y\ x2 < e\ /\ 2 \Longrightarrow dist\ x1\ x2 < e$
by (*rule dist-triangle-half-l*, *simp-all add*: *dist-commute*)

lemma *norm-triangle-half-r*:
shows $norm\ (y - x1) < e\ /\ 2 \Longrightarrow norm\ (y - x2) < e\ /\ 2 \Longrightarrow norm\ (x1 - x2) < e$
using *dist-triangle-half-r* **unfolding** *dist-norm*[*THEN sym*] **by** *auto*

lemma *norm-triangle-half-l*: **assumes** $norm\ (x - y) < e\ /\ 2\ norm\ (x' - (y)) < e\ /\ 2$
shows $norm\ (x - x') < e$
using *dist-triangle-half-l*[*OF assms*[*unfolded dist-norm*[*THEN sym*]]]
unfolding *dist-norm*[*THEN sym*] **.**

lemma *norm-triangle-le*: $norm(x) + norm\ y <= e ==> norm(x + y) <= e$
by (*metis order-trans norm-triangle-ineq*)

lemma *norm-triangle-lt*: $norm(x) + norm(y) < e ==> norm(x + y) < e$
by (*metis basic-trans-rules*(*21*) *norm-triangle-ineq*)

lemma *dist-triangle-add*:
fixes $x\ y\ x'\ y' :: 'a$::*real-normed-vector*
shows $dist\ (x + y)\ (x' + y') <= dist\ x\ x' + dist\ y\ y'$
by *norm*

lemma *dist-mul*[*simp*]: $dist\ (c *s x)\ (c *s y) = |c| * dist\ x\ y$
unfolding *dist-norm vector-ssub-ldistrib*[*symmetric*] *norm-mul* **..**

lemma *dist-triangle-add-half*:
fixes $x\ x'\ y\ y'$ :: *'a*::*real-normed-vector*
shows $dist\ x\ x' < e\ /\ 2 \Longrightarrow dist\ y\ y' < e\ /\ 2 \Longrightarrow dist(x + y)\ (x' + y') < e$
by *norm*

lemma *setsum-component* [*simp*]:
fixes f:: *'a* $\Rightarrow$ (*'b*::*comm-monoid-add*) ^*'n*
shows $(setsum\ f\ S)\$i = setsum\ (\lambda x.\ (f\ x)\$i)\ S$
by (*cases finite S*, *induct S set*: *finite*, *simp-all*)

lemma *setsum-eq*: $setsum\ f\ S = (\chi\ i.\ setsum\ (\lambda x.\ (f\ x)\$i\ )\ S)$
by (*simp add*: *Cart-eq*)

lemma *setsum-clauses*:
shows $setsum\ f\ \{\} = 0$
and *finite S* $\Longrightarrow setsum\ f\ (insert\ x\ S) =$
(*if x* $\in$ *S then setsum f S else f x* + *setsum f S*)
by (*auto simp add*: *insert-absorb*)

lemma *setsum-cmul*:
fixes f:: *'c* $\Rightarrow$ (*'a*::*semiring-1*) ^*'n*
shows $setsum\ (\lambda x.\ c *s f\ x)\ S = c *s setsum\ f\ S$
by (*simp add*: *Cart-eq setsum-right-distrib*)

lemma *setsum-norm*:
fixes f :: *'a* $\Rightarrow$ *'b*::*real-normed-vector*
assumes *fS*: *finite S*
shows $norm\ (setsum\ f\ S) <= setsum\ (\lambda x.\ norm(f\ x))\ S$
proof(*induct rule*: *finite-induct*[*OF fS*])
case *1* **thus** *?case* **by** *simp*
next
case (*2 x S*)
from *2.hyps* **have** $norm\ (setsum\ f\ (insert\ x\ S)) \leq norm\ (f\ x) + norm\ (setsum\ f\ S)$ **by** (*simp add*: *norm-triangle-ineq*)
also have $\ldots \leq norm\ (f\ x) + setsum\ (\lambda x.\ norm(f\ x))\ S$
using *2.hyps* **by** *simp*
finally show *?case* **using** *2.hyps* **by** *simp*
qed

lemma *setsum-norm-le*:
fixes f :: *'a* $\Rightarrow$ *'b*::*real-normed-vector*
assumes *fS*: *finite S*
and *fg*: $\forall x \in S.\ norm\ (f\ x) \leq g\ x$
shows $norm\ (setsum\ f\ S) \leq setsum\ g\ S$
proof−
from *fg* **have** $setsum\ (\lambda x.\ norm(f\ x))\ S <= setsum\ g\ S$
by − (*rule setsum-mono*, *simp*)
then show *?thesis* **using** *setsum-norm*[*OF fS*, *of f*] *fg*
by *arith*

qed

lemma *setsum-norm-bound*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
assumes *fS*: *finite S*
and K: $\forall x \in S.\ norm\ (f\ x) \leq K$
shows *norm* (*setsum f S*) $\leq$ *of-nat* (*card S*) $*$ K
using *setsum-norm-le*[*OF fS K*] *setsum-constant*[*symmetric*]
by *simp*

lemma *setsum-vmul*:
fixes $f :: 'a \Rightarrow 'b$::*semiring-0*
assumes *fS*: *finite S*
shows *setsum f S* $*s$ v = *setsum* ($\lambda x.\ f\ x\ *s\ v$) S
proof(*induct rule*: *finite-induct*[*OF fS*])
case *1* **then show** *?case* **by** *simp*
next
case (*2 x F*)
from *2.hyps* **have** *setsum f* (*insert x F*) $*s$ v = (*f x* + *setsum f F*) $*s$ v
by *simp*
also have ... = *f x* $*s$ v + *setsum f F* $*s$ v
by (*simp add*: *vector-sadd-rdistrib*)
also have ... = *setsum* ($\lambda x.\ f\ x\ *s\ v$) (*insert x F*) **using** *2.hyps* **by** *simp*
finally show *?case* **.**
qed

lemma *setsum-group*:
assumes *fS*: *finite S* **and** *fT*: *finite T* **and** *fST*: *f* ' *S* $\subseteq$ *T*
shows *setsum* ($\lambda y.\ setsum\ g\ \{x.\ x \in S \wedge f\ x = y\}$) *T* = *setsum g S*

apply (*subst setsum-image-gen*[*OF fS*, *of g f*])
apply (*rule setsum-mono-zero-right*[*OF fT fST*])
by (*auto intro*: *setsum-0'*)

lemma *vsum-norm-allsubsets-bound*:
fixes f:: $'a \Rightarrow$ *real* ^ $'n$
assumes *fP*: *finite P* **and** *fPs*: $\bigwedge Q.\ Q \subseteq P \Longrightarrow norm\ (setsum\ f\ Q) \leq e$
shows *setsum* ($\lambda x.\ norm\ (f\ x)$) *P* $\leq$ *2* $*$ *real CARD*($'n$) $*$ *e*
proof−
let *?d* = *real CARD*($'n$)
let *?nf* = $\lambda x.\ norm\ (f\ x)$
let *?U* = *UNIV* :: $'n$ *set*
have *th0*: *setsum* ($\lambda x.\ setsum\ (\lambda i.\ |f\ x\ \$\ i|)\ ?U$) *P* = *setsum* ($\lambda i.\ setsum\ (\lambda x.$ $|f\ x\ \$\ i|)\ P$) *?U*
by (*rule setsum-commute*)
have *th1*: *2* $*$ *?d* $*$ *e* = *of-nat* (*card ?U*) $*$ (*2* $*$ *e*) **by** (*simp add*: *real-of-nat-def*)

have *setsum ?nf P* $\leq$ *setsum* (λx. *setsum* (λi. $|f\ x\ \$\ i|$) *?U*) *P*
apply (*rule setsum-mono*)
by (*rule norm-le-l1*)
also have ... $\leq$ *2* $*$ *?d* $*$ *e*
unfolding *th0 th1*
proof(*rule setsum-bounded*)
fix *i* **assume** *i*: $i \in$ *?U*
let *?Pp* = $\{x.\ x \in P \wedge f\ x\ \$\ i \geq 0\}$
let *?Pn* = $\{x.\ x \in P \wedge f\ x\ \$\ i < 0\}$
have *thp*: P = *?Pp* $\cup$ *?Pn* **by** *auto*
have *thp0*: *?Pp* $\cap$ *?Pn* $=\{\}$ **by** *auto*
have *PpP*: *?Pp* $\subseteq P$ **and** *PnP*: *?Pn* $\subseteq P$ **by** *blast+*
have *Ppe*:*setsum* (λx. $|f\ x\ \$\ i|$) *?Pp* $\leq e$
using *component-le-norm*[*of setsum* (λx. *f x*) *?Pp i*] *fPs*[*OF PpP*]
by (*auto intro*: *abs-le-D1*)
have *Pne*: *setsum* (λx. $|f\ x\ \$\ i|$) *?Pn* $\leq e$
using *component-le-norm*[*of setsum* (λx. $-$ *f x*) *?Pn i*] *fPs*[*OF PnP*]
by (*auto simp add*: *setsum-negf intro*: *abs-le-D1*)
have *setsum* (λx. $|f\ x\ \$\ i|$) P = *setsum* (λx. $|f\ x\ \$\ i|$) *?Pp* + *setsum* ($\lambda x$. $|f\ x\ \$\ i|$) *?Pn*
apply (*subst thp*)
apply (*rule setsum-Un-zero*)
using *fP thp0* **by** *auto*
also have ... $\leq$ *2*$*$*e* **using** *Pne Ppe* **by** *arith*
finally show *setsum* (λx. $|f\ x\ \$\ i|$) $P \leq$ *2*$*$*e* **.**
qed
finally show *?thesis* **.**
qed

lemma *dot-lsum*: *finite S* $\Longrightarrow$ *setsum f S* $\cdot$ *y* = *setsum* (λx. *f x* $\cdot$ *y*) *S*
apply(*induct rule*: *finite-induct*) **by**(*auto simp add*: *inner-simps*)

lemma *dot-rsum*: *finite S* $\Longrightarrow$ *y* $\cdot$ *setsum f S* = *setsum* (λx. *y* $\cdot$ *f x*) *S*
apply(*induct rule*: *finite-induct*) **by**(*auto simp add*: *inner-simps*)

13.6 Basis vectors in coordinate directions.

definition *basis k* = (χ *i*. *if i* = *k then 1 else 0*)

lemma *basis-component* [*simp*]: *basis k* \$ *i* = (*if k*=*i then 1 else 0*)
unfolding *basis-def* **by** *simp*

lemma *delta-mult-idempotent*:
(*if k*=*a then 1 else* (*0*::*'a*::*semiring-1*)) $*$ (*if k*=*a then 1 else 0*) = (*if k*=*a then 1 else 0*) **by** (*cases k*=*a*, *auto*)

lemma *norm-basis*:
shows *norm* (*basis k* :: *real* ^*'n*) = *1*
apply (*simp add*: *basis-def norm-eq-sqrt-inner*) **unfolding** *inner-vector-def*

apply (*vector delta-mult-idempotent*)
using *setsum-delta*[*of UNIV* :: *'n set k* λ*k. 1*::*real*] **by** *auto*

lemma *norm-basis-1*: *norm*(*basis 1* :: *real ^'n*::{*finite*,*one*}) = *1*
by (*rule norm-basis*)

lemma *vector-choose-size*: *0* <= *c* ==> ∃(*x*::*real^'n*). *norm x* = *c*
apply (*rule exI*[**where** *x*=*c* ∗*s basis arbitrary*])
by (*simp only*: *norm-mul norm-basis*)

lemma *vector-choose-dist*: **assumes** *e*: *0* <= *e*
shows ∃(*y*::*real^'n*). *dist x y* = *e*
proof−
from *vector-choose-size*[*OF e*] **obtain** *c*:: *real ^'n* **where** *norm c* = *e*
by *blast*
then have *dist x* (*x* − *c*) = *e* **by** (*simp add*: *dist-norm*)
then show *?thesis* **by** *blast*
qed

lemma *basis-inj*: *inj* (*basis* :: *'n* ⇒ *real ^'n*)
by (*simp add*: *inj-on-def Cart-eq*)

lemma *cond-value-iff*: *f* (*if b then x else y*) = (*if b then f x else f y*)
by *auto*

lemma *basis-expansion*:
setsum (λ*i*. (*x*$*i*) ∗*s basis i*) *UNIV* = (*x*::(*'a*::*ring-1*) *^'n*) (**is** *?lhs* = *?rhs* **is** *setsum ?f ?S* = -)
by (*auto simp add*: *Cart-eq cond-value-iff setsum-delta*[*of ?S*, **where** *?'b* = *'a*, *simplified*] *cong del*: *if-weak-cong*)

lemma *smult-conv-scaleR*: *c* ∗*s x* = *scaleR c x*
unfolding *vector-scalar-mult-def vector-scaleR-def* **by** *simp*

lemma *basis-expansion'*:
setsum (λ*i*. (*x*$*i*) ∗*R basis i*) *UNIV* = *x*
by (*rule basis-expansion* [**where** *'a*=*real*, *unfolded smult-conv-scaleR*])

lemma *basis-expansion-unique*:
setsum (λ*i*. *f i* ∗*s basis i*) *UNIV* = (*x*::(*'a*::*comm-ring-1*) *^'n*) ⟷ (∀ *i*. *f i* = *x*$*i*)
by (*simp add*: *Cart-eq setsum-delta cond-value-iff cong del*: *if-weak-cong*)

lemma *cond-application-beta*: (*if b then f else g*) *x* = (*if b then f x else g x*)
by *auto*

lemma *dot-basis*:
shows *basis i* · *x* = *x*$*i x* · (*basis i*) = (*x*$*i*)
unfolding *inner-vector-def* **by** (*auto simp add*: *basis-def cond-application-beta*

cond-value-iff setsum-delta cong del: *if-weak-cong*)

lemma *inner-basis*:
fixes $x :: 'a::\{real\text{-}inner, real\text{-}algebra\text{-}1\}\ \hat{}\ 'n$
shows $inner\ (basis\ i)\ x = inner\ 1\ (x\ \$\ i)$
and $inner\ x\ (basis\ i) = inner\ (x\ \$\ i)\ 1$
unfolding *inner-vector-def basis-def*
by (*auto simp add*: *cond-application-beta cond-value-iff setsum-delta cong del*: *if-weak-cong*)

lemma *basis-eq-0*: $basis\ i = (0::'a::semiring\text{-}1\hat{}'n) \longleftrightarrow False$
by (*auto simp add*: *Cart-eq*)

lemma *basis-nonzero*:
shows $basis\ k \neq (0::'a::semiring\text{-}1\ \hat{}'n)$
by (*simp add*: *basis-eq-0*)

lemma *vector-eq-ldot*: $(\forall x.\ x \cdot y = x \cdot z) \longleftrightarrow y = z$
proof
assume $\forall x.\ x \cdot y = x \cdot z$
hence $\forall x.\ x \cdot (y - z) = 0$ **by** (*simp add*: *inner-simps*)
hence $(y - z) \cdot (y - z) = 0$ **..**
thus $y = z$ **by** *simp*
qed *simp*

lemma *vector-eq-rdot*: $(\forall z.\ x \cdot z = y \cdot z) \longleftrightarrow x = y$
proof
assume $\forall z.\ x \cdot z = y \cdot z$
hence $\forall z.\ (x - y) \cdot z = 0$ **by** (*simp add*: *inner-simps*)
hence $(x - y) \cdot (x - y) = 0$ **..**
thus $x = y$ **by** *simp*
qed *simp*

13.7 Orthogonality.

definition $orthogonal\ x\ y \longleftrightarrow (x \cdot y = 0)$

lemma *orthogonal-basis*:
shows $orthogonal\ (basis\ i)\ x \longleftrightarrow x\$i = (0::real)$
by (*auto simp add*: *orthogonal-def inner-vector-def basis-def cond-value-iff cond-application-beta setsum-delta cong del*: *if-weak-cong*)

lemma *orthogonal-basis-basis*:
shows $orthogonal\ (basis\ i :: real\hat{}'n)\ (basis\ j) \longleftrightarrow i \neq j$
unfolding *orthogonal-basis*[*of i*] *basis-component*[*of j*] **by** *simp*

lemma *orthogonal-clauses*:
$orthogonal\ a\ 0$
$orthogonal\ a\ x \Longrightarrow orthogonal\ a\ (c *_R x)$

orthogonal a x ==> orthogonal a (−x)
orthogonal a x ⟹ orthogonal a y ==> orthogonal a (x + y)
orthogonal a x ⟹ orthogonal a y ==> orthogonal a (x − y)
orthogonal 0 a
*orthogonal x a ==> orthogonal (c *$_R$ *x) a*
orthogonal x a ==> orthogonal (−x) a
orthogonal x a ⟹ orthogonal y a ==> orthogonal (x + y) a
orthogonal x a ⟹ orthogonal y a ==> orthogonal (x − y) a
unfolding *orthogonal-def inner-simps* **by** *auto*

lemma *orthogonal-commute*: *orthogonal x y ⟷ orthogonal y x*
by (*simp add*: *orthogonal-def inner-commute*)

13.8 Linear functions.

definition
linear :: ('*a*::*real-vector* ⇒ '*b*::*real-vector*) ⇒ *bool* **where**
linear f ⟷ (∀ *x y*. *f*(*x* + *y*) = *f x* + *f y*) ∧ (∀ *c x*. *f*(*c* *$_R$ *x*) = *c* *$_R$ *f x*)

lemma *linearI*: **assumes** ⋀*x y*. *f* (*x* + *y*) = *f x* + *f y* ⋀*c x*. *f* (*c* *$_R$ *x*) = *c* *$_R$ *f x*
shows *linear f* **using** *assms* **unfolding** *linear-def* **by** *auto*

lemma *linear-compose-cmul*: *linear f ==> linear* (λ*x*. *c* *$_R$ *f x*)
by (*simp add*: *linear-def algebra-simps*)

lemma *linear-compose-neg*: *linear f ==> linear* (λ*x*. −(*f*(*x*)))
by (*simp add*: *linear-def*)

lemma *linear-compose-add*: *linear f ⟹ linear g ==> linear* (λ*x*. *f*(*x*) + *g*(*x*))
by (*simp add*: *linear-def algebra-simps*)

lemma *linear-compose-sub*: *linear f ⟹ linear g ==> linear* (λ*x*. *f x* − *g x*)
by (*simp add*: *linear-def algebra-simps*)

lemma *linear-compose*: *linear f ⟹ linear g ==> linear* (*g o f*)
by (*simp add*: *linear-def*)

lemma *linear-id*: *linear id* **by** (*simp add*: *linear-def id-def*)

lemma *linear-zero*: *linear* (λ*x*. *0*) **by** (*simp add*: *linear-def*)

lemma *linear-compose-setsum*:
assumes *fS*: *finite S* **and** *lS*: ∀ *a* ∈ *S*. *linear* (*f a*)
shows *linear*(λ*x*. *setsum* (λ*a*. *f a x*) *S*)
using *lS*
apply (*induct rule*: *finite-induct*[*OF fS*])
by (*auto simp add*: *linear-zero intro*: *linear-compose-add*)

lemma *linear-vmul-component*:
assumes *lf*: *linear f*
shows *linear* ($\lambda x.\ f\ x\ \$\ k\ *_R\ v$)
using *lf*
by (*auto simp add*: *linear-def algebra-simps*)

lemma *linear-0*: *linear f* ==> *f 0 = 0*
unfolding *linear-def*
apply *clarsimp*
apply (*erule allE*[**where** *x=0*::*'a*])
apply *simp*
done

lemma *linear-cmul*: *linear f* ==> $f(c *_R x) = c *_R f\ x$ **by** (*simp add*: *linear-def*)

lemma *linear-neg*: *linear f* ==> $f\ (-x) = -\ f\ x$
using *linear-cmul* [**where** *c=−1*] **by** *simp*

lemma *linear-add*: *linear f* ==> $f(x + y) = f\ x + f\ y$ **by** (*metis linear-def*)

lemma *linear-sub*: *linear f* ==> $f(x - y) = f\ x - f\ y$
by (*simp add*: *diff-def linear-add linear-neg*)

lemma *linear-setsum*:
assumes *lf*: *linear f* **and** *fS*: *finite S*
shows *f* (*setsum g S*) = *setsum* (*f o g*) *S*
proof (*induct rule*: *finite-induct*[*OF fS*])
case *1* **thus** *?case* **by** (*simp add*: *linear-0*[*OF lf*])
next
case (*2 x F*)
have *f* (*setsum g* (*insert x F*)) = *f* (*g x* + *setsum g F*) **using** *2.hyps*
by *simp*
also have ... = *f* (*g x*) + *f* (*setsum g F*) **using** *linear-add*[*OF lf*] **by** *simp*
also have ... = *setsum* (*f o g*) (*insert x F*) **using** *2.hyps* **by** *simp*
finally show *?case* **.**
qed

lemma *linear-setsum-mul*:
assumes *lf*: *linear f* **and** *fS*: *finite S*
shows *f* (*setsum* ($\lambda i.\ c\ i\ *_R\ v\ i$) *S*) = *setsum* ($\lambda i.\ c\ i\ *_R\ f$ (*v i*)) *S*
using *linear-setsum*[*OF lf fS*, *of* $\lambda i.\ c\ i\ *_R\ v\ i$, *unfolded o-def*]
linear-cmul[*OF lf*] **by** *simp*

lemma *linear-injective-0*:
assumes *lf*: *linear f*
shows *inj f* $\longleftrightarrow$ ($\forall x.\ f\ x = 0 \longrightarrow x = 0$)
proof−
have *inj f* $\longleftrightarrow$ ($\forall\ x\ y.\ f\ x = f\ y \longrightarrow x = y$) **by** (*simp add*: *inj-on-def*)
also have ... $\longleftrightarrow$ ($\forall\ x\ y.\ f\ x - f\ y = 0 \longrightarrow x - y = 0$) **by** *simp*

also have $\ldots \longleftrightarrow (\forall\ x\ y.\ f\ (x - y) = 0 \longrightarrow x - y = 0)$
by (*simp add*: *linear-sub*[*OF lf*])
also have $\ldots \longleftrightarrow (\forall\ x.\ f\ x = 0 \longrightarrow x = 0)$ **by** *auto*
finally show *?thesis* **.**
qed

lemma *linear-bounded*:
fixes f:: *real* ^'*m* ⇒ *real* ^'*n*
assumes *lf*: *linear f*
shows $\exists B.\ \forall x.$ *norm* $(f\ x) \leq B *$ *norm* x
proof−
let *?S* = *UNIV*:: '*m set*
let *?B* = *setsum* $(\lambda i.$ *norm*$(f($*basis* $i)))$ *?S*
have *fS*: *finite ?S* **by** *simp*
{fix x:: *real* ^ '*m*
let *?g* = $(\lambda i.\ (x\$i) *_R$ (*basis* $i)$ :: *real* ^ '*m*)
have *norm* $(f\ x)$ = *norm* $(f$ (*setsum* $(\lambda i.\ (x\$i) *_R$ (*basis* $i))$ *?S*))
by (*simp add*: *basis-expansion'*)
also have $\ldots$ = *norm* (*setsum* $(\lambda i.\ (x\$i) *_R f$ (*basis* $i))$*?S*)
using *linear-setsum*[*OF lf fS*, *of ?g*, *unfolded o-def*] *linear-cmul*[*OF lf*]
by *auto*
finally have *th0*: *norm* $(f\ x)$ = *norm* (*setsum* $(\lambda i.\ (x\$i) *_R f$ (*basis* $i))$*?S*) **.**
{fix i **assume** i: $i \in$ *?S*
from *component-le-norm*[*of x i*]
have *norm* $((x\$i) *_R f$ (*basis* i :: *real* ^'*m*)) ≤ *norm* $(f$ (*basis* $i)) *$ *norm* x
unfolding *norm-scaleR*
apply (*simp only*: *mult-commute*)
apply (*rule mult-mono*)
by (*auto simp add*: *field-simps*) **}**
then have *th*: $\forall i \in$ *?S*. *norm* $((x\$i) *_R f$ (*basis* i :: *real* ^'*m*)) ≤ *norm* $(f$ (*basis* $i)) *$ *norm* x **by** *metis*
from *setsum-norm-le*[*OF fS*, *of* $\lambda i.\ (x\$i) *_R$ $(f$ (*basis* $i))$, *OF th*]
have *norm* $(f\ x) \leq$ *?B* $*$ *norm* x **unfolding** *th0 setsum-left-distrib* **by** *metis***}**
then show *?thesis* **by** *blast*
qed

lemma *linear-bounded-pos*:
fixes f:: *real* ^'*n* ⇒ *real* ^'*m*
assumes *lf*: *linear f*
shows $\exists B > 0.\ \forall x.$ *norm* $(f\ x) \leq B *$ *norm* x
proof−
from *linear-bounded*[*OF lf*] **obtain** B **where**
B: $\forall x.$ *norm* $(f\ x) \leq B *$ *norm* x **by** *blast*
let *?K* = $|B| + 1$
have *Kp*: *?K* > *0* **by** *arith*
{assume C: $B < 0$
have *norm* (*1*::*real* ^ '*n*) > *0* **by** *simp*
with C **have** $B *$ *norm* (*1*:: *real* ^ '*n*) < *0*
by (*simp add*: *mult-less-0-iff*)

```
    with B[rule-format, of 1] norm-ge-zero[of f 1] have False by simp
   }
   then have Bp: B ≥ 0 by ferrack
   {fix x::real ^ 'n
    have norm (f x) ≤ ?K * norm x
    using B[rule-format, of x] norm-ge-zero[of x] norm-ge-zero[of f x] Bp
    apply (auto simp add: field-simps split add: abs-split)
    apply (erule order-trans, simp)
    done
  }
  then show ?thesis using Kp by blast
qed

lemma linear-conv-bounded-linear:
  fixes f :: real ^ - ⇒ real ^ -
  shows linear f ⟷ bounded-linear f
proof
  assume linear f
  show bounded-linear f
  proof
    fix x y show f (x + y) = f x + f y
      using ‹linear f› unfolding linear-def by simp
  next
    fix r x show f (scaleR r x) = scaleR r (f x)
      using ‹linear f› unfolding linear-def
      by (simp add: smult-conv-scaleR)
  next
    have ∃ B. ∀ x. norm (f x) ≤ B * norm x
      using ‹linear f› by (rule linear-bounded)
    thus ∃ K. ∀ x. norm (f x) ≤ norm x * K
      by (simp add: mult-commute)
  qed
next
  assume bounded-linear f
  then interpret f: bounded-linear f .
  show linear f
    unfolding linear-def smult-conv-scaleR
    by (simp add: f.add f.scaleR)
qed

lemma bounded-linearI': fixes f::real^'n ⇒ real^'m
  assumes ⋀x y. f (x + y) = f x + f y ⋀c x. f (c *R x) = c *R f x
  shows bounded-linear f unfolding linear-conv-bounded-linear[THEN sym]
  by(rule linearI[OF assms])
```

13.9 Bilinear functions.

```
definition bilinear f ⟷ (∀ x. linear(λy. f x y)) ∧ (∀ y. linear(λx. f x y))
```

lemma *bilinear-ladd*: *bilinear h ==> h (x + y) z = (h x z) + (h y z)*
by (*simp add*: *bilinear-def linear-def*)
lemma *bilinear-radd*: *bilinear h ==> h x (y + z) = (h x y) + (h x z)*
by (*simp add*: *bilinear-def linear-def*)

lemma *bilinear-lmul*: *bilinear h ==> h (c *$_R$ x) y = c *$_R$ (h x y)*
by (*simp add*: *bilinear-def linear-def*)

lemma *bilinear-rmul*: *bilinear h ==> h x (c *$_R$ y) = c *$_R$ (h x y)*
by (*simp add*: *bilinear-def linear-def*)

lemma *bilinear-lneg*: *bilinear h ==> h (- x) y = -(h x y)*
by (*simp only*: *scaleR-minus1-left* [*symmetric*] *bilinear-lmul*)

lemma *bilinear-rneg*: *bilinear h ==> h x (- y) = - h x y*
by (*simp only*: *scaleR-minus1-left* [*symmetric*] *bilinear-rmul*)

lemma (**in** *ab-group-add*) *eq-add-iff*: *x = x + y ⟷ y = 0*
using *add-imp-eq*[*of x y 0*] **by** *auto*

lemma *bilinear-lzero*:
assumes *bh*: *bilinear h* **shows** *h 0 x = 0*
using *bilinear-ladd*[*OF bh*, *of 0 0 x*]
by (*simp add*: *eq-add-iff field-simps*)

lemma *bilinear-rzero*:
assumes *bh*: *bilinear h* **shows** *h x 0 = 0*
using *bilinear-radd*[*OF bh*, *of x 0 0*]
by (*simp add*: *eq-add-iff field-simps*)

lemma *bilinear-lsub*: *bilinear h ==> h (x - y) z = h x z - h y z*
by (*simp add*: *diff-def bilinear-ladd bilinear-lneg*)

lemma *bilinear-rsub*: *bilinear h ==> h z (x - y) = h z x - h z y*
by (*simp add*: *diff-def bilinear-radd bilinear-rneg*)

lemma *bilinear-setsum*:
assumes *bh*: *bilinear h* **and** *fS*: *finite S* **and** *fT*: *finite T*
shows *h (setsum f S) (setsum g T) = setsum (λ(i,j). h (f i) (g j)) (S × T)*
proof−
have *h (setsum f S) (setsum g T) = setsum (λx. h (f x) (setsum g T)) S*
apply (*rule linear-setsum*[*unfolded o-def*])
using *bh fS* **by** (*auto simp add*: *bilinear-def*)
also have *... = setsum (λx. setsum (λy. h (f x) (g y)) T) S*
apply (*rule setsum-cong*, *simp*)
apply (*rule linear-setsum*[*unfolded o-def*])
using *bh fT* **by** (*auto simp add*: *bilinear-def*)
finally show *?thesis* **unfolding** *setsum-cartesian-product* **.**
qed

lemma *bilinear-bounded*:
fixes *h*:: *real ^'m ⇒ real^'n ⇒ real ^'k*
assumes *bh*: *bilinear h*
shows ∃ *B*. ∀ *x y*. *norm* (*h x y*) ≤ *B* ∗ *norm x* ∗ *norm y*
proof−
let *?M* = *UNIV* :: *'m set*
let *?N* = *UNIV* :: *'n set*
let *?B* = *setsum* (λ(*i*,*j*). *norm* (*h* (*basis i*) (*basis j*))) (*?M* × *?N*)
have *fM*: *finite ?M* **and** *fN*: *finite ?N* **by** *simp-all*
{fix *x*:: *real ^ 'm* **and** *y* :: *real^'n*
have *norm* (*h x y*) = *norm* (*h* (*setsum* (λ*i*. (*x*$*i*) ∗$_R$ *basis i*) *?M*) (*setsum* (λ*i*. (*y*$*i*) ∗$_R$ *basis i*) *?N*)) **unfolding** *basis-expansion'* **..**
also have ... = *norm* (*setsum* (λ (*i*,*j*). *h* ((*x*$*i*) ∗$_R$ *basis i*) ((*y*$*j*) ∗$_R$ *basis j*)) (*?M* × *?N*)) **unfolding** *bilinear-setsum*[*OF bh fM fN*] **..**
finally have *th*: *norm* (*h x y*) = ... **.**
have *norm* (*h x y*) ≤ *?B* ∗ *norm x* ∗ *norm y*
apply (*simp add*: *setsum-left-distrib th*)
apply (*rule setsum-norm-le*)
using *fN fM*
apply *simp*
apply (*auto simp add*: *bilinear-rmul*[*OF bh*] *bilinear-lmul*[*OF bh*] *field-simps simp del*: *scaleR-scaleR*)
apply (*rule mult-mono*)
apply (*auto simp add*: *zero-le-mult-iff component-le-norm*)
apply (*rule mult-mono*)
apply (*auto simp add*: *zero-le-mult-iff component-le-norm*)
done}
then show *?thesis* **by** *metis*
qed

lemma *bilinear-bounded-pos*:
fixes *h*:: *real ^'m ⇒ real^'n ⇒ real ^'k*
assumes *bh*: *bilinear h*
shows ∃ *B* > *0*. ∀ *x y*. *norm* (*h x y*) ≤ *B* ∗ *norm x* ∗ *norm y*
proof−
from *bilinear-bounded*[*OF bh*] **obtain** *B* **where**
B: ∀ *x y*. *norm* (*h x y*) ≤ *B* ∗ *norm x* ∗ *norm y* **by** *blast*
let *?K* = |*B*| + *1*
have *Kp*: *?K* > *0* **by** *arith*
have *KB*: *B* < *?K* **by** *arith*
{fix *x*::*real ^'m* **and** *y* :: *real ^'n*
from *KB Kp*
have *B* ∗ *norm x* ∗ *norm y* ≤ *?K* ∗ *norm x* ∗ *norm y*
apply −
apply (*rule mult-right-mono*, *rule mult-right-mono*)
by *auto*
then have *norm* (*h x y*) ≤ *?K* ∗ *norm x* ∗ *norm y*
using *B*[*rule-format*, *of x y*] **by** *simp***}**

with *Kp* **show** *?thesis* **by** *blast*
qed

lemma *bilinear-conv-bounded-bilinear*:
fixes h :: *real* ^ - ⇒ *real* ^ - ⇒ *real* ^ -
shows *bilinear* h ⟷ *bounded-bilinear* h
proof
assume *bilinear* h
show *bounded-bilinear* h
proof
fix x y z **show** $h\ (x + y)\ z = h\ x\ z + h\ y\ z$
using ‹*bilinear* h› **unfolding** *bilinear-def linear-def* **by** *simp*
next
fix x y z **show** $h\ x\ (y + z) = h\ x\ y + h\ x\ z$
using ‹*bilinear* h› **unfolding** *bilinear-def linear-def* **by** *simp*
next
fix r x y **show** $h\ (scaleR\ r\ x)\ y = scaleR\ r\ (h\ x\ y)$
using ‹*bilinear* h› **unfolding** *bilinear-def linear-def*
by (*simp add*: *smult-conv-scaleR*)
next
fix r x y **show** $h\ x\ (scaleR\ r\ y) = scaleR\ r\ (h\ x\ y)$
using ‹*bilinear* h› **unfolding** *bilinear-def linear-def*
by (*simp add*: *smult-conv-scaleR*)
next
have ∃ B. ∀ x y. $norm\ (h\ x\ y) \leq B * norm\ x * norm\ y$
using ‹*bilinear* h› **by** (*rule bilinear-bounded*)
thus ∃ K. ∀ x y. $norm\ (h\ x\ y) \leq norm\ x * norm\ y * K$
by (*simp add*: *mult-ac*)
qed
next
assume *bounded-bilinear* h
then interpret h: *bounded-bilinear* h **.**
show *bilinear* h
unfolding *bilinear-def linear-conv-bounded-linear*
using *h.bounded-linear-left h.bounded-linear-right*
by *simp*
qed

13.10 Adjoints.

definition $adjoint\ f = (SOME\ f'.\ \forall\ x\ y.\ f\ x \cdot y = x \cdot f'\ y)$

lemma *adjoint-unique*:
assumes ∀ x y. $inner\ (f\ x)\ y = inner\ x\ (g\ y)$
shows $adjoint\ f = g$
unfolding *adjoint-def*
proof (*rule some-equality*)
show ∀ x y. $inner\ (f\ x)\ y = inner\ x\ (g\ y)$ **using** *assms* **.**
next

fix h **assume** $\forall x\ y.\ inner\ (f\ x)\ y = inner\ x\ (h\ y)$
hence $\forall x\ y.\ inner\ x\ (g\ y) = inner\ x\ (h\ y)$ **using** *assms* **by** *simp*
hence $\forall x\ y.\ inner\ x\ (g\ y - h\ y) = 0$ **by** (*simp add*: *inner-diff-right*)
hence $\forall y.\ inner\ (g\ y - h\ y)\ (g\ y - h\ y) = 0$ **by** *simp*
hence $\forall y.\ h\ y = g\ y$ **by** *simp*
thus $h = g$ **by** (*simp add*: *ext*)
qed

lemma *choice-iff*: $(\forall x.\ \exists y.\ P\ x\ y) \longleftrightarrow (\exists f.\ \forall x.\ P\ x\ (f\ x))$ **by** *metis*

TODO: The following lemmas about adjoints should hold for any Hilbert space (i.e. complete inner product space). (see http://en.wikipedia.org/wiki/Hermitian_adjoint)

lemma *adjoint-works-lemma*:
fixes f:: *real* ^ $'n \Rightarrow$ *real* ^ $'m$
assumes *lf*: *linear f*
shows $\forall x\ y.\ f\ x \cdot y = x \cdot adjoint\ f\ y$
proof−
let *?N* = *UNIV* :: $'n$ *set*
let *?M* = *UNIV* :: $'m$ *set*
have *fN*: *finite ?N* **by** *simp*
have *fM*: *finite ?M* **by** *simp*
{fix y:: *real* ^ $'m$
let *?w* = $(\chi\ i.\ (f\ (basis\ i) \cdot y))$ :: *real* ^ $'n$
{fix x
have $f\ x \cdot y = f\ (setsum\ (\lambda i.\ (x\$i) *_R basis\ i)\ ?N) \cdot y$
by (*simp only*: *basis-expansion'*)
also have $\ldots = (setsum\ (\lambda i.\ (x\$i) *_R f\ (basis\ i))\ ?N) \cdot y$
unfolding *linear-setsum*[*OF lf fN*]
by (*simp add*: *linear-cmul*[*OF lf*])
finally have $f\ x \cdot y = x \cdot ?w$
apply (*simp only*:)
apply (*simp add*: *inner-vector-def setsum-left-distrib setsum-right-distrib setsum-commute*[*of* - *?M ?N*] *field-simps*)
done}
}
then show *?thesis* **unfolding** *adjoint-def*
some-eq-ex[*of* $\lambda f'.\ \forall x\ y.\ f\ x \cdot y = x \cdot f'\ y$]
using *choice-iff*[*of* $\lambda a\ b.\ \forall x.\ f\ x \cdot a = x \cdot b$]
by *metis*
qed

lemma *adjoint-works*:
fixes f:: *real* ^ $'n \Rightarrow$ *real* ^ $'m$
assumes *lf*: *linear f*
shows $x \cdot adjoint\ f\ y = f\ x \cdot y$
using *adjoint-works-lemma*[*OF lf*] **by** *metis*

lemma *adjoint-linear*:

fixes f:: *real ^'n ⇒ real ^'m*
assumes *lf*: *linear f*
shows *linear* (*adjoint f*)
unfolding *linear-def vector-eq-ldot*[**where** *'a=real^'n*, *symmetric*] **apply** *safe*
unfolding *inner-simps smult-conv-scaleR adjoint-works*[*OF lf*] **by** *auto*

lemma *adjoint-clauses*:
fixes f:: *real ^'n ⇒ real ^'m*
assumes *lf*: *linear f*
shows $x \cdot$ *adjoint f y* $= f\ x \cdot y$
and *adjoint f y* $\cdot\ x = y \cdot f\ x$
by (*simp-all add*: *adjoint-works*[*OF lf*] *inner-commute*)

lemma *adjoint-adjoint*:
fixes f:: *real ^'n ⇒ real ^'m*
assumes *lf*: *linear f*
shows *adjoint* (*adjoint f*) $= f$
by (*rule adjoint-unique*, *simp add*: *adjoint-clauses* [*OF lf*])

13.11 Matrix operations

Matrix notation. NB: an MxN matrix is of type ((*'a*, *'n*) *cart*, *'m*) *cart*, not ((*'a*, *'m*) *cart*, *'n*) *cart*

definition *matrix-matrix-mult* :: (*'a*::*semiring-1*) *^'n^'m ⇒ 'a ^'p^'n ⇒ 'a ^ 'p ^'m* (**infixl** ** 70*)
where *m ** m' == (χ i j. setsum (λk. ((m$i)$k) * ((m'$k)$j)) (UNIV :: 'n set)) ::'a ^ 'p ^'m*

definition *matrix-vector-mult* :: (*'a*::*semiring-1*) *^'n^'m ⇒ 'a ^'n ⇒ 'a ^ 'm* (**infixl** *v 70*)
where *m *v x ≡ (χ i. setsum (λj. ((m$i)$j) * (x$j)) (UNIV ::'n set)) :: 'a^'m*

definition *vector-matrix-mult* :: *'a ^ 'm ⇒* (*'a*::*semiring-1*) *^'n^'m ⇒ 'a ^'n* (**infixl** *v* 70*)
where *v v* m == (χ j. setsum (λi. ((m$i)$j) * (v$i)) (UNIV :: 'm set)) :: 'a^'n*

definition (*mat*::*'a*::*zero* => *'a ^'n^'n*) *k* = (χ *i j. if i = j then k else 0*)
definition *transpose* **where**
(*transpose*::*'a^'n^'m ⇒ 'a^'m^'n*) *A* = (χ *i j.* ((*A*$*j*)$*i*))
definition (*row*::*'m* => *'a ^'n^'m ⇒ 'a ^'n*) *i A* = (χ *j.* ((*A*$*i*)$*j*))
definition (*column*::*'n* =>*'a^'n^'m* =>*'a^'m*) *j A* = (χ *i.* ((*A*$*i*)$*j*))
definition *rows*(*A*::*'a^'n^'m*) = { *row i A* | *i. i* ∈ (*UNIV* :: *'m set*)}
definition *columns*(*A*::*'a^'n^'m*) = { *column i A* | *i. i* ∈ (*UNIV* :: *'n set*)}

lemma *mat-0*[*simp*]: *mat 0 = 0* **by** (*vector mat-def*)
lemma *matrix-add-ldistrib*: (*A ** (B + C*)) = (*A ** B*) + (*A ** C*)
by (*vector matrix-matrix-mult-def setsum-addf*[*symmetric*] *field-simps*)

lemma *matrix-mul-lid*:
fixes *A* :: *'a::semiring-1 ^ 'm ^ 'n*
shows *mat 1 ** A = A*
apply (*simp add*: *matrix-matrix-mult-def mat-def*)
apply *vector*
by (*auto simp only*: *cond-value-iff cond-application-beta setsum-delta'*[*OF finite*]
mult-1-left mult-zero-left if-True UNIV-I)

lemma *matrix-mul-rid*:
fixes *A* :: *'a::semiring-1 ^ 'm ^ 'n*
shows *A ** mat 1 = A*
apply (*simp add*: *matrix-matrix-mult-def mat-def*)
apply *vector*
by (*auto simp only*: *cond-value-iff cond-application-beta setsum-delta*[*OF finite*]
mult-1-right mult-zero-right if-True UNIV-I cong: *if-cong*)

lemma *matrix-mul-assoc*: *A ** (B ** C) = (A ** B) ** C*
apply (*vector matrix-matrix-mult-def setsum-right-distrib setsum-left-distrib mult-assoc*)
apply (*subst setsum-commute*)
apply *simp*
done

lemma *matrix-vector-mul-assoc*: *A *v (B *v x) = (A ** B) *v x*
apply (*vector matrix-matrix-mult-def matrix-vector-mult-def setsum-right-distrib*
setsum-left-distrib mult-assoc)
apply (*subst setsum-commute*)
apply *simp*
done

lemma *matrix-vector-mul-lid*: *mat 1 *v x = (x::'a::semiring-1 ^ 'n)*
apply (*vector matrix-vector-mult-def mat-def*)
by (*simp add*: *cond-value-iff cond-application-beta*
setsum-delta' cong del: *if-weak-cong*)

lemma *matrix-transpose-mul*: *transpose(A ** B) = transpose B ** transpose (A::'a::comm-semiring-1^-^-)*
by (*simp add*: *matrix-matrix-mult-def transpose-def Cart-eq mult-commute*)

lemma *matrix-eq*:
fixes *A B* :: *'a::semiring-1 ^ 'n ^ 'm*
shows *A = B* ⟷ (∀ *x*. *A *v x = B *v x*) (**is** *?lhs* ⟷ *?rhs*)
apply *auto*
apply (*subst Cart-eq*)
apply *clarify*
apply (*clarsimp simp add*: *matrix-vector-mult-def basis-def cond-value-iff cond-application-beta*
Cart-eq cong del: *if-weak-cong*)
apply (*erule-tac x=basis ia* **in** *allE*)
apply (*erule-tac x=i* **in** *allE*)
by (*auto simp add*: *basis-def cond-value-iff cond-application-beta setsum-delta*[*OF*

finite] *cong del*: *if-weak-cong*)

lemma *matrix-vector-mul-component*:
shows $((A::real\hat{}-\hat{}-) *v\ x)\$k = (A\$k) \cdot x$
by (*simp add*: *matrix-vector-mult-def inner-vector-def*)

lemma *dot-lmul-matrix*: $((x::real\ \hat{}-)\ v*\ A) \cdot y = x \cdot (A *v\ y)$
apply (*simp add*: *inner-vector-def matrix-vector-mult-def vector-matrix-mult-def setsum-left-distrib setsum-right-distrib mult-ac*)
apply (*subst setsum-commute*)
by *simp*

lemma *transpose-mat*: *transpose* (*mat n*) = *mat n*
by (*vector transpose-def mat-def*)

lemma *transpose-transpose*: *transpose*(*transpose A*) = *A*
by (*vector transpose-def*)

lemma *row-transpose*:
fixes *A*:: *'a*::*semiring-1*$\hat{}-\hat{}$-
shows *row i* (*transpose A*) = *column i A*
by (*simp add*: *row-def column-def transpose-def Cart-eq*)

lemma *column-transpose*:
fixes *A*:: *'a*::*semiring-1*$\hat{}-\hat{}$-
shows *column i* (*transpose A*) = *row i A*
by (*simp add*: *row-def column-def transpose-def Cart-eq*)

lemma *rows-transpose*: *rows*(*transpose* (*A*::*'a*::*semiring-1*$\hat{}-\hat{}$-)) = *columns A*
by (*auto simp add*: *rows-def columns-def row-transpose intro*: *set-ext*)

lemma *columns-transpose*: *columns*(*transpose* (*A*::*'a*::*semiring-1*$\hat{}-\hat{}$-)) = *rows A*
by (*metis transpose-transpose rows-transpose*)

Two sometimes fruitful ways of looking at matrix-vector multiplication.

lemma *matrix-mult-dot*: $A *v\ x = (\chi\ i.\ A\$i \cdot x)$
by (*simp add*: *matrix-vector-mult-def inner-vector-def*)

lemma *matrix-mult-vsum*: (*A*::*'a*::*comm-semiring-1*$\hat{}$*'n*$\hat{}$*'m*) *v x* = *setsum* ($\lambda i.$ (*x*\$*i*) *s column i A*) (*UNIV*:: *'n set*)
by (*simp add*: *matrix-vector-mult-def Cart-eq column-def mult-commute*)

lemma *vector-componentwise*:
(*x*::*'a*::*ring-1*$\hat{}$*'n*) = ($\chi\ j.$ *setsum* ($\lambda i.$ (*x*\$*i*) * (*basis i* :: *'a*$\hat{}$*'n*)\$*j*) (*UNIV* :: *'n set*))
apply (*subst basis-expansion*[*symmetric*])
by (*vector Cart-eq setsum-component*)

lemma *linear-componentwise*:

fixes *f*:: *real ^'m ⇒ real ^ -*
assumes *lf*: *linear f*
shows *(f x)$j = setsum (λi. (x$i) * (f (basis i)$j)) (UNIV :: 'm set)* **(is** *?lhs = ?rhs)*
proof−
let *?M = (UNIV :: 'm set)*
let *?N = (UNIV :: 'n set)*
have *fM*: *finite ?M* **by** *simp*
have *?rhs = (setsum (λi.(x$i) *R f (basis i)) ?M)$j*
unfolding *vector-smult-component*[*symmetric*] *smult-conv-scaleR*
unfolding *setsum-component*[*of (λi.(x$i) *R f (basis i :: real^'m)) ?M*]
..
then show *?thesis* **unfolding** *linear-setsum-mul*[*OF lf fM*, *symmetric*] *basis-expansion'*
..
qed

Inverse matrices (not necessarily square)

definition *invertible*(*A*::*'a*::*semiring-1^'n^'m*) ⟷ (∃ *A'*::*'a^'m^'n*. *A ** *A' = mat 1 ∧ A' ** A = mat 1*)

definition *matrix-inv*(*A*:: *'a*::*semiring-1^'n^'m*) =
(*SOME A'*::*'a^'m^'n*. *A ** *A' = mat 1 ∧ A' ** A = mat 1*)

Correspondence between matrices and linear operators.

definition *matrix*:: (*'a*::{*plus*,*times*, *one*, *zero*}*^'m ⇒ 'a ^ 'n*) ⇒ *'a^'m^'n*
where *matrix f = (χ i j. (f(basis j))$i)*

lemma *matrix-vector-mul-linear*: *linear*(*λx. A *v (x*::*real ^ -))
by (*simp add*: *linear-def matrix-vector-mult-def Cart-eq field-simps setsum-right-distrib setsum-addf*)

lemma *matrix-works*: **assumes** *lf*: *linear f* **shows** *matrix f *v x = f (x*::*real ^ 'n)*
apply (*simp add*: *matrix-def matrix-vector-mult-def Cart-eq mult-commute*)
apply *clarify*
apply (*rule linear-componentwise*[*OF lf*, *symmetric*])
done

lemma *matrix-vector-mul*: *linear f ==> f = (λx. matrix f *v (x*::*real ^ 'n))* **by** (*simp add*: *ext matrix-works*)

lemma *matrix-of-matrix-vector-mul*: *matrix*(*λx. A *v (x :: real ^ 'n)) = A*
by (*simp add*: *matrix-eq matrix-vector-mul-linear matrix-works*)

lemma *matrix-compose*:
assumes *lf*: *linear (f*::*real^'n ⇒ real^'m)*
and *lg*: *linear (g*::*real^'m ⇒ real^-)*
shows *matrix (g o f) = matrix g ** matrix f*
using *lf lg linear-compose*[*OF lf lg*] *matrix-works*[*OF linear-compose*[*OF lf lg*]]

by (*simp add*: *matrix-eq matrix-works matrix-vector-mul-assoc*[*symmetric*] *o-def*)

lemma *matrix-vector-column*:(*A*::'*a*::*comm-semiring-1*^'*n*^-) ∗*v x* = *setsum* (λ*i*. (*x*$*i*) ∗*s* ((*transpose A*)$*i*)) (*UNIV*:: '*n set*)
by (*simp add*: *matrix-vector-mult-def transpose-def Cart-eq mult-commute*)

lemma *adjoint-matrix*: *adjoint*(λ*x*. (*A*::*real*^'*n*^'*m*) ∗*v x*) = (λ*x*. *transpose A* ∗*v x*)
apply (*rule adjoint-unique*)
apply (*simp add*: *transpose-def inner-vector-def matrix-vector-mult-def setsum-left-distrib setsum-right-distrib*)
apply (*subst setsum-commute*)
apply (*auto simp add*: *mult-ac*)
done

lemma *matrix-adjoint*: **assumes** *lf*: *linear* (*f* :: *real*^'*n* ⇒ *real* ^'*m*)
shows *matrix*(*adjoint f*) = *transpose*(*matrix f*)
apply (*subst matrix-vector-mul*[*OF lf*])
unfolding *adjoint-matrix matrix-of-matrix-vector-mul* **..**

13.12 Interlude: Some properties of real sets

lemma *seq-mono-lemma*: **assumes** ∀(*n*::*nat*) ≥ *m*. (*d n* :: *real*) < *e n* **and** ∀ *n* ≥ *m*. *e n* <= *e m*
shows ∀ *n* ≥ *m*. *d n* < *e m*
using *prems* **apply** *auto*
apply (*erule-tac x*=*n* **in** *allE*)
apply (*erule-tac x*=*n* **in** *allE*)
apply *auto*
done

lemma *infinite-enumerate*: **assumes** *fS*: *infinite S*
shows ∃ *r*. *subseq r* ∧ (∀ *n*. *r n* ∈ *S*)
unfolding *subseq-def*
using *enumerate-in-set*[*OF fS*] *enumerate-mono*[*of* - - *S*] *fS* **by** *auto*

lemma *approachable-lt-le*: (∃(*d*::*real*)>*0*. ∀ *x*. *f x* < *d* ⟶ *P x*) ⟷ (∃ *d*>*0*. ∀ *x*. *f x* ≤ *d* ⟶ *P x*)
apply *auto*
apply (*rule-tac x*=*d*/*2* **in** *exI*)
apply *auto*
done

lemma *triangle-lemma*:
assumes *x*: *0* <= (*x*::*real*) **and** *y*:*0* <= *y* **and** *z*: *0* <= *z* **and** *xy*: *x*^*2* <= *y*^*2* + *z*^*2*
shows *x* <= *y* + *z*

proof−
have $y\hat{}2 + z\hat{}2 \leq y\hat{}2 + 2{*}y{*}z + z\hat{}2$ **using** z y **by** (*simp add*: *mult-nonneg-nonneg*)
with *xy* **have** *th*: $x\ \hat{}2 \leq (y{+}z)\hat{}2$ **by** (*simp add*: *power2-eq-square field-simps*)
from y z **have** *yz*: $y + z \geq 0$ **by** *arith*
from *power2-le-imp-le*[*OF th yz*] **show** *?thesis* **.**
qed

lemma *lambda-skolem*: $(\forall\, i.\ \exists\, x.\ P\ i\ x) \longleftrightarrow$
$(\exists\, x{::}'a\ \hat{}\ 'n.\ \forall\, i.\ P\ i\ (x\$i))$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof−
let *?S* = (*UNIV* :: $'n$ *set*)
{assume *H*: *?rhs*
then have *?lhs* **by** *auto***}**
moreover
{assume *H*: *?lhs*
then obtain *f* **where** *f*:$\forall\, i.\ P\ i\ (f\ i)$ **unfolding** *choice-iff* **by** *metis*
let *?x* = $(\chi\ i.\ (f\ i))$ :: $'a\ \hat{}\ 'n$
{fix *i*
from *f* **have** $P\ i\ (f\ i)$ **by** *metis*
then have $P\ i$ (*?x*\$*i*) **by** *auto*
}
hence $\forall\, i.\ P\ i$ (*?x*\$*i*) **by** *metis*
hence *?rhs* **by** *metis* **}**
ultimately show *?thesis* **by** *metis*
qed

lemma *vec-in-image-vec*: *vec* $x \in (vec\ `\ S) \longleftrightarrow x \in S$ **by** *auto*

lemma *vec-add*: $vec(x + y) = vec\ x + vec\ y$ **by** (*vector vec-def*)
lemma *vec-sub*: $vec(x - y) = vec\ x - vec\ y$ **by** (*vector vec-def*)
lemma *vec-cmul*: $vec(c{*}\ x) = c\ {*}s\ vec\ x$ **by** (*vector vec-def*)
lemma *vec-neg*: $vec(-\ x) = -\ vec\ x$ **by** (*vector vec-def*)

lemma *vec-setsum*: **assumes** *fS*: *finite S*
shows $vec(setsum\ f\ S) = setsum\ (vec\ o\ f)\ S$
apply (*induct rule*: *finite-induct*[*OF fS*])
apply (*simp*)
apply (*auto simp add*: *vec-add*)
done

lemma *setsum-Plus*:
⟦*finite A*; *finite B*⟧ ⟹
$(\sum x{\in}A\ {<+>}\ B.\ g\ x) = (\sum x{\in}A.\ g\ (Inl\ x)) + (\sum x{\in}B.\ g\ (Inr\ x))$
unfolding *Plus-def*
by (*subst setsum-Un-disjoint*, *auto simp add*: *setsum-reindex*)

lemma *setsum-UNIV-sum*:
fixes g :: $'a{::}finite + 'b{::}finite \Rightarrow$ -

shows $(\sum x{\in}UNIV.\ g\ x) = (\sum x{\in}UNIV.\ g\ (Inl\ x)) + (\sum x{\in}UNIV.\ g\ (Inr\ x))$
apply (*subst UNIV-Plus-UNIV* [*symmetric*])
apply (*rule setsum-Plus* [*OF finite finite*])
done

TODO: move to NthRoot

lemma *sqrt-add-le-add-sqrt*:
assumes x: $0 \leq x$ **and** y: $0 \leq y$
shows $sqrt\ (x + y) \leq sqrt\ x + sqrt\ y$
apply (*rule power2-le-imp-le*)
apply (*simp add*: *real-sum-squared-expand add-nonneg-nonneg x y*)
apply (*simp add*: *mult-nonneg-nonneg x y*)
apply (*simp add*: *add-nonneg-nonneg x y*)
done

13.13 A generic notion of "hull" (convex, affine, conic hull and closure).

definition *hull* :: *'a set set* ⇒ *'a set* ⇒ *'a set* (**infixl** *hull 75*) **where**
$S\ hull\ s = Inter\ \{t.\ t \in S \wedge s \subseteq t\}$

lemma *hull-same*: $s \in S \Longrightarrow S\ hull\ s = s$
unfolding *hull-def* **by** *auto*

lemma *hull-in*: $(\bigwedge T.\ T \subseteq S ==> Inter\ T \in S) ==> (S\ hull\ s) \in S$
unfolding *hull-def subset-iff* **by** *auto*

lemma *hull-eq*: $(\bigwedge T.\ T \subseteq S ==> Inter\ T \in S) ==> (S\ hull\ s) = s \longleftrightarrow s \in S$
using *hull-same*[*of s S*] *hull-in*[*of S s*] **by** *metis*

lemma *hull-hull*: $S\ hull\ (S\ hull\ s) = S\ hull\ s$
unfolding *hull-def* **by** *blast*

lemma *hull-subset*[*intro*]: $s \subseteq (S\ hull\ s)$
unfolding *hull-def* **by** *blast*

lemma *hull-mono*: $s \subseteq t ==> (S\ hull\ s) \subseteq (S\ hull\ t)$
unfolding *hull-def* **by** *blast*

lemma *hull-antimono*: $S \subseteq T ==> (T\ hull\ s) \subseteq (S\ hull\ s)$
unfolding *hull-def* **by** *blast*

lemma *hull-minimal*: $s \subseteq t \Longrightarrow t \in S ==> (S\ hull\ s) \subseteq t$
unfolding *hull-def* **by** *blast*

lemma *subset-hull*: $t \in S ==> S\ hull\ s \subseteq t \longleftrightarrow s \subseteq t$
unfolding *hull-def* **by** *blast*

lemma *hull-unique*: $s \subseteq t \Longrightarrow t \in S \Longrightarrow (\bigwedge t'.\ s \subseteq t' \Longrightarrow t' \in S \Longrightarrow t \subseteq t') \Longrightarrow (S\ hull\ s = t)$
unfolding *hull-def* **by** *auto*

lemma *hull-induct*: $(\bigwedge x.\ x \in S \Longrightarrow P\ x) \Longrightarrow Q\ \{x.\ P\ x\} \Longrightarrow \forall x \in Q\ hull\ S.\ P\ x$
using *hull-minimal*[*of S* $\{x.\ P\ x\}$ *Q*]
by (*auto simp add*: *subset-eq Collect-def mem-def*)

lemma *hull-inc*: $x \in S \Longrightarrow x \in P\ hull\ S$ **by** (*metis hull-subset subset-eq*)

lemma *hull-union-subset*: $(S\ hull\ s) \cup (S\ hull\ t) \subseteq (S\ hull\ (s \cup t))$
unfolding *Un-subset-iff* **by** (*metis hull-mono Un-upper1 Un-upper2*)

lemma *hull-union*: **assumes** *T*: $\bigwedge T.\ T \subseteq S \Longrightarrow Inter\ T \in S$
shows $S\ hull\ (s \cup t) = S\ hull\ (S\ hull\ s \cup S\ hull\ t)$
apply *rule*
apply (*rule hull-mono*)
unfolding *Un-subset-iff*
apply (*metis hull-subset Un-upper1 Un-upper2 subset-trans*)
apply (*rule hull-minimal*)
apply (*metis hull-union-subset*)
apply (*metis hull-in T*)
done

lemma *hull-redundant-eq*: $a \in (S\ hull\ s) \longleftrightarrow (S\ hull\ (insert\ a\ s) = S\ hull\ s)$
unfolding *hull-def* **by** *blast*

lemma *hull-redundant*: $a \in (S\ hull\ s) \Longrightarrow (S\ hull\ (insert\ a\ s) = S\ hull\ s)$
by (*metis hull-redundant-eq*)

Archimedian properties and useful consequences.

lemma *real-arch-simple*: $\exists n.\ x <= real\ (n::nat)$
using *reals-Archimedean2*[*of x*] **apply** *auto* **by** (*rule-tac x=Suc n* **in** *exI*, *auto*)
lemmas *real-arch-lt* = *reals-Archimedean2*

lemmas *real-arch* = *reals-Archimedean3*

lemma *real-arch-inv*: $0 < e \longleftrightarrow (\exists n::nat.\ n \neq 0 \wedge 0 < inverse\ (real\ n) \wedge inverse\ (real\ n) < e)$
using *reals-Archimedean*
apply (*auto simp add*: *field-simps*)
apply (*subgoal-tac inverse* (*real n*) > *0*)
apply *arith*
apply *simp*
done

lemma *real-pow-lbound*: $0 <= x \Longrightarrow 1 + real\ n * x <= (1 + x)\ \hat{}\ n$
proof(*induct n*)
case *0* **thus** *?case* **by** *simp*

next
case (*Suc n*)
hence *h*: *1 + real n * x ≤ (1 + x) ^ n* **by** *simp*
from *h* **have** *p*: *1 ≤ (1 + x) ^ n* **using** *Suc.prems* **by** *simp*
from *h* **have** *1 + real n * x + x ≤ (1 + x) ^ n + x* **by** *simp*
also have *... ≤ (1 + x) ^ Suc n* **apply** (*subst diff-le-0-iff-le*[*symmetric*])
apply (*simp add*: *field-simps*)
using *mult-left-mono*[*OF p Suc.prems*] **by** *simp*
finally show *?case* **by** (*simp add*: *real-of-nat-Suc field-simps*)
qed

lemma *real-arch-pow*: **assumes** *x*: *1 < (x*::*real*) **shows** ∃ *n. y < x^n*
proof−
from *x* **have** *x0*: *x − 1 > 0* **by** *arith*
from *real-arch*[*OF x0*, *rule-format*, *of y*]
obtain *n*::*nat* **where** *n*:*y < real n * (x − 1)* **by** *metis*
from *x0* **have** *x00*: *x− 1 ≥ 0* **by** *arith*
from *real-pow-lbound*[*OF x00*, *of n*] *n*
have *y < x^n* **by** *auto*
then show *?thesis* **by** *metis*
qed

lemma *real-arch-pow2*: ∃ *n.* (*x*::*real*) *< 2^ n*
using *real-arch-pow*[*of 2 x*] **by** *simp*

lemma *real-arch-pow-inv*: **assumes** *y*: (*y*::*real*) *> 0* **and** *x1*: *x < 1*
shows ∃ *n. x^n < y*
proof−
{assume *x0*: *x > 0*
from *x0 x1* **have** *ix*: *1 < 1/x* **by** (*simp add*: *field-simps*)
from *real-arch-pow*[*OF ix*, *of 1/y*]
obtain *n* **where** *n*: *1/y < (1/x) ^n* **by** *blast*
then
have *?thesis* **using** *y x0* **by** (*auto simp add*: *field-simps power-divide*) **}**
moreover
{assume ¬ *x > 0* **with** *y x1* **have** *?thesis* **apply** *auto* **by** (*rule exI*[**where** *x=1*], *auto*)**}**
ultimately show *?thesis* **by** *metis*
qed

lemma *forall-pos-mono*: (⋀*d e*::*real. d < e* ⟹ *P d* ==> *P e*) ⟹ (⋀*n*::*nat. n* ≠ *0* ==> *P*(*inverse*(*real n*))) ⟹ (⋀*e. 0 < e* ==> *P e*)
by (*metis real-arch-inv*)

lemma *forall-pos-mono-1*: (⋀*d e*::*real. d < e* ⟹ *P d* ==> *P e*) ⟹ (⋀*n.* *P*(*inverse*(*real* (*Suc n*)))) ==> *0 < e* ==> *P e*
apply (*rule forall-pos-mono*)
apply *auto*
apply (*atomize*)

apply (*erule-tac x=n − 1* **in** *allE*)
apply *auto*
done

lemma *real-archimedian-rdiv-eq-0*: **assumes** *x0*: $x \geq 0$ **and** *c*: $c \geq 0$ **and** *xc*: $\forall (m::nat)>0.\ real\ m * x \leq c$
shows $x = 0$
proof−
{assume $x \neq 0$ **with** *x0* **have** *xp*: $x > 0$ **by** *arith*
from *real-arch*[*OF xp*, *rule-format*, *of c*] **obtain** *n*::*nat* **where** *n*: $c < real\ n * x$ **by** *blast*
with *xc*[*rule-format*, *of n*] **have** $n = 0$ **by** *arith*
with *n c* **have** *False* **by** *simp*}
then show *?thesis* **by** *blast*
qed

13.14 Geometric progression

lemma *sum-gp-basic*: $((1::'a::\{field\}) - x) * setsum\ (\lambda i.\ x\hat{}i)\ \{0\ ..\ n\} = (1 - x\hat{}(Suc\ n))$
(**is** *?lhs* = *?rhs*)
proof−
{assume *x1*: $x = 1$ **hence** *?thesis* **by** *simp*}
moreover
{assume *x1*: $x \neq 1$
hence *x1'*: $x - 1 \neq 0\ 1 - x \neq 0\ x - 1 = -(1 - x) - (1 - x) \neq 0$ **by** *auto*
from *geometric-sum*[*OF x1*, *of Suc n*, *unfolded x1'*]
have $(-(1 - x)) * setsum\ (\lambda i.\ x\hat{}i)\ \{0\ ..\ n\} = -(1 - x\hat{}(Suc\ n))$
unfolding *atLeastLessThanSuc-atLeastAtMost*
using *x1'* **apply** (*auto simp only*: *field-simps*)
apply (*simp add*: *field-simps*)
done
then have *?thesis* **by** (*simp add*: *field-simps*) **}**
ultimately show *?thesis* **by** *metis*
qed

lemma *sum-gp-multiplied*: **assumes** *mn*: $m <= n$
shows $((1::'a::\{field\}) - x) * setsum\ (op\ \hat{}\ x)\ \{m..n\} = x\hat{}m - x\hat{}\ Suc\ n$
(**is** *?lhs* = *?rhs*)
proof−
let *?S* = $\{0..(n - m)\}$
from *mn* **have** *mn'*: $n - m \geq 0$ **by** *arith*
let *?f* = *op* + *m*
have *i*: *inj-on ?f ?S* **unfolding** *inj-on-def* **by** *auto*
have *f*: *?f* ‘ *?S* = $\{m..n\}$
using *mn* **apply** (*auto simp add*: *image-iff Bex-def*) **by** *arith*
have *th*: *op* ˆ *x o op* + *m* = $(\lambda i.\ x\hat{}m * x\hat{}i)$
by (*rule ext*, *simp add*: *power-add power-mult*)
from *setsum-reindex*[*OF i*, *of op* ˆ *x*, *unfolded f th setsum-right-distrib*[*symmetric*]]

have *?lhs* = $x\hat{}m * ((1 - x) * setsum\ (op\ \hat{}\ x)\ \{0..n - m\})$ **by** *simp*
then show *?thesis* **unfolding** *sum-gp-basic* **using** *mn*
by (*simp add*: *field-simps power-add*[*symmetric*])
qed

lemma *sum-gp*: $setsum\ (op\ \hat{}\ (x::'a::\{field\}))\ \{m\ ..\ n\} =$
$(if\ n < m\ then\ 0\ else\ if\ x = 1\ then\ of\text{-}nat\ ((n + 1) - m)$
$else\ (x\hat{}\ m - x\hat{}\ (Suc\ n))\ /\ (1 - x))$
proof−
{assume *nm*: $n < m$ **hence** *?thesis* **by** *simp***}**
moreover
{assume $\neg\ n < m$ **hence** *nm*: $m \leq n$ **by** *arith*
{assume *x*: $x = 1$ **hence** *?thesis* **by** *simp***}**
moreover
{assume *x*: $x \neq 1$ **hence** *nz*: $1 - x \neq 0$ **by** *simp*
from *sum-gp-multiplied*[*OF nm*, *of x*] *nz* **have** *?thesis* **by** (*simp add*: *field-simps*)**}**
ultimately have *?thesis* **by** *metis*
}
ultimately show *?thesis* **by** *metis*
qed

lemma *sum-gp-offset*: $setsum\ (op\ \hat{}\ (x::'a::\{field\}))\ \{m\ ..\ m+n\} =$
$(if\ x = 1\ then\ of\text{-}nat\ n + 1\ else\ x\hat{}m * (1 - x\hat{}Suc\ n)\ /\ (1 - x))$
unfolding *sum-gp*[*of x m m + n*] *power-Suc*
by (*simp add*: *field-simps power-add*)

13.15 A bit of linear algebra.

definition
$subspace :: 'a::real\text{-}vector\ set \Rightarrow bool$ **where**
$subspace\ S \longleftrightarrow 0 \in S \wedge (\forall x \in S.\ \forall y \in S.\ x + y \in S) \wedge (\forall c.\ \forall x \in S.\ c *_R x \in S)$

definition $span\ S = (subspace\ hull\ S)$
definition $dependent\ S \longleftrightarrow (\exists a \in S.\ a \in span(S - \{a\}))$
abbreviation $independent\ s == \sim(dependent\ s)$

Closure properties of subspaces.

lemma *subspace-UNIV*[*simp*]: $subspace(UNIV)$ **by** (*simp add*: *subspace-def*)

lemma *subspace-0*: $subspace\ S ==> 0 \in S$ **by** (*metis subspace-def*)

lemma *subspace-add*: $subspace\ S \Longrightarrow x \in S \Longrightarrow y \in S ==> x + y \in S$
by (*metis subspace-def*)

lemma *subspace-mul*: $subspace\ S \Longrightarrow x \in S \Longrightarrow c *_R x \in S$
by (*metis subspace-def*)

lemma *subspace-neg*: *subspace* $S \Longrightarrow x \in S \Longrightarrow - x \in S$
by (*metis scaleR-minus1-left subspace-mul*)

lemma *subspace-sub*: *subspace* $S \Longrightarrow x \in S \Longrightarrow y \in S \Longrightarrow x - y \in S$
by (*metis diff-def subspace-add subspace-neg*)

lemma *subspace-setsum*:
assumes *sA*: *subspace A* **and** *fB*: *finite B*
and *f*: $\forall x \in B.\ f\ x \in A$
shows *setsum f B* $\in A$
using *fB f sA*
apply(*induct rule*: *finite-induct*[*OF fB*])
by (*simp add*: *subspace-def sA*, *auto simp add*: *sA subspace-add*)

lemma *subspace-linear-image*:
assumes *lf*: *linear f* **and** *sS*: *subspace S*
shows *subspace*(*f ' S*)
using *lf sS linear-0*[*OF lf*]
unfolding *linear-def subspace-def*
apply (*auto simp add*: *image-iff*)
apply (*rule-tac x*=*x* + *y* **in** *bexI*, *auto*)
apply (*rule-tac x*=*c* $*_R$ *x* **in** *bexI*, *auto*)
done

lemma *subspace-linear-preimage*: *linear f* ==> *subspace S* ==> *subspace* {*x*. *f x* $\in$ *S*}
by (*auto simp add*: *subspace-def linear-def linear-0*[*of f*])

lemma *subspace-trivial*: *subspace* {*0*}
by (*simp add*: *subspace-def*)

lemma *subspace-inter*: *subspace A* $\Longrightarrow$ *subspace B* ==> *subspace* $(A \cap B)$
by (*simp add*: *subspace-def*)

lemma *span-mono*: $A \subseteq B$ ==> *span A* $\subseteq$ *span B*
by (*metis span-def hull-mono*)

lemma *subspace-span*: *subspace*(*span S*)
unfolding *span-def*
apply (*rule hull-in*[*unfolded mem-def*])
apply (*simp only*: *subspace-def Inter-iff Int-iff subset-eq*)
apply *auto*
apply (*erule-tac x*=*X* **in** *ballE*)
apply (*simp add*: *mem-def*)
apply *blast*
apply (*erule-tac x*=*X* **in** *ballE*)
apply (*erule-tac x*=*X* **in** *ballE*)
apply (*erule-tac x*=*X* **in** *ballE*)

apply (*clarsimp simp add*: *mem-def*)
apply *simp*
apply *simp*
apply *simp*
apply (*erule-tac x*=*X* **in** *ballE*)
apply (*erule-tac x*=*X* **in** *ballE*)
apply (*simp add*: *mem-def*)
apply *simp*
apply *simp*
done

lemma *span-clauses*:
$a \in S$ ==> $a \in$ *span* S
$0 \in$ *span* S
$x \in$ *span* $S \Longrightarrow y \in$ *span* S ==> $x + y \in$ *span* S
$x \in$ *span* $S \Longrightarrow c *_R x \in$ *span* S
by (*metis span-def hull-subset subset-eq*)
(*metis subspace-span subspace-def*)+

lemma *span-induct*: **assumes** *SP*: $\bigwedge x.\ x \in S$ ==> $P\ x$
and P: *subspace* P **and** x: $x \in$ *span* S **shows** $P\ x$
proof−
from *SP* **have** *SP'*: $S \subseteq P$ **by** (*simp add*: *mem-def subset-eq*)
from P **have** P': $P \in$ *subspace* **by** (*simp add*: *mem-def*)
from x *hull-minimal*[*OF SP' P'*, *unfolded span-def*[*symmetric*]]
show $P\ x$ **by** (*metis mem-def subset-eq*)
qed

lemma *span-empty*: *span* {} = {*0*}
apply (*simp add*: *span-def*)
apply (*rule hull-unique*)
apply (*auto simp add*: *mem-def subspace-def*)
unfolding *mem-def*[*of* 0::'*a*, *symmetric*]
apply *simp*
done

lemma *independent-empty*: *independent* {}
by (*simp add*: *dependent-def*)

lemma *independent-mono*: *independent* $A \Longrightarrow B \subseteq A$ ==> *independent* B
apply (*clarsimp simp add*: *dependent-def span-mono*)
apply (*subgoal-tac span* $(B - \{a\}) \leq$ *span* $(A - \{a\})$)
apply *force*
apply (*rule span-mono*)
apply *auto*
done

lemma *span-subspace*: $A \subseteq B \Longrightarrow B \leq$ *span* $A \Longrightarrow$ *subspace* $B \Longrightarrow$ *span* $A = B$
by (*metis order-antisym span-def hull-minimal mem-def*)

lemma *span-induct'*: **assumes** *SP*: $\forall x \in S.\ P\ x$
and *P*: *subspace P* **shows** $\forall x \in span\ S.\ P\ x$
using *span-induct SP P* **by** *blast*

inductive *span-induct-alt-help* **for** *S*:: *'a*::*real-vector* ⇒ *bool*
where
span-induct-alt-help-0: *span-induct-alt-help S 0*
| *span-induct-alt-help-S*: $x \in S \Longrightarrow$ *span-induct-alt-help S z* $\Longrightarrow$ *span-induct-alt-help S* $(c *_R x + z)$

lemma *span-induct-alt'*:
assumes *h0*: *h 0* **and** *hS*: $\bigwedge c\ x\ y.\ x \in S \Longrightarrow h\ y \Longrightarrow h\ (c *_R x + y)$ **shows** $\forall x \in span\ S.\ h\ x$
proof−
{fix *x*:: *'a* **assume** *x*: *span-induct-alt-help S x*
have *h x*
apply (*rule span-induct-alt-help.induct*[*OF x*])
apply (*rule h0*)
apply (*rule hS*, *assumption*, *assumption*)
done}
note *th0* = *this*
{fix *x* **assume** *x*: $x \in span\ S$

have *span-induct-alt-help S x*
proof(*rule span-induct*[**where** *x*=*x* **and** *S*=*S*])
show $x \in span\ S$ **using** *x* **.**
next
fix *x* **assume** *xS* : $x \in S$
from *span-induct-alt-help-S*[*OF xS span-induct-alt-help-0*, *of 1*]
show *span-induct-alt-help S x* **by** *simp*
next
have *span-induct-alt-help S 0* **by** (*rule span-induct-alt-help-0*)
moreover
{fix *x y* **assume** *h*: *span-induct-alt-help S x span-induct-alt-help S y*
from *h*
have *span-induct-alt-help S* $(x + y)$
apply (*induct rule*: *span-induct-alt-help.induct*)
apply *simp*
unfolding *add-assoc*
apply (*rule span-induct-alt-help-S*)
apply *assumption*
apply *simp*
done}
moreover
{fix *c x* **assume** *xt*: *span-induct-alt-help S x*
then have *span-induct-alt-help S* $(c *_R x)$
apply (*induct rule*: *span-induct-alt-help.induct*)
apply (*simp add*: *span-induct-alt-help-0*)

```
        apply (simp add: scaleR-right-distrib)
        apply (rule span-induct-alt-help-S)
        apply assumption
        apply simp
        done
    }
    ultimately show subspace (span-induct-alt-help S)
      unfolding subspace-def mem-def Ball-def by blast
  qed}
 with th0 show ?thesis by blast
qed

lemma span-induct-alt:
 assumes h0: h 0 and hS: ⋀c x y. x ∈ S ⟹ h y ⟹ h (c *R x + y) and x: x
∈ span S
 shows h x
using span-induct-alt'[of h S] h0 hS x by blast
```

Individual closure properties.

```
lemma span-superset: x ∈ S ==> x ∈ span S by (metis span-clauses(1))

lemma span-0: 0 ∈ span S by (metis subspace-span subspace-0)

lemma span-add: x ∈ span S ⟹ y ∈ span S ==> x + y ∈ span S
 by (metis subspace-add subspace-span)

lemma span-mul: x ∈ span S ==> (c *R x) ∈ span S
 by (metis subspace-span subspace-mul)

lemma span-neg: x ∈ span S ==> − x ∈ span S
 by (metis subspace-neg subspace-span)

lemma span-sub: x ∈ span S ⟹ y ∈ span S ==> x − y ∈ span S
 by (metis subspace-span subspace-sub)

lemma span-setsum: finite A ⟹ ∀ x ∈ A. f x ∈ span S ==> setsum f A ∈ span
S
 by (rule subspace-setsum, rule subspace-span)

lemma span-add-eq: x ∈ span S ⟹ x + y ∈ span S ⟷ y ∈ span S
 apply (auto simp only: span-add span-sub)
 apply (subgoal-tac (x + y) − x ∈ span S, simp)
 by (simp only: span-add span-sub)
```

Mapping under linear image.

```
lemma span-linear-image: assumes lf: linear f
 shows span (f ‘ S) = f ‘ (span S)
proof−
 {fix x
```

```
    assume x: x ∈ span (f ‘ S)
    have x ∈ f ‘ span S
      apply (rule span-induct[where x=x and S = f ‘ S])
      apply (clarsimp simp add: image-iff)
      apply (frule span-superset)
      apply blast
      apply (simp only: mem-def)
      apply (rule subspace-linear-image[OF lf])
      apply (rule subspace-span)
      apply (rule x)
      done}
  moreover
  {fix x assume x: x ∈ span S
   have th0:(λa. f a ∈ span (f ‘ S)) = {x. f x ∈ span (f ‘ S)} apply (rule set-ext)
     unfolding mem-def Collect-def ..
    have f x ∈ span (f ‘ S)
      apply (rule span-induct[where S=S])
      apply (rule span-superset)
      apply simp
      apply (subst th0)
      apply (rule subspace-linear-preimage[OF lf subspace-span, of f ‘ S])
      apply (rule x)
      done}
  ultimately show ?thesis by blast
qed
```

The key breakdown property.

```
lemma span-breakdown:
  assumes bS: b ∈ S and aS: a ∈ span S
  shows ∃k. a − k *R b ∈ span (S − {b}) (is ?P a)
proof−
  {fix x assume xS: x ∈ S
    {assume ab: x = b
     then have ?P x
       apply simp
       apply (rule exI[where x=1], simp)
       by (rule span-0)}
    moreover
    {assume ab: x ≠ b
     then have ?P x  using xS
       apply −
       apply (rule exI[where x=0])
       apply (rule span-superset)
       by simp}
    ultimately have ?P x by blast}
  moreover have subspace ?P
    unfolding subspace-def
    apply auto
    apply (simp add: mem-def)
```

apply (*rule exI*[**where** *x=0*])
using *span-0*[*of* $S - \{b\}$]
apply (*simp add*: *mem-def*)
apply (*clarsimp simp add*: *mem-def*)
apply (*rule-tac x=k + ka* **in** *exI*)
apply (*subgoal-tac* $x + y - (k + ka) *_R b = (x - k*_R b) + (y - ka *_R b)$)
apply (*simp only*:)
apply (*rule span-add*[*unfolded mem-def*])
apply *assumption*+
apply (*simp add*: *algebra-simps*)
apply (*clarsimp simp add*: *mem-def*)
apply (*rule-tac x= c*k* **in** *exI*)
apply (*subgoal-tac* $c *_R x - (c * k) *_R b = c*_R (x - k*_R b)$)
apply (*simp only*:)
apply (*rule span-mul*[*unfolded mem-def*])
apply *assumption*
by (*simp add*: *algebra-simps*)
ultimately show *?P a* **using** *aS span-induct*[**where** *S=S* **and** *P= ?P*] **by** *metis*
qed

lemma *span-breakdown-eq*:
$x \in span\ (insert\ a\ S) \longleftrightarrow (\exists k.\ (x - k *_R a) \in span\ S)$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof−
{assume *x*: $x \in span\ (insert\ a\ S)$
from *x span-breakdown*[*of a insert a S x*]
have *?rhs* **apply** *clarsimp*
apply (*rule-tac x= k* **in** *exI*)
apply (*rule set-rev-mp*[*of* - *span* $(S - \{a\})$ -])
apply *assumption*
apply (*rule span-mono*)
apply *blast*
done}
moreover
{ fix *k* **assume** *k*: $x - k *_R a \in span\ S$
have *eq*: $x = (x - k *_R a) + k *_R a$ **by** *vector*
have $(x - k *_R a) + k *_R a \in span\ (insert\ a\ S)$
apply (*rule span-add*)
apply (*rule set-rev-mp*[*of* - *span S* -])
apply (*rule k*)
apply (*rule span-mono*)
apply *blast*
apply (*rule span-mul*)
apply (*rule span-superset*)
apply *blast*
done
then have *?lhs* **using** *eq* **by** *metis***}**
ultimately show *?thesis* **by** *blast*
qed

Hence some "reversal" results.

lemma *in-span-insert*:
assumes *a*: $a \in span\ (insert\ b\ S)$ **and** *na*: $a \notin span\ S$
shows $b \in span\ (insert\ a\ S)$
proof−
from *span-breakdown*[*of b insert b S a*, *OF insertI1 a*]
obtain *k* **where** *k*: $a - k *_R b \in span\ (S - \{b\})$ **by** *auto*
{assume *k0*: $k = 0$
with *k* **have** $a \in span\ S$
apply (*simp*)
apply (*rule set-rev-mp*)
apply *assumption*
apply (*rule span-mono*)
apply *blast*
done
with *na* **have** *?thesis* **by** *blast***}**
moreover
{assume *k0*: $k \neq 0$
have *eq*: $b = (1/k) *_R a - ((1/k) *_R a - b)$ **by** *vector*
from *k0* **have** *eq'*: $(1/k) *_R (a - k *_R b) = (1/k) *_R a - b$
by (*simp add*: *algebra-simps*)
from *k* **have** $(1/k) *_R (a - k *_R b) \in span\ (S - \{b\})$
by (*rule span-mul*)
hence *th*: $(1/k) *_R a - b \in span\ (S - \{b\})$
unfolding *eq'* **.**

from *k*
have *?thesis*
apply (*subst eq*)
apply (*rule span-sub*)
apply (*rule span-mul*)
apply (*rule span-superset*)
apply *blast*
apply (*rule set-rev-mp*)
apply (*rule th*)
apply (*rule span-mono*)
using *na* **by** *blast***}**
ultimately show *?thesis* **by** *blast*
qed

lemma *in-span-delete*:
assumes *a*: $a \in span\ S$
and *na*: $a \notin span\ (S - \{b\})$
shows $b \in span\ (insert\ a\ (S - \{b\}))$
apply (*rule in-span-insert*)
apply (*rule set-rev-mp*)
apply (*rule a*)
apply (*rule span-mono*)
apply *blast*

apply (*rule na*)
done

Transitivity property.

lemma *span-trans*:
assumes *x*: $x \in$ *span* S **and** *y*: $y \in$ *span* (*insert x S*)
shows $y \in$ *span* S
proof−
from *span-breakdown*[*of x insert x S y*, *OF insertI1 y*]
obtain *k* **where** *k*: $y - k *_R x \in$ *span* $(S - \{x\})$ **by** *auto*
have *eq*: $y = (y - k *_R x) + k *_R x$ **by** *vector*
show *?thesis*
apply (*subst eq*)
apply (*rule span-add*)
apply (*rule set-rev-mp*)
apply (*rule k*)
apply (*rule span-mono*)
apply *blast*
apply (*rule span-mul*)
by (*rule x*)
qed

An explicit expansion is sometimes needed.

lemma *span-explicit*:
span P = $\{y. \exists S\ u.$ *finite S* $\wedge$ $S \subseteq P \wedge$ *setsum* $(\lambda v.\ u\ v *_R v)\ S = y\}$
(**is** - = *?E* **is** - = $\{y.$ *?h y*$\}$ **is** - = $\{y. \exists S\ u.$ *?Q S u y*$\}$)
proof−
{fix *x* **assume** *x*: $x \in$ *?E*
then obtain *S u* **where** *fS*: *finite S* **and** *SP*: $S \subseteq P$ **and** *u*: *setsum* $(\lambda v.\ u\ v *_R v)\ S = x$
by *blast*
have $x \in$ *span P*
unfolding *u*[*symmetric*]
apply (*rule span-setsum*[*OF fS*])
using *span-mono*[*OF SP*]
by (*auto intro*: *span-superset span-mul*)**}**
moreover
have $\forall x \in$ *span P*. $x \in$ *?E*
unfolding *mem-def Collect-def*
proof(*rule span-induct-alt'*)
show *?h 0*
apply (*rule exI*[**where** *x*={}]) **by** *simp*
next
fix *c x y*
assume *x*: $x \in P$ **and** *hy*: *?h y*
from *hy* **obtain** *S u* **where** *fS*: *finite S* **and** *SP*: $S \subseteq P$
and *u*: *setsum* $(\lambda v.\ u\ v *_R v)\ S = y$ **by** *blast*
let *?S* = *insert x S*
let *?u* = $\lambda y.$ *if* $y = x$ *then* (*if* $x \in S$ *then u y* + *c else c*)

```
            else u y
   from fS SP x have th0: finite (insert x S) insert x S ⊆ P by blast+
   {assume xS: x ∈ S
    have S1: S = (S − {x}) ∪ {x}
      and Sss:finite (S − {x}) finite {x} (S −{x}) ∩ {x} = {} using xS fS by
auto
    have setsum (λv. ?u v *R v) ?S =(∑ v∈S − {x}. u v *R v) + (u x + c) *R
x
      using xS
      by (simp add: setsum-Un-disjoint[OF Sss, unfolded S1[symmetric]]
        setsum-clauses(2)[OF fS] cong del: if-weak-cong)
    also have ... = (∑ v∈S. u v *R v) + c *R x
      apply (simp add: setsum-Un-disjoint[OF Sss, unfolded S1[symmetric]])
      by (simp add: algebra-simps)
    also have ... = c*R x + y
      by (simp add: add-commute u)
    finally have setsum (λv. ?u v *R v) ?S = c*R x + y .
   then have ?Q ?S ?u (c*R x + y) using th0 by blast}
  moreover
  {assume xS: x ∉ S
   have th00: (∑ v∈S. (if v = x then c else u v) *R v) = y
    unfolding u[symmetric]
    apply (rule setsum-cong2)
    using xS by auto
   have ?Q ?S ?u (c*R x + y) using fS xS th0
    by (simp add: th00 setsum-clauses add-commute cong del: if-weak-cong)}
  ultimately have ?Q ?S ?u (c*R x + y)
   by (cases x ∈ S, simp, simp)
   then show ?h (c*R x + y)
    apply −
    apply (rule exI[where x=?S])
    apply (rule exI[where x=?u]) by metis
  qed
  ultimately show ?thesis by blast
qed

lemma dependent-explicit:
  dependent P ⟷ (∃ S u. finite S ∧ S ⊆ P ∧ (∃ v∈S. u v ≠ 0 ∧ setsum (λv. u
v *R v) S = 0)) (is ?lhs = ?rhs)
proof−
  {assume dP: dependent P
   then obtain a S u where aP: a ∈ P and fS: finite S
    and SP: S ⊆ P − {a} and ua: setsum (λv. u v *R v) S = a
    unfolding dependent-def span-explicit by blast
   let ?S = insert a S
   let ?u = λy. if y = a then − 1 else u y
   let ?v = a
   from aP SP have aS: a ∉ S by blast
   from fS SP aP have th0: finite ?S ?S ⊆ P ?v ∈ ?S ?u ?v ≠ 0 by auto
```

```
    have s0: setsum (λv. ?u v *R v) ?S = 0
      using fS aS
      apply (simp add: vector-smult-lneg setsum-clauses field-simps)
      apply (subst (2) ua[symmetric])
      apply (rule setsum-cong2)
      by auto
    with th0 have ?rhs
      apply −
      apply (rule exI[where x= ?S])
      apply (rule exI[where x= ?u])
      by clarsimp}
  moreover
  {fix S u v assume fS: finite S
     and SP: S ⊆ P and vS: v ∈ S and uv: u v ≠ 0
    and u: setsum (λv. u v *R v) S = 0
    let ?a = v
    let ?S = S − {v}
    let ?u = λi. (− u i) / u v
    have th0: ?a ∈ P finite ?S ?S ⊆ P       using fS SP vS by auto
    have setsum (λv. ?u v *R v) ?S = setsum (λv. (− (inverse (u ?a))) *R (u v
*R v)) S − ?u v *R v
      using fS vS uv
      by (simp add: setsum-diff1 vector-smult-lneg divide-inverse
        vector-smult-assoc field-simps)
    also have ... = ?a
      unfolding scaleR-right.setsum [symmetric] u
      using uv by simp
    finally  have setsum (λv. ?u v *R v) ?S = ?a .
    with th0 have ?lhs
      unfolding dependent-def span-explicit
      apply −
      apply (rule bexI[where x= ?a])
      apply (simp-all del: scaleR-minus-left)
      apply (rule exI[where x= ?S])
      by (auto simp del: scaleR-minus-left)}
  ultimately show ?thesis by blast
qed


lemma span-finite:
  assumes fS: finite S
  shows span S = {y. ∃ u. setsum (λv. u v *R v) S = y}
  (is - = ?rhs)
proof−
  {fix y assume y: y ∈ span S
    from y obtain S' u where fS': finite S' and SS': S' ⊆ S and
      u: setsum (λv. u v *R v) S' = y unfolding span-explicit by blast
    let ?u = λx. if x ∈ S' then u x else 0
    from setsum-restrict-set[OF fS, of λv. u v *R v S', symmetric] SS'
```

```
    have setsum (λv. ?u v *R v) S = setsum (λv. u v *R v) S'
      unfolding cond-value-iff cond-application-beta
      by (simp add: cond-value-iff inf-absorb2 cong del: if-weak-cong)
    hence setsum (λv. ?u v *R v) S = y by (metis u)
    hence y ∈ ?rhs by auto}
  moreover
  {fix y u assume u: setsum (λv. u v *R v) S = y
    then have y ∈ span S using fS unfolding span-explicit by auto}
  ultimately show ?thesis by blast
qed
```

Standard bases are a spanning set, and obviously finite.

```
lemma span-stdbasis:span {basis i :: real^'n | i. i ∈ (UNIV :: 'n set)} = UNIV
apply (rule set-ext)
apply auto
apply (subst basis-expansion'[symmetric])
apply (rule span-setsum)
apply simp
apply auto
apply (rule span-mul)
apply (rule span-superset)
apply (auto simp add: Collect-def mem-def)
done

lemma finite-stdbasis: finite {basis i ::real^'n |i. i∈ (UNIV:: 'n set)} (is finite
?S)
proof−
  have eq: ?S = basis ' UNIV by blast
  show ?thesis unfolding eq by auto
qed

lemma card-stdbasis: card {basis i ::real^'n |i. i∈ (UNIV :: 'n set)} = CARD('n)
(is card ?S = -)
proof−
  have eq: ?S = basis ' UNIV by blast
  show ?thesis unfolding eq using card-image[OF basis-inj] by simp
qed


lemma independent-stdbasis-lemma:
  assumes x: (x::real ^ 'n) ∈ span (basis ' S)
  and iS: i ∉ S
  shows (x$i) = 0
proof−
  let ?U = UNIV :: 'n set
  let ?B = basis ' S
  let ?P = λ(x::real^-). ∀ i∈ ?U. i ∉ S ⟶ x$i =0
 {fix x::real^- assume xS: x∈ ?B
   from xS have ?P x by auto}
```

```
 moreover
 have subspace ?P
   by (auto simp add: subspace-def Collect-def mem-def)
 ultimately show ?thesis
   using x span-induct[of ?B ?P x] iS by blast
qed

lemma independent-stdbasis: independent {basis i ::real^'n |i. i∈ (UNIV :: 'n
set)}
proof−
  let ?I = UNIV :: 'n set
  let ?b = basis :: - ⇒ real ^'n
  let ?B = ?b ‘ ?I
  have eq: {?b i|i. i ∈ ?I} = ?B
    by auto
  {assume d: dependent ?B
    then obtain k where k: k ∈ ?I ?b k ∈ span (?B − {?b k})
      unfolding dependent-def by auto
    have eq1: ?B − {?b k} = ?B − ?b ‘ {k}  by simp
    have eq2: ?B − {?b k} = ?b ‘ (?I − {k})
      unfolding eq1
      apply (rule inj-on-image-set-diff[symmetric])
      apply (rule basis-inj) using k(1) by auto
    from k(2) have th0: ?b k ∈ span (?b ‘ (?I − {k})) unfolding eq2 .
    from independent-stdbasis-lemma[OF th0, of k, simplified]
    have False by simp}
  then show ?thesis unfolding eq dependent-def ..
qed
```

This is useful for building a basis step-by-step.

```
lemma independent-insert:
  independent(insert a S) ⟷
      (if a ∈ S then independent S
              else independent S ∧ a ∉ span S) (is ?lhs ⟷ ?rhs)
proof−
  {assume aS: a ∈ S
    hence ?thesis using insert-absorb[OF aS] by simp}
  moreover
  {assume aS: a ∉ S
    {assume i: ?lhs
      then have ?rhs using aS
        apply simp
        apply (rule conjI)
        apply (rule independent-mono)
        apply assumption
        apply blast
        by (simp add: dependent-def)}
    moreover
    {assume i: ?rhs
```

```
    have ?lhs using i aS
      apply simp
      apply (auto simp add: dependent-def)
      apply (case-tac aa = a, auto)
      apply (subgoal-tac insert a S − {aa} = insert a (S − {aa}))
      apply simp
      apply (subgoal-tac a ∈ span (insert aa (S − {aa})))
      apply (subgoal-tac insert aa (S − {aa}) = S)
      apply simp
      apply blast
      apply (rule in-span-insert)
      apply assumption
      apply blast
      apply blast
      done}
   ultimately have ?thesis by blast}
  ultimately show ?thesis by blast
qed
```

The degenerate case of the Exchange Lemma.

```
lemma mem-delete: x ∈ (A − {a}) ⟷ x ≠ a ∧ x ∈ A
  by blast

lemma span-span: span (span A) = span A
  unfolding span-def hull-hull ..

lemma span-inc: S ⊆ span S
  by (metis subset-eq span-superset)

lemma spanning-subset-independent:
  assumes BA: B ⊆ A and iA: independent A
  and AsB: A ⊆ span B
  shows A = B
proof
  from BA show B ⊆ A .
next
  from span-mono[OF BA] span-mono[OF AsB]
  have sAB: span A = span B unfolding span-span by blast

  {fix x assume x: x ∈ A
    from iA have th0: x ∉ span (A − {x})
      unfolding dependent-def using x by blast
    from x have xsA: x ∈ span A by (blast intro: span-superset)
    have A − {x} ⊆ A by blast
    hence th1:span (A − {x}) ⊆ span A by (metis span-mono)
    {assume xB: x ∉ B
      from xB BA have B ⊆ A −{x} by blast
      hence span B ⊆ span (A − {x}) by (metis span-mono)
      with th1 th0 sAB have x ∉ span A by blast
```

```
    with x have False by (metis span-superset)}
   then have x ∈ B by blast}
  then show A ⊆ B by blast
qed
```

The general case of the Exchange Lemma, the key to what follows.

```
lemma exchange-lemma:
  assumes f:finite t and i: independent s
  and sp:s ⊆ span t
  shows ∃t'. (card t' = card t) ∧ finite t' ∧ s ⊆ t' ∧ t' ⊆ s ∪ t ∧ s ⊆ span t'
using f i sp
proof(induct card (t − s) arbitrary: s t rule: less-induct)
  case less
  note ft = ‹finite t› and s = ‹independent s› and sp = ‹s ⊆ span t›
  let ?P = λt'. (card t' = card t) ∧ finite t' ∧ s ⊆ t' ∧ t' ⊆ s ∪ t ∧ s ⊆ span t'
  let ?ths = ∃t'. ?P t'
  {assume st: s ⊆ t
   from st ft span-mono[OF st] have ?ths apply − apply (rule exI[where x=t])
     by (auto intro: span-superset)}
  moreover
  {assume st: t ⊆ s

   from spanning-subset-independent[OF st s sp]
     st ft span-mono[OF st] have ?ths apply − apply (rule exI[where x=t])
     by (auto intro: span-superset)}
  moreover
  {assume st: ¬ s ⊆ t ¬ t ⊆ s
   from st(2) obtain b where b: b ∈ t b ∉ s by blast
     from b have t − {b} − s ⊂ t − s by blast
     then have cardlt: card (t − {b} − s) < card (t − s) using ft
       by (auto intro: psubset-card-mono)
     from b ft have ct0: card t ≠ 0 by auto
   {assume stb: s ⊆ span(t −{b})
     from ft have ftb: finite (t −{b}) by auto
     from less(1)[OF cardlt ftb s stb]
      obtain u where u: card u = card (t−{b}) s ⊆ u u ⊆ s ∪ (t − {b}) s ⊆
span u and fu: finite u by blast
     let ?w = insert b u
     have th0: s ⊆ insert b u using u by blast
     from u(3) b have u ⊆ s ∪ t by blast
     then have th1: insert b u ⊆ s ∪ t using u b by blast
     have bu: b ∉ u using b u by blast
     from u(1) ft b have card u = (card t − 1) by auto
     then
     have th2: card (insert b u) = card t
       using card-insert-disjoint[OF fu bu] ct0 by auto
     from u(4) have s ⊆ span u .
     also have ... ⊆ span (insert b u) apply (rule span-mono) by blast
     finally have th3: s ⊆ span (insert b u) .
```

```
    from th0 th1 th2 th3 fu have th: ?P ?w  by blast
    from th have ?ths by blast}
  moreover
  {assume stb: ¬ s ⊆ span(t −{b})
    from stb obtain a where a: a ∈ s a ∉ span (t − {b}) by blast
    have ab: a ≠ b using a b by blast
    have at: a ∉ t using a ab span-superset[of a t− {b}] by auto
    have mlt: card ((insert a (t − {b})) − s) < card (t − s)
      using cardlt ft a b by auto
    have ft': finite (insert a (t − {b})) using ft by auto
    {fix x assume xs: x ∈ s
      have t: t ⊆ (insert b (insert a (t −{b}))) using b by auto
      from b(1) have b ∈ span t by (simp add: span-superset)
      have bs: b ∈ span (insert a (t − {b})) apply(rule in-span-delete)
        using  a sp unfolding subset-eq by auto
      from xs sp have x ∈ span t by blast
      with span-mono[OF t]
      have x: x ∈ span (insert b (insert a (t − {b}))) ..
      from span-trans[OF bs x] have x ∈ span (insert a (t − {b}))  .}
    then have sp': s ⊆ span (insert a (t − {b})) by blast

    from less(1)[OF mlt ft' s sp'] obtain u where
      u: card u = card (insert a (t −{b})) finite u s ⊆ u u ⊆ s ∪ insert a (t
−{b})
      s ⊆ span u by blast
    from u a b ft at ct0 have ?P u by auto
    then have ?ths by blast }
  ultimately have ?ths by blast
 }
 ultimately
 show ?ths  by blast
qed
```

This implies corresponding size bounds.

```
lemma independent-span-bound:
  assumes f: finite t and i: independent s and sp:s ⊆ span t
  shows finite s ∧ card s ≤ card t
  by (metis exchange-lemma[OF f i sp] finite-subset card-mono)


lemma finite-Atleast-Atmost-nat[simp]: finite {f x |x. x∈ (UNIV::'a::finite set)}
proof−
  have eq: {f x |x. x∈ UNIV} = f ' UNIV by auto
  show ?thesis unfolding eq
    apply (rule finite-imageI)
    apply (rule finite)
    done
qed
```

lemma *independent-bound*:
fixes *S*:: (*real^'n*) *set*
shows *independent S* ⟹ *finite S* ∧ *card S* <= *CARD*('*n*)
apply (*subst card-stdbasis*[*symmetric*])
apply (*rule independent-span-bound*)
apply (*rule finite-Atleast-Atmost-nat*)
apply *assumption*
unfolding *span-stdbasis*
apply (*rule subset-UNIV*)
done

lemma *dependent-biggerset*: (*finite* (*S*::(*real ^'n*) *set*) ==> *card S* > *CARD*('*n*))
==> *dependent S*
by (*metis independent-bound not-less*)

Hence we can create a maximal independent subset.

lemma *maximal-independent-subset-extend*:
assumes *sv*: (*S*::(*real^'n*) *set*) ⊆ *V* **and** *iS*: *independent S*
shows ∃ *B*. *S* ⊆ *B* ∧ *B* ⊆ *V* ∧ *independent B* ∧ *V* ⊆ *span B*
using *sv iS*
proof(*induct CARD*('*n*) − *card S arbitrary*: *S rule*: *less-induct*)
case *less*
note *sv* = ‹*S* ⊆ *V*› **and** *i* = ‹*independent S*›
let *?P* = λ*B*. *S* ⊆ *B* ∧ *B* ⊆ *V* ∧ *independent B* ∧ *V* ⊆ *span B*
let *?ths* = ∃ *x*. *?P x*
let *?d* = *CARD*('*n*)
{assume *V* ⊆ *span S*
then have *?ths* **using** *sv i* **by** *blast* **}**
moreover
{assume *VS*: ¬ *V* ⊆ *span S*
from *VS* **obtain** *a* **where** *a*: *a* ∈ *V a* ∉ *span S* **by** *blast*
from *a* **have** *aS*: *a* ∉ *S* **by** (*auto simp add*: *span-superset*)
have *th0*: *insert a S* ⊆ *V* **using** *a sv* **by** *blast*
from *independent-insert*[*of a S*] *i a*
have *th1*: *independent* (*insert a S*) **by** *auto*
have *mlt*: *?d* − *card* (*insert a S*) < *?d* − *card S*
using *aS a independent-bound*[*OF th1*]
by *auto*

from *less*(*1*)[*OF mlt th0 th1*]
obtain *B* **where** *B*: *insert a S* ⊆ *B B* ⊆ *V independent B V* ⊆ *span B*
by *blast*
from *B* **have** *?P B* **by** *auto*
then have *?ths* **by** *blast***}**
ultimately show *?ths* **by** *blast*
qed

lemma *maximal-independent-subset*:

$\exists (B::(real\ \hat{}\ 'n)\ set).\ B \subseteq V \wedge independent\ B \wedge V \subseteq span\ B$
by (*metis maximal-independent-subset-extend*[*of* {}::(*real* ^ *'n*) *set*] *empty-subsetI independent-empty*)

Notion of dimension.

definition $dim\ V = (SOME\ n.\ \exists B.\ B \subseteq V \wedge independent\ B \wedge V \subseteq span\ B \wedge (card\ B = n))$

lemma *basis-exists*: $\exists B.\ (B::(real\ \hat{}\ 'n)\ set) \subseteq V \wedge independent\ B \wedge V \subseteq span\ B \wedge (card\ B = dim\ V)$
unfolding *dim-def some-eq-ex*[*of* $\lambda n.\ \exists B.\ B \subseteq V \wedge independent\ B \wedge V \subseteq span\ B \wedge (card\ B = n)$]
using *maximal-independent-subset*[*of V*] *independent-bound*
by *auto*

Consequences of independence or spanning for cardinality.

lemma *independent-card-le-dim*:
assumes $(B::(real\ \hat{}\ 'n)\ set) \subseteq V$ **and** *independent B* **shows** $card\ B \leq dim\ V$
proof −
from *basis-exists*[*of V*] ‹$B \subseteq V$›
obtain B' **where** *independent* B' **and** $B \subseteq span\ B'$ **and** $card\ B' = dim\ V$ **by** *blast*
with *independent-span-bound*[*OF* - ‹*independent B*› ‹$B \subseteq span\ B'$›] *independent-bound*[*of* B']
show *?thesis* **by** *auto*
qed

lemma *span-card-ge-dim*: $(B::(real\ \hat{}\ 'n)\ set) \subseteq V \Longrightarrow V \subseteq span\ B \Longrightarrow finite\ B \Longrightarrow dim\ V \leq card\ B$
by (*metis basis-exists*[*of V*] *independent-span-bound subset-trans*)

lemma *basis-card-eq-dim*:
$B \subseteq (V::(real\ \hat{}\ 'n)\ set) \Longrightarrow V \subseteq span\ B \Longrightarrow independent\ B \Longrightarrow finite\ B \wedge card\ B = dim\ V$
by (*metis order-eq-iff independent-card-le-dim span-card-ge-dim independent-bound*)

lemma *dim-unique*: $(B::(real\ \hat{}\ 'n)\ set) \subseteq V \Longrightarrow V \subseteq span\ B \Longrightarrow independent\ B \Longrightarrow card\ B = n \Longrightarrow dim\ V = n$
by (*metis basis-card-eq-dim*)

More lemmas about dimension.

lemma *dim-univ*: $dim\ (UNIV::(real\hat{}\ 'n)\ set) = CARD('n)$
apply (*rule dim-unique*[*of* {*basis i* |*i*. *i*∈ (*UNIV* :: *'n set*)}])
by (*auto simp only*: *span-stdbasis card-stdbasis finite-stdbasis independent-stdbasis*)

lemma *dim-subset*:
$(S::(real\ \hat{}\ 'n)\ set) \subseteq T \Longrightarrow dim\ S \leq dim\ T$
using *basis-exists*[*of T*] *basis-exists*[*of S*]
by (*metis independent-card-le-dim subset-trans*)

lemma *dim-subset-univ*: *dim* (*S*:: (*real* ^ *'n*) *set*) ≤ *CARD*(*'n*)
by (*metis dim-subset subset-UNIV dim-univ*)

Converses to those.

lemma *card-ge-dim-independent*:
assumes *BV*:(*B*::(*real* ^ *'n*) *set*) ⊆ *V* **and** *iB*:*independent B* **and** *dVB*:*dim V* ≤ *card B*
shows *V* ⊆ *span B*
proof−
{fix *a* **assume** *aV*: *a* ∈ *V*
{assume *aB*: *a* ∉ *span B*
then have *iaB*: *independent* (*insert a B*) **using** *iB aV BV* **by** (*simp add*: *independent-insert*)
from *aV BV* **have** *th0*: *insert a B* ⊆ *V* **by** *blast*
from *aB* **have** *a* ∉*B* **by** (*auto simp add*: *span-superset*)
with *independent-card-le-dim*[*OF th0 iaB*] *dVB independent-bound*[*OF iB*]
have *False* **by** *auto* **}**
then have *a* ∈ *span B* **by** *blast***}**
then show *?thesis* **by** *blast*
qed

lemma *card-le-dim-spanning*:
assumes *BV*: (*B*:: (*real* ^ *'n*) *set*) ⊆ *V* **and** *VB*: *V* ⊆ *span B*
and *fB*: *finite B* **and** *dVB*: *dim V* ≥ *card B*
shows *independent B*
proof−
{fix *a* **assume** *a*: *a* ∈ *B a* ∈ *span* (*B* −{*a*})
from *a fB* **have** *c0*: *card B* ≠ *0* **by** *auto*
from *a fB* **have** *cb*: *card* (*B* −{*a*}) = *card B* − *1* **by** *auto*
from *BV a* **have** *th0*: *B* −{*a*} ⊆ *V* **by** *blast*
{fix *x* **assume** *x*: *x* ∈ *V*
from *a* **have** *eq*: *insert a* (*B* −{*a*}) = *B* **by** *blast*
from *x VB* **have** *x'*: *x* ∈ *span B* **by** *blast*
from *span-trans*[*OF a*(*2*), *unfolded eq*, *OF x'*]
have *x* ∈ *span* (*B* −{*a*}) **. }**
then have *th1*: *V* ⊆ *span* (*B* −{*a*}) **by** *blast*
have *th2*: *finite* (*B* −{*a*}) **using** *fB* **by** *auto*
from *span-card-ge-dim*[*OF th0 th1 th2*]
have *c*: *dim V* ≤ *card* (*B* −{*a*}) **.**
from *c c0 dVB cb* **have** *False* **by** *simp***}**
then show *?thesis* **unfolding** *dependent-def* **by** *blast*
qed

lemma *card-eq-dim*: (*B*:: (*real* ^ *'n*) *set*) ⊆ *V* ⟹ *card B* = *dim V* ⟹ *finite B* ⟹ *independent B* ⟷ *V* ⊆ *span B*
by (*metis order-eq-iff card-le-dim-spanning*
card-ge-dim-independent)

More general size bound lemmas.

lemma *independent-bound-general*:
independent (*S*:: (*real^'n*) *set*) ⟹ *finite S* ∧ *card S* ≤ *dim S*
by (*metis independent-card-le-dim independent-bound subset-refl*)

lemma *dependent-biggerset-general*: (*finite* (*S*:: (*real^'n*) *set*) ⟹ *card S* > *dim S*) ⟹ *dependent S*
using *independent-bound-general*[*of S*] **by** (*metis linorder-not-le*)

lemma *dim-span*: *dim* (*span* (*S*:: (*real ^'n*) *set*)) = *dim S*
proof−
have *th0*: *dim S* ≤ *dim* (*span S*)
by (*auto simp add*: *subset-eq intro*: *dim-subset span-superset*)
from *basis-exists*[*of S*]
obtain *B* **where** *B*: *B* ⊆ *S independent B S* ⊆ *span B card B* = *dim S* **by** *blast*
from *B* **have** *fB*: *finite B card B* = *dim S* **using** *independent-bound* **by** *blast*+
have *bSS*: *B* ⊆ *span S* **using** *B*(*1*) **by** (*metis subset-eq span-inc*)
have *sssB*: *span S* ⊆ *span B* **using** *span-mono*[*OF B*(*3*)] **by** (*simp add*: *span-span*)
from *span-card-ge-dim*[*OF bSS sssB fB*(*1*)] *th0* **show** *?thesis*
using *fB*(*2*) **by** *arith*
qed

lemma *subset-le-dim*: (*S*:: (*real ^'n*) *set*) ⊆ *span T* ⟹ *dim S* ≤ *dim T*
by (*metis dim-span dim-subset*)

lemma *span-eq-dim*: *span* (*S*:: (*real ^'n*) *set*) = *span T* ==> *dim S* = *dim T*
by (*metis dim-span*)

lemma *spans-image*:
assumes *lf*: *linear f* **and** *VB*: *V* ⊆ *span B*
shows *f* ‘ *V* ⊆ *span* (*f* ‘ *B*)
unfolding *span-linear-image*[*OF lf*]
by (*metis VB image-mono*)

lemma *dim-image-le*:
fixes *f* :: *real^'n* ⇒ *real^'m*
assumes *lf*: *linear f* **shows** *dim* (*f* ‘ *S*) ≤ *dim* (*S*:: (*real ^'n*) *set*)
proof−
from *basis-exists*[*of S*] **obtain** *B* **where**
B: *B* ⊆ *S independent B S* ⊆ *span B card B* = *dim S* **by** *blast*
from *B* **have** *fB*: *finite B card B* = *dim S* **using** *independent-bound* **by** *blast*+
have *dim* (*f* ‘ *S*) ≤ *card* (*f* ‘ *B*)
apply (*rule span-card-ge-dim*)
using *lf B fB* **by** (*auto simp add*: *span-linear-image spans-image subset-image-iff*)
also have ... ≤ *dim S* **using** *card-image-le*[*OF fB*(*1*)] *fB* **by** *simp*
finally show *?thesis* **.**
qed

Relation between bases and injectivity/surjectivity of map.

lemma *spanning-surjective-image*:

assumes *us*: *UNIV* ⊆ *span S*
and *lf*: *linear f* **and** *sf*: *surj f*
shows *UNIV* ⊆ *span* (*f* ' *S*)
proof−
have *UNIV* ⊆ *f* ' *UNIV* **using** *sf* **by** (*auto simp add*: *surj-def*)
also have ... ⊆ *span* (*f* ' *S*) **using** *spans-image*[*OF lf us*] .
finally show *?thesis* .
qed

lemma *independent-injective-image*:
assumes *iS*: *independent S* **and** *lf*: *linear f* **and** *fi*: *inj f*
shows *independent* (*f* ' *S*)
proof−
{fix *a* **assume** *a*: *a* ∈ *S f a* ∈ *span* (*f* ' *S* − {*f a*})
have *eq*: *f* ' *S* − {*f a*} = *f* ' (*S* − {*a*}) **using** *fi*
by (*auto simp add*: *inj-on-def*)
from *a* **have** *f a* ∈ *f* ' *span* (*S* −{*a*})
unfolding *eq span-linear-image*[*OF lf*, *of S* − {*a*}] **by** *blast*
hence *a* ∈ *span* (*S* −{*a*}) **using** *fi* **by** (*auto simp add*: *inj-on-def*)
with *a*(*1*) *iS* **have** *False* **by** (*simp add*: *dependent-def*) **}**
then show *?thesis* **unfolding** *dependent-def* **by** *blast*
qed

Picking an orthogonal replacement for a spanning set.

definition *pairwise R S* ⟷ (∀ *x* ∈ *S*. ∀ *y*∈ *S*. *x*≠*y* ⟶ *R x y*)

lemma *vector-sub-project-orthogonal*: (*b*::*real*^'*n*) · (*x* − ((*b* · *x*) / (*b* · *b*)) ∗*s b*) = *0*
unfolding *inner-simps smult-conv-scaleR* **by** *auto*

lemma *basis-orthogonal*:
fixes *B* :: (*real* ^'*n*) *set*
assumes *fB*: *finite B*
shows ∃ *C*. *finite C* ∧ *card C* ≤ *card B* ∧ *span C* = *span B* ∧ *pairwise orthogonal C*
(**is** ∃ *C*. *?P B C*)
proof(*induct rule*: *finite-induct*[*OF fB*])
case *1* **thus** *?case* **apply** (*rule exI*[**where** *x*={}]) **by** (*auto simp add*: *pairwise-def*)
next
case (*2 a B*)
note *fB* = ‹*finite B*› **and** *aB* = ‹*a* ∉ *B*›
from ‹∃ *C*. *finite C* ∧ *card C* ≤ *card B* ∧ *span C* = *span B* ∧ *pairwise orthogonal C*›
obtain *C* **where** *C*: *finite C card C* ≤ *card B*
span C = *span B pairwise orthogonal C* **by** *blast*
let *?a* = *a* − *setsum* (λ*x*. (*x* · *a* / (*x* · *x*)) ∗*R x*) *C*
let *?C* = *insert ?a C*
from *C*(*1*) **have** *fC*: *finite ?C* **by** *simp*
from *fB aB C*(*1*,*2*) **have** *cC*: *card ?C* ≤ *card* (*insert a B*) **by** (*simp add*:

card-insert-if)
{fix *x k*
have *th0*: $\bigwedge$(*a*::'*b*::*comm-ring*) *b c*. *a* − (*b* − *c*) = *c* + (*a* − *b*) **by** (*simp add*: *field-simps*)
have *x* − *k* ∗$_R$ (*a* − (∑ *x*∈*C*. (*x* · *a* / (*x* · *x*)) ∗$_R$ *x*)) ∈ *span C* ⟷ *x* − *k* ∗$_R$ *a* ∈ *span C*
apply (*simp only*: *scaleR-right-diff-distrib th0*)
apply (*rule span-add-eq*)
apply (*rule span-mul*)
apply (*rule span-setsum*[*OF C*(*1*)])
apply *clarify*
apply (*rule span-mul*)
by (*rule span-superset*)**}**
then have *SC*: *span ?C* = *span* (*insert a B*)
unfolding *expand-set-eq span-breakdown-eq C*(*3*)[*symmetric*] **by** *auto*
thm *pairwise-def*
{fix *x y* **assume** *xC*: *x* ∈ *?C* **and** *yC*: *y* ∈ *?C* **and** *xy*: *x* ≠ *y*
{assume *xa*: *x* = *?a* **and** *ya*: *y* = *?a*
have *orthogonal x y* **using** *xa ya xy* **by** *blast***}**
moreover
{assume *xa*: *x* = *?a* **and** *ya*: *y* ≠ *?a y* ∈ *C*
from *ya* **have** *Cy*: *C* = *insert y* (*C* − {*y*}) **by** *blast*
have *fth*: *finite* (*C* − {*y*}) **using** *C* **by** *simp*
have *orthogonal x y*
using *xa ya*
unfolding *orthogonal-def xa inner-simps diff-eq-0-iff-eq*
apply *simp*
apply (*subst Cy*)
using *C*(*1*) *fth*
apply (*simp only*: *setsum-clauses*) **unfolding** *smult-conv-scaleR*
apply (*auto simp add*: *inner-simps inner-commute*[*of y a*] *dot-lsum*[*OF fth*])
apply (*rule setsum-0'*)
apply *clarsimp*
apply (*rule C*(*4*)[*unfolded pairwise-def orthogonal-def*, *rule-format*])
by *auto***}**
moreover
{assume *xa*: *x* ≠ *?a x* ∈ *C* **and** *ya*: *y* = *?a*
from *xa* **have** *Cx*: *C* = *insert x* (*C* − {*x*}) **by** *blast*
have *fth*: *finite* (*C* − {*x*}) **using** *C* **by** *simp*
have *orthogonal x y*
using *xa ya*
unfolding *orthogonal-def ya inner-simps diff-eq-0-iff-eq*
apply *simp*
apply (*subst Cx*)
using *C*(*1*) *fth*
apply (*simp only*: *setsum-clauses*) **unfolding** *smult-conv-scaleR*
apply (*subst inner-commute*[*of x*])
apply (*auto simp add*: *inner-simps inner-commute*[*of x a*] *dot-rsum*[*OF fth*])
apply (*rule setsum-0'*)

apply *clarsimp*
apply (*rule C(4)*[*unfolded pairwise-def orthogonal-def*, *rule-format*])
by *auto*}
moreover
{**assume** *xa*: $x \in C$ **and** *ya*: $y \in C$
have *orthogonal x y* **using** *xa ya xy C(4)* **unfolding** *pairwise-def* **by** *blast*}
ultimately have *orthogonal x y* **using** *xC yC* **by** *blast*}
then have *CPO*: *pairwise orthogonal ?C* **unfolding** *pairwise-def* **by** *blast*
from *fC cC SC CPO* **have** *?P* (*insert a B*) *?C* **by** *blast*
then show *?case* **by** *blast*
qed

lemma *orthogonal-basis-exists*:
fixes $V :: (real\ \hat{}\ 'n)\ set$
shows $\exists B.\ independent\ B \wedge B \subseteq span\ V \wedge V \subseteq span\ B \wedge (card\ B = dim\ V) \wedge pairwise\ orthogonal\ B$
proof−
from *basis-exists*[*of V*] **obtain** *B* **where** *B*: $B \subseteq V$ *independent B* $V \subseteq span\ B$ *card B* $= dim\ V$ **by** *blast*
from *B* **have** *fB*: *finite B card B* $= dim\ V$ **using** *independent-bound* **by** *auto*
from *basis-orthogonal*[*OF fB(1)*] **obtain** *C* **where**
C: *finite C card C* $\leq$ *card B span C* $=$ *span B pairwise orthogonal C* **by** *blast*
from *C B*
have *CSV*: $C \subseteq span\ V$ **by** (*metis span-inc span-mono subset-trans*)
from *span-mono*[*OF B(3)*] *C* **have** *SVC*: *span V* $\subseteq$ *span C* **by** (*simp add*: *span-span*)
from *card-le-dim-spanning*[*OF CSV SVC C(1)*] *C*(*2*,*3*) *fB*
have *iC*: *independent C* **by** (*simp add*: *dim-span*)
from *C fB* **have** *card C* $\leq$ *dim V* **by** *simp*
moreover have *dim V* $\leq$ *card C* **using** *span-card-ge-dim*[*OF CSV SVC C(1)*]
by (*simp add*: *dim-span*)
ultimately have *CdV*: *card C* $=$ *dim V* **using** *C(1)* **by** *simp*
from *C B CSV CdV iC* **show** *?thesis* **by** *auto*
qed

lemma *span-eq*: *span S* $=$ *span T* $\longleftrightarrow$ $S \subseteq span\ T \wedge T \subseteq span\ S$
using *span-inc*[*unfolded subset-eq*] **using** *span-mono*[*of T span S*] *span-mono*[*of S span T*]
by(*auto simp add*: *span-span*)

Low-dimensional subset is in a hyperplane (weak orthogonal complement).

lemma *span-not-univ-orthogonal*:
assumes *sU*: *span S* $\neq$ *UNIV*
shows $\exists (a::\ real\ \hat{}\ 'n).\ a \neq 0 \wedge (\forall x \in span\ S.\ a \cdot x = 0)$
proof−
from *sU* **obtain** *a* **where** *a*: $a \notin span\ S$ **by** *blast*
from *orthogonal-basis-exists* **obtain** *B* **where**
B: *independent B B* $\subseteq$ *span S S* $\subseteq$ *span B card B* $=$ *dim S pairwise orthogonal B*

```
    by blast
  from B have fB: finite B card B = dim S using independent-bound by auto
  from span-mono[OF B(2)] span-mono[OF B(3)]
  have sSB: span S = span B by (simp add: span-span)
  let ?a = a − setsum (λb. (a · b / (b · b)) *R b) B
  have setsum (λb. (a · b / (b · b)) *R b) B ∈ span S
    unfolding sSB
    apply (rule span-setsum[OF fB(1)])
    apply clarsimp
    apply (rule span-mul)
    by (rule span-superset)
  with a have a0:?a ≠ 0 by auto
  have ∀ x∈span B. ?a · x = 0
  proof(rule span-induct')
     show subspace (λx. ?a · x = 0) by (auto simp add: subspace-def mem-def
inner-simps smult-conv-scaleR)

next
    {fix x assume x: x ∈ B
      from x have B': B = insert x (B − {x}) by blast
      have fth: finite (B − {x}) using fB by simp
      have ?a · x = 0
        apply (subst B') using fB fth
        unfolding setsum-clauses(2)[OF fth]
        apply simp unfolding inner-simps smult-conv-scaleR
        apply (clarsimp simp add: inner-simps smult-conv-scaleR dot-lsum)
        apply (rule setsum-0', rule ballI)
        unfolding inner-commute
      by (auto simp add: x field-simps intro: B(5)[unfolded pairwise-def orthogonal-def,
rule-format])}
    then show ∀ x ∈ B. ?a · x = 0 by blast
  qed
  with a0 show ?thesis unfolding sSB by (auto intro: exI[where x=?a])
qed

lemma span-not-univ-subset-hyperplane:
  assumes SU: span S ≠ (UNIV ::(real^'n) set)
  shows ∃ a. a ≠0 ∧ span S ⊆ {x. a · x = 0}
  using span-not-univ-orthogonal[OF SU] by auto

lemma lowdim-subset-hyperplane:
  assumes d: dim S < CARD('n::finite)
  shows ∃(a::real ^'n). a ≠ 0 ∧ span S ⊆ {x. a · x = 0}
proof−
  {assume span S = UNIV
    hence dim (span S) = dim (UNIV :: (real ^'n) set) by simp
    hence dim S = CARD('n) by (simp add: dim-span dim-univ)
    with d have False by arith}
  hence th: span S ≠ UNIV by blast
```

from *span-not-univ-subset-hyperplane*[*OF th*] **show** *?thesis* **.**
qed

We can extend a linear basis-basis injection to the whole set.

lemma *linear-indep-image-lemma*:
assumes *lf*: *linear f* **and** *fB*: *finite B*
and *ifB*: *independent* (*f ' B*)
and *fi*: *inj-on f B* **and** *xsB*: $x \in$ *span B*
and *fx*: *f x* = *0*
shows $x = 0$
using *fB ifB fi xsB fx*
proof(*induct arbitrary*: *x rule*: *finite-induct*[*OF fB*])
case *1* **thus** *?case* **by** (*auto simp add*: *span-empty*)
next
case (*2 a b x*)
have *fb*: *finite b* **using** *2.prems* **by** *simp*
have *th0*: *f ' b* $\subseteq$ *f '* (*insert a b*)
apply (*rule image-mono*) **by** *blast*
from *independent-mono*[*OF 2.prems*(*2*) *th0*]
have *ifb*: *independent* (*f ' b*) **.**
have *fib*: *inj-on f b*
apply (*rule subset-inj-on* [*OF 2.prems*(*3*)])
by *blast*
from *span-breakdown*[*of a insert a b*, *simplified*, *OF 2.prems*(*4*)]
obtain *k* **where** *k*: $x - k *_R a \in$ *span* (*b* $-\{a\}$) **by** *blast*
have *f* $(x - k *_R a) \in$ *span* (*f ' b*)
unfolding *span-linear-image*[*OF lf*]
apply (*rule imageI*)
using *k span-mono*[*of b*$-\{a\}$ *b*] **by** *blast*
hence *f x* $- k *_R$ *f a* $\in$ *span* (*f ' b*)
by (*simp add*: *linear-sub*[*OF lf*] *linear-cmul*[*OF lf*])
hence *th*: $-k *_R$ *f a* $\in$ *span* (*f ' b*)
using *2.prems*(*5*) **by** (*simp add*: *vector-smult-lneg*)
{assume *k0*: *k* = *0*
from *k0 k* **have** *x* $\in$ *span* (*b* $-\{a\}$) **by** *simp*
then have *x* $\in$ *span b* **using** *span-mono*[*of b*$-\{a\}$ *b*]
by *blast***}**
moreover
{assume *k0*: *k* $\neq$ *0*
from *span-mul*[*OF th*, *of* $-$ *1*/ *k*] *k0*
have *th1*: *f a* $\in$ *span* (*f ' b*)
by (*auto simp add*: *vector-smult-assoc*)
from *inj-on-image-set-diff*[*OF 2.prems*(*3*), *of insert a b* $\{a\}$, *symmetric*]
have *tha*: *f ' insert a b* $-$ *f ' {a}* = *f ' (insert a b* $-\{a\}$) **by** *blast*
from *2.prems*(*2*)[*unfolded dependent-def bex-simps*(*10*), *rule-format*, *of f a*]
have *f a* $\notin$ *span* (*f ' b*) **using** *tha*
using *2.hyps*(*2*)
2.prems(*3*) **by** *auto*
with *th1* **have** *False* **by** *blast*

```
    then have x ∈ span b by blast}
  ultimately have xsb: x ∈ span b by blast
  from 2.hyps(3)[OF fb ifb fib xsb 2.prems(5)]
  show x = 0 .
qed
```

We can extend a linear mapping from basis.

```
lemma linear-independent-extend-lemma:
  fixes f :: 'a::real-vector ⇒ 'b::real-vector
  assumes fi: finite B and ib: independent B
  shows ∃ g. (∀ x∈ span B. ∀ y∈ span B. g (x + y) = g x + g y)
          ∧ (∀ x∈ span B. ∀ c. g (c*R x) = c *R g x)
          ∧ (∀ x∈ B. g x = f x)
using ib fi
proof(induct rule: finite-induct[OF fi])
  case 1 thus ?case by (auto simp add: span-empty)
next
  case (2 a b)
  from 2.prems 2.hyps have ibf: independent b finite b
    by (simp-all add: independent-insert)
  from 2.hyps(3)[OF ibf] obtain g where
    g: ∀ x∈span b. ∀ y∈span b. g (x + y) = g x + g y
    ∀ x∈span b. ∀ c. g (c *R x) = c *R g x ∀ x∈b. g x = f x by blast
  let ?h = λz. SOME k. (z − k *R a) ∈ span b
  {fix z assume z: z ∈ span (insert a b)
    have th0: z − ?h z *R a ∈ span b
      apply (rule someI-ex)
      unfolding span-breakdown-eq[symmetric]
      using z .
    {fix k assume k: z − k *R a ∈ span b
      have eq: z − ?h z *R a − (z − k*R a) = (k − ?h z) *R a
        by (simp add: field-simps scaleR-left-distrib [symmetric])
      from span-sub[OF th0 k]
      have khz: (k − ?h z) *R a ∈ span b by (simp add: eq)
      {assume k ≠ ?h z hence k0: k − ?h z ≠ 0 by simp
        from k0 span-mul[OF khz, of 1 /(k − ?h z)]
        have a ∈ span b by (simp add: vector-smult-assoc)
        with 2.prems(1) 2.hyps(2) have False
          by (auto simp add: dependent-def)}
      then have k = ?h z by blast}
    with th0 have z − ?h z *R a ∈ span b ∧ (∀ k. z − k *R a ∈ span b ⟶ k =
?h z) by blast}
  note h = this
  let ?g = λz. ?h z *R f a + g (z − ?h z *R a)
  {fix x y assume x: x ∈ span (insert a b) and y: y ∈ span (insert a b)
    have tha: ⋀(x::'a) y a k l. (x + y) − (k + l) *R a = (x − k *R a) + (y − l
*R a)
      by (simp add: algebra-simps)
    have addh: ?h (x + y) = ?h x + ?h y
```

```
    apply (rule conjunct2[OF h, rule-format, symmetric])
    apply (rule span-add[OF x y])
    unfolding tha
    by (metis span-add x y conjunct1[OF h, rule-format])
  have ?g (x + y) = ?g x + ?g y
    unfolding addh tha
    g(1)[rule-format,OF conjunct1[OF h, OF x] conjunct1[OF h, OF y]]
    by (simp add: scaleR-left-distrib)}
 moreover
 {fix x:: 'a and c:: real  assume x: x ∈ span (insert a b)
  have tha: ⋀(x::'a) c k a. c *R x - (c * k) *R a = c *R (x - k *R a)
    by (simp add: algebra-simps)
  have hc: ?h (c *R x) = c * ?h x
    apply (rule conjunct2[OF h, rule-format, symmetric])
    apply (metis span-mul x)
    by (metis tha span-mul x conjunct1[OF h])
  have ?g (c *R x) = c*R ?g x
    unfolding hc tha g(2)[rule-format, OF conjunct1[OF h, OF x]]
    by (simp add: algebra-simps)}
 moreover
 {fix x assume x: x ∈ (insert a b)
  {assume xa: x = a
    have ha1: 1 = ?h a
      apply (rule conjunct2[OF h, rule-format])
      apply (metis span-superset insertI1)
      using conjunct1[OF h, OF span-superset, OF insertI1]
      by (auto simp add: span-0)

    from xa ha1[symmetric] have ?g x = f x
      apply simp
      using g(2)[rule-format, OF span-0, of 0]
      by simp}
  moreover
  {assume xb: x ∈ b
    have h0: 0 = ?h x
      apply (rule conjunct2[OF h, rule-format])
      apply (metis  span-superset x)
      apply simp
      apply (metis span-superset xb)
      done
    have ?g x = f x
      by (simp add: h0[symmetric] g(3)[rule-format, OF xb])}
  ultimately have ?g x = f x using x by blast }
 ultimately show ?case apply - apply (rule exI[where x=?g]) by blast
qed

lemma linear-independent-extend:
 assumes iB: independent (B:: (real ^'n) set)
 shows ∃ g. linear g ∧ (∀ x∈B. g x = f x)
```

```
proof−
  from maximal-independent-subset-extend[of B UNIV] iB
  obtain C where C: B ⊆ C independent C ⋀x. x ∈ span C by auto

  from C(2) independent-bound[of C] linear-independent-extend-lemma[of C f]
  obtain g where g: (∀ x∈ span C. ∀ y∈ span C. g (x + y) = g x + g y)
          ∧ (∀ x∈ span C. ∀ c. g (c∗R x) = c ∗R g x)
          ∧ (∀ x∈ C. g x = f x) by blast
  from g show ?thesis unfolding linear-def using C
    apply clarsimp by blast
qed
```

Can construct an isomorphism between spaces of same dimension.

```
lemma card-le-inj: assumes fA: finite A and fB: finite B
  and c: card A ≤ card B shows (∃f. f ' A ⊆ B ∧ inj-on f A)
using fB c
proof(induct arbitrary: B rule: finite-induct[OF fA])
  case 1 thus ?case by simp
next
  case (2 x s t)
  thus ?case
  proof(induct rule: finite-induct[OF 2.prems(1)])
    case 1    then show ?case by simp
  next
    case (2 y t)
    from 2.prems(1,2,5) 2.hyps(1,2) have cst:card s ≤ card t by simp
    from 2.prems(3) [OF 2.hyps(1) cst] obtain f where
      f: f ' s ⊆ t ∧ inj-on f s by blast
    from f 2.prems(2) 2.hyps(2) show ?case
      apply −
      apply (rule exI[where x = λz. if z = x then y else f z])
      by (auto simp add: inj-on-def)
  qed
qed

lemma card-subset-eq: assumes fB: finite B and AB: A ⊆ B and
  c: card A = card B
  shows A = B
proof−
  from fB AB have fA: finite A by (auto intro: finite-subset)
  from fA fB have fBA: finite (B − A) by auto
  have e: A ∩ (B − A) = {} by blast
  have eq: A ∪ (B − A) = B using AB by blast
  from card-Un-disjoint[OF fA fBA e, unfolded eq c]
  have card (B − A) = 0 by arith
  hence B − A = {} unfolding card-eq-0-iff using fA fB by simp
  with AB show A = B by blast
qed
```

lemma *subspace-isomorphism*:
assumes *s*: *subspace* (*S*:: (*real* ^'*n*) *set*)
and *t*: *subspace* (*T* :: (*real* ^'*m*) *set*)
and *d*: *dim S* = *dim T*
shows ∃*f*. *linear f* ∧ *f* ' *S* = *T* ∧ *inj-on f S*
proof−
from *basis-exists*[*of S*] *independent-bound* **obtain** *B* **where**
B: *B* ⊆ *S independent B S* ⊆ *span B card B* = *dim S* **and** *fB*: *finite B* **by** *blast*
from *basis-exists*[*of T*] *independent-bound* **obtain** *C* **where**
C: *C* ⊆ *T independent C T* ⊆ *span C card C* = *dim T* **and** *fC*: *finite C* **by** *blast*
from *B*(*4*) *C*(*4*) *card-le-inj*[*of B C*] *d* **obtain** *f* **where**
f: *f* ' *B* ⊆ *C inj-on f B* **using** ‹*finite B*› ‹*finite C*› **by** *auto*
from *linear-independent-extend*[*OF B*(*2*)] **obtain** *g* **where**
g: *linear g* ∀ *x*∈ *B*. *g x* = *f x* **by** *blast*
from *inj-on-iff-eq-card*[*OF fB*, *of f*] *f*(*2*)
have *card* (*f* ' *B*) = *card B* **by** *simp*
with *B*(*4*) *C*(*4*) **have** *ceq*: *card* (*f* ' *B*) = *card C* **using** *d*
by *simp*
have *g* ' *B* = *f* ' *B* **using** *g*(*2*)
by (*auto simp add*: *image-iff*)
also have ... = *C* **using** *card-subset-eq*[*OF fC f*(*1*) *ceq*] **.**
finally have *gBC*: *g* ' *B* = *C* **.**
have *gi*: *inj-on g B* **using** *f*(*2*) *g*(*2*)
by (*auto simp add*: *inj-on-def*)
note *g0* = *linear-indep-image-lemma*[*OF g*(*1*) *fB*, *unfolded gBC*, *OF C*(*2*) *gi*]
{fix *x y* **assume** *x*: *x* ∈ *S* **and** *y*: *y* ∈ *S* **and** *gxy*:*g x* = *g y*
from *B*(*3*) *x y* **have** *x'*: *x* ∈ *span B* **and** *y'*: *y* ∈ *span B* **by** *blast*+
from *gxy* **have** *th0*: *g* (*x* − *y*) = *0* **by** (*simp add*: *linear-sub*[*OF g*(*1*)])
have *th1*: *x* − *y* ∈ *span B* **using** *x' y'* **by** (*metis span-sub*)
have *x*=*y* **using** *g0*[*OF th1 th0*] **by** *simp* **}**
then have *giS*: *inj-on g S*
unfolding *inj-on-def* **by** *blast*
from *span-subspace*[*OF B*(*1*,*3*) *s*]
have *g* ' *S* = *span* (*g* ' *B*) **by** (*simp add*: *span-linear-image*[*OF g*(*1*)])
also have ... = *span C* **unfolding** *gBC* **..**
also have ... = *T* **using** *span-subspace*[*OF C*(*1*,*3*) *t*] **.**
finally have *gS*: *g* ' *S* = *T* **.**
from *g*(*1*) *gS giS* **show** *?thesis* **by** *blast*
qed

Linear functions are equal on a subspace if they are on a spanning set.

lemma *subspace-kernel*:
assumes *lf*: *linear f*
shows *subspace* {*x*. *f x* = *0*}
apply (*simp add*: *subspace-def*)
by (*simp add*: *linear-add*[*OF lf*] *linear-cmul*[*OF lf*] *linear-0*[*OF lf*])

lemma *linear-eq-0-span*:

assumes *lf*: *linear f* **and** *f0*: $\forall x{\in}B.\ f\ x = 0$
shows $\forall x \in span\ B.\ f\ x = 0$
proof
fix *x* **assume** *x*: $x \in span\ B$
let *?P* = $\lambda x.\ f\ x = 0$
from *subspace-kernel*[*OF lf*] **have** *subspace ?P* **unfolding** *Collect-def* **.**
with *x f0 span-induct*[*of B ?P x*] **show** $f\ x = 0$ **by** *blast*
qed

lemma *linear-eq-0*:
assumes *lf*: *linear f* **and** *SB*: $S \subseteq span\ B$ **and** *f0*: $\forall x{\in}B.\ f\ x = 0$
shows $\forall x \in S.\ f\ x = 0$
by (*metis linear-eq-0-span*[*OF lf*] *subset-eq SB f0*)

lemma *linear-eq*:
assumes *lf*: *linear f* **and** *lg*: *linear g* **and** *S*: $S \subseteq span\ B$
and *fg*: $\forall\ x{\in}\ B.\ f\ x = g\ x$
shows $\forall x{\in}\ S.\ f\ x = g\ x$
proof−
let *?h* = $\lambda x.\ f\ x - g\ x$
from *fg* **have** *fg'*: $\forall x{\in}\ B.$ *?h* $x = 0$ **by** *simp*
from *linear-eq-0*[*OF linear-compose-sub*[*OF lf lg*] *S fg'*]
show *?thesis* **by** *simp*
qed

lemma *linear-eq-stdbasis*:
assumes *lf*: *linear* (*f*::*real^'m* ⇒ -) **and** *lg*: *linear g*
and *fg*: $\forall i.\ f\ (basis\ i) = g(basis\ i)$
shows $f = g$
proof−
let *?U* = *UNIV* :: *'m set*
let *?I* = {*basis i*:: *real^'m*|*i*. $i \in$ *?U*}
{fix *x* **assume** *x*: $x \in$ (*UNIV* :: (*real^'m*) *set*)
from *equalityD2*[*OF span-stdbasis*]
have *IU*: (*UNIV* :: (*real^'m*) *set*) $\subseteq$ *span ?I* **by** *blast*
from *linear-eq*[*OF lf lg IU*] *fg x*
have $f\ x = g\ x$ **unfolding** *Collect-def Ball-def mem-def* **by** *metis***}**
then show *?thesis* **by** (*auto intro*: *ext*)
qed

Similar results for bilinear functions.

lemma *bilinear-eq*:
assumes *bf*: *bilinear f*
and *bg*: *bilinear g*
and *SB*: $S \subseteq span\ B$ **and** *TC*: $T \subseteq span\ C$
and *fg*: $\forall x{\in}\ B.\ \forall y{\in}\ C.\ f\ x\ y = g\ x\ y$
shows $\forall x{\in}S.\ \forall y{\in}T.\ f\ x\ y = g\ x\ y$
proof−
let *?P* = $\lambda x.\ \forall y{\in}\ span\ C.\ f\ x\ y = g\ x\ y$

from *bf bg* **have** *sp*: *subspace ?P*
unfolding *bilinear-def linear-def subspace-def bf bg*
by(*auto simp add*: *span-0 mem-def bilinear-lzero*[*OF bf*] *bilinear-lzero*[*OF bg*] *span-add Ball-def intro*: *bilinear-ladd*[*OF bf*])

have $\forall x \in span\ B.\ \forall y \in span\ C.\ f\ x\ y = g\ x\ y$
apply −
apply (*rule ballI*)
apply (*rule span-induct*[*of B ?P*])
defer
apply (*rule sp*)
apply *assumption*
apply (*clarsimp simp add*: *Ball-def*)
apply (*rule-tac* $P{=}\lambda y.\ f\ xa\ y = g\ xa\ y$ **and** $S{=}C$ **in** *span-induct*)
using *fg*
apply (*auto simp add*: *subspace-def*)
using *bf bg* **unfolding** *bilinear-def linear-def*
by(*auto simp add*: *span-0 mem-def bilinear-rzero*[*OF bf*] *bilinear-rzero*[*OF bg*] *span-add Ball-def intro*: *bilinear-ladd*[*OF bf*])
then show *?thesis* **using** *SB TC* **by** (*auto intro*: *ext*)
qed

lemma *bilinear-eq-stdbasis*:
assumes *bf*: *bilinear* (f:: *real^'m* ⇒ *real^'n* ⇒ -)
and *bg*: *bilinear g*
and *fg*: $\forall i\ j.\ f\ (basis\ i)\ (basis\ j) = g\ (basis\ i)\ (basis\ j)$
shows $f = g$
proof−
from *fg* **have** *th*: $\forall x \in \{basis\ i |\ i.\ i \in (UNIV :: 'm\ set)\}.\ \forall y \in \{basis\ j\ |j.\ j \in (UNIV :: 'n\ set)\}.\ f\ x\ y = g\ x\ y$ **by** *blast*
from *bilinear-eq*[*OF bf bg equalityD2*[*OF span-stdbasis*] *equalityD2*[*OF span-stdbasis*] *th*] **show** *?thesis* **by** (*blast intro*: *ext*)
qed

Detailed theorems about left and right invertibility in general case.

lemma *left-invertible-transpose*:
$(\exists(B).\ B\ **\ transpose\ (A) = mat\ (1::'a::comm\text{-}semiring\text{-}1)) \longleftrightarrow (\exists(B).\ A\ **\ B = mat\ 1)$
by (*metis matrix-transpose-mul transpose-mat transpose-transpose*)

lemma *right-invertible-transpose*:
$(\exists(B).\ transpose\ (A)\ **\ B = mat\ (1::'a::comm\text{-}semiring\text{-}1)) \longleftrightarrow (\exists(B).\ B\ **\ A = mat\ 1)$
by (*metis matrix-transpose-mul transpose-mat transpose-transpose*)

lemma *linear-injective-left-inverse*:
assumes *lf*: *linear* (f::*real ^'n* ⇒ *real ^'m*) **and** *fi*: *inj f*
shows $\exists g.\ linear\ g \wedge g\ o\ f = id$
proof−

from *linear-independent-extend*[*OF independent-injective-image*, *OF independent-stdbasis*, *OF lf fi*]
obtain *h*:: *real ^'m ⇒ real ^'n* **where** *h*: *linear h* ∀ *x* ∈ *f ' {basis i|i. i* ∈ (*UNIV*::*'n set*)}. *h x* = *inv f x* **by** *blast*
from *h*(*2*)
have *th*: ∀ *i*. (*h* ∘ *f*) (*basis i*) = *id* (*basis i*)
using *inv-o-cancel*[*OF fi*, *unfolded stupid-ext*[*symmetric*] *id-def o-def*]
by *auto*

from *linear-eq-stdbasis*[*OF linear-compose*[*OF lf h*(*1*)] *linear-id th*]
have *h o f* = *id* **.**
then show *?thesis* **using** *h*(*1*) **by** *blast*
qed

lemma *linear-surjective-right-inverse*:
assumes *lf*: *linear* (*f*:: *real ^'m ⇒ real ^'n*) **and** *sf*: *surj f*
shows ∃ *g*. *linear g* ∧ *f o g* = *id*
proof−
from *linear-independent-extend*[*OF independent-stdbasis*]
obtain *h*:: *real ^'n ⇒ real ^'m* **where**
h: *linear h* ∀ *x*∈ {*basis i*| *i*. *i*∈ (*UNIV* :: *'n set*)}. *h x* = *inv f x* **by** *blast*
from *h*(*2*)
have *th*: ∀ *i*. (*f o h*) (*basis i*) = *id* (*basis i*)
using *sf*
apply (*auto simp add*: *surj-iff o-def stupid-ext*[*symmetric*])
apply (*erule-tac x*=*basis i* **in** *allE*)
by *auto*

from *linear-eq-stdbasis*[*OF linear-compose*[*OF h*(*1*) *lf*] *linear-id th*]
have *f o h* = *id* **.**
then show *?thesis* **using** *h*(*1*) **by** *blast*
qed

lemma *matrix-left-invertible-injective*:
(∃ *B*. (*B*::*real^'m^'n*) ** (*A*::*real^'n^'m*) = *mat 1*) ⟷ (∀ *x y*. *A* ∗*v x* = *A* ∗*v y* ⟶ *x* = *y*)
proof−
{fix *B*:: *real^'m^'n* **and** *x y* **assume** *B*: *B* ** *A* = *mat 1* **and** *xy*: *A* ∗*v x* = *A*∗*v y*
from *xy* **have** *B*∗*v* (*A* ∗*v x*) = *B* ∗*v* (*A*∗*v y*) **by** *simp*
hence *x* = *y*
unfolding *matrix-vector-mul-assoc B matrix-vector-mul-lid* **.}**
moreover
{assume *A*: ∀ *x y*. *A* ∗*v x* = *A* ∗*v y* ⟶ *x* = *y*
hence *i*: *inj* (*op* ∗*v A*) **unfolding** *inj-on-def* **by** *auto*
from *linear-injective-left-inverse*[*OF matrix-vector-mul-linear i*]
obtain *g* **where** *g*: *linear g g o op* ∗*v A* = *id* **by** *blast*
have *matrix g* ** *A* = *mat 1*
unfolding *matrix-eq matrix-vector-mul-lid matrix-vector-mul-assoc*[*symmetric*]

matrix-works[*OF g*(*1*)]
using *g*(*2*) **by** (*simp add*: *o-def id-def stupid-ext*)
then have ∃ *B*. (*B*::*real* ^′*m*^′*n*) ** *A* = *mat 1* **by** *blast*}
ultimately show *?thesis* **by** *blast*
qed

lemma *matrix-left-invertible-ker*:
(∃ *B*. (*B*::*real* ^′*m*^′*n*) ** (*A*::*real*^′*n*^′*m*) = *mat 1*) ⟷ (∀ *x*. *A* *v *x* = *0* ⟶ *x* = *0*)
unfolding *matrix-left-invertible-injective*
using *linear-injective-0*[*OF matrix-vector-mul-linear*, *of A*]
by (*simp add*: *inj-on-def*)

lemma *matrix-right-invertible-surjective*:
(∃ *B*. (*A*::*real*^′*n*^′*m*) ** (*B*::*real*^′*m*^′*n*) = *mat 1*) ⟷ *surj* (λ*x*. *A* *v *x*)
proof−
{fix *B* :: *real* ^′*m*^′*n* **assume** *AB*: *A* ** *B* = *mat 1*
{fix *x* :: *real* ^ ′*m*
have *A* *v (*B* *v *x*) = *x*
by (*simp add*: *matrix-vector-mul-lid matrix-vector-mul-assoc AB*)}
hence *surj* (*op* *v *A*) **unfolding** *surj-def* **by** *metis* **}**
moreover
{assume *sf*: *surj* (*op* *v *A*)
from *linear-surjective-right-inverse*[*OF matrix-vector-mul-linear sf*]
obtain *g*:: *real* ^′*m* ⇒ *real* ^′*n* **where** *g*: *linear g op* *v *A o g* = *id*
by *blast*

have *A* ** (*matrix g*) = *mat 1*
unfolding *matrix-eq matrix-vector-mul-lid*
matrix-vector-mul-assoc[*symmetric*] *matrix-works*[*OF g*(*1*)]
using *g*(*2*) **unfolding** *o-def stupid-ext*[*symmetric*] *id-def*
.
hence ∃ *B*. *A* ** (*B*::*real*^′*m*^′*n*) = *mat 1* **by** *blast*
}
ultimately show *?thesis* **unfolding** *surj-def* **by** *blast*
qed

lemma *matrix-left-invertible-independent-columns*:
fixes *A* :: *real*^′*n*^′*m*
shows (∃(*B*::*real* ^′*m*^′*n*). *B* ** *A* = *mat 1*) ⟷ (∀ *c*. *setsum* (λ*i*. *c i* *s *column i A*) (*UNIV* :: ′*n set*) = *0* ⟶ (∀ *i*. *c i* = *0*))
(**is** *?lhs* ⟷ *?rhs*)
proof−
let *?U* = *UNIV* :: ′*n set*
{assume *k*: ∀ *x*. *A* *v *x* = *0* ⟶ *x* = *0*
{fix *c i* **assume** *c*: *setsum* (λ*i*. *c i* *s *column i A*) *?U* = *0*
and *i*: *i* ∈ *?U*
let *?x* = χ *i*. *c i*
have *th0*:*A* *v *?x* = *0*

```
      using c
      unfolding matrix-mult-vsum Cart-eq
      by auto
    from k[rule-format, OF th0] i
    have c i = 0 by (vector Cart-eq)}
  hence ?rhs by blast}
 moreover
 {assume H: ?rhs
  {fix x assume x: A *v x = 0
   let ?c = λi. ((x$i ):: real)
   from H[rule-format, of ?c, unfolded matrix-mult-vsum[symmetric], OF x]
   have x = 0 by vector}}
 ultimately show ?thesis unfolding matrix-left-invertible-ker by blast
qed

lemma matrix-right-invertible-independent-rows:
 fixes A :: real^'n^'m
 shows (∃(B::real^'m^'n). A ** B = mat 1) ⟷ (∀ c. setsum (λi. c i *s row i
A) (UNIV :: 'm set) = 0 ⟶ (∀ i. c i = 0))
 unfolding left-invertible-transpose[symmetric]
   matrix-left-invertible-independent-columns
 by (simp add: column-transpose)

lemma matrix-right-invertible-span-columns:
  (∃(B::real ^'n^'m). (A::real ^'m^'n) ** B = mat 1) ⟷ span (columns A) =
UNIV (is ?lhs = ?rhs)
proof−
 let ?U = UNIV :: 'm set
 have fU: finite ?U by simp
 have lhseq: ?lhs ⟷ (∀ y. ∃(x::real^'m). setsum (λi. (x$i) *s column i A) ?U
= y)
   unfolding matrix-right-invertible-surjective matrix-mult-vsum surj-def
   apply (subst eq-commute) ..
 have rhseq: ?rhs ⟷ (∀ x. x ∈ span (columns A)) by blast
 {assume h: ?lhs
  {fix x:: real ^'n
     from h[unfolded lhseq, rule-format, of x] obtain y:: real ^'m
       where y: setsum (λi. (y$i) *s column i A) ?U = x by blast
     have x ∈ span (columns A)
       unfolding y[symmetric]
       apply (rule span-setsum[OF fU])
       apply clarify
       unfolding smult-conv-scaleR
       apply (rule span-mul)
       apply (rule span-superset)
       unfolding columns-def
       by blast}
   then have ?rhs unfolding rhseq by blast}
 moreover
```

{assume *h*:*?rhs*
let *?P* = λ(*y*::*real* ^'*n*). ∃(*x*::*real*^'*m*). *setsum* (λ*i*. (*x*$*i*) ∗*s column i A*) *?U* = *y*
{fix *y* **have** *?P y*
proof(*rule span-induct-alt*[*of ?P columns A*, *folded smult-conv-scaleR*])
show ∃*x*::*real* ^ '*m*. *setsum* (λ*i*. (*x*$*i*) ∗*s column i A*) *?U* = *0*
by (*rule exI*[**where** *x*=*0*], *simp*)
next
fix *c y1 y2* **assume** *y1*: *y1* ∈ *columns A* **and** *y2*: *?P y2*
from *y1* **obtain** *i* **where** *i*: *i* ∈ *?U y1* = *column i A*
unfolding *columns-def* **by** *blast*
from *y2* **obtain** *x*:: *real* ^'*m* **where**
x: *setsum* (λ*i*. (*x*$*i*) ∗*s column i A*) *?U* = *y2* **by** *blast*
let *?x* = (χ *j*. *if j* = *i then c* + (*x*$*i*) *else* (*x*$*j*))::*real*^'*m*
show *?P* (*c*∗*s y1* + *y2*)
proof(*rule exI*[**where** *x*= *?x*], *vector*, *auto simp add*: *i x*[*symmetric*] *cond-value-iff right-distrib cond-application-beta cong del*: *if-weak-cong*)
fix *j*
have *th*: ∀ *xa* ∈ *?U*. (*if xa* = *i then* (*c* + (*x*$*i*)) ∗ ((*column xa A*)$*j*) *else* (*x*$*xa*) ∗ ((*column xa A*$*j*))) = (*if xa* = *i then c* ∗ ((*column i A*)$*j*) *else 0*) + ((*x*$*xa*) ∗ ((*column xa A*)$*j*)) **using** *i*(*1*)
by (*simp add*: *field-simps*)
have *setsum* (λ*xa*. *if xa* = *i then* (*c* + (*x*$*i*)) ∗ ((*column xa A*)$*j*) *else* (*x*$*xa*) ∗ ((*column xa A*$*j*))) *?U* = *setsum* (λ*xa*. (*if xa* = *i then c* ∗ ((*column i A*)$*j*) *else 0*) + ((*x*$*xa*) ∗ ((*column xa A*)$*j*))) *?U*
apply (*rule setsum-cong*[*OF refl*])
using *th* **by** *blast*
also have ... = *setsum* (λ*xa*. *if xa* = *i then c* ∗ ((*column i A*)$*j*) *else 0*) *?U* + *setsum* (λ*xa*. ((*x*$*xa*) ∗ ((*column xa A*)$*j*))) *?U*
by (*simp add*: *setsum-addf*)
also have ... = *c* ∗ ((*column i A*)$*j*) + *setsum* (λ*xa*. ((*x*$*xa*) ∗ ((*column xa A*)$*j*))) *?U*
unfolding *setsum-delta*[*OF fU*]
using *i*(*1*) **by** *simp*
finally show *setsum* (λ*xa*. *if xa* = *i then* (*c* + (*x*$*i*)) ∗ ((*column xa A*)$*j*) *else* (*x*$*xa*) ∗ ((*column xa A*$*j*))) *?U* = *c* ∗ ((*column i A*)$*j*) + *setsum* (λ*xa*. ((*x*$*xa*) ∗ ((*column xa A*)$*j*))) *?U* **.**
qed
next
show *y* ∈ *span* (*columns A*) **unfolding** *h* **by** *blast*
qed}
then have *?lhs* **unfolding** *lhseq* **..}**
ultimately show *?thesis* **by** *blast*
qed

lemma *matrix-left-invertible-span-rows*:
(∃(*B*::*real*^'*m*^'*n*). *B* ** (*A*::*real*^'*n*^'*m*) = *mat 1*) ⟷ *span* (*rows A*) = *UNIV*
unfolding *right-invertible-transpose*[*symmetric*]

```
  unfolding columns-transpose[symmetric]
  unfolding matrix-right-invertible-span-columns
 ..
```

An injective map $(real, 'n)\ cart \Rightarrow (real, 'n)\ cart$ is also surjective.

```
lemma linear-injective-imp-surjective:
  assumes lf: linear (f:: real ^'n ⇒ real ^'n) and fi: inj f
  shows surj f
proof−
  let ?U = UNIV :: (real ^'n) set
  from basis-exists[of ?U] obtain B
    where B: B ⊆ ?U independent B ?U ⊆ span B card B = dim ?U
    by blast
  from B(4) have d: dim ?U = card B by simp
  have th: ?U ⊆ span (f ' B)
    apply (rule card-ge-dim-independent)
    apply blast
    apply (rule independent-injective-image[OF B(2) lf fi])
    apply (rule order-eq-refl)
    apply (rule sym)
    unfolding d
    apply (rule card-image)
    apply (rule subset-inj-on[OF fi])
    by blast
  from th show ?thesis
    unfolding span-linear-image[OF lf] surj-def
    using B(3) by blast
qed
```

And vice versa.

```
lemma surjective-iff-injective-gen:
  assumes fS: finite S and fT: finite T and c: card S = card T
  and ST: f ' S ⊆ T
  shows (∀ y ∈ T. ∃ x ∈ S. f x = y) ⟷ inj-on f S (is ?lhs ⟷ ?rhs)
proof−
  {assume h: ?lhs
    {fix x y assume x: x ∈ S and y: y ∈ S and f: f x = f y
      from x fS have S0: card S ≠ 0 by auto
      {assume xy: x ≠ y
        have th: card S ≤ card (f ' (S − {y}))
          unfolding c
          apply (rule card-mono)
          apply (rule finite-imageI)
          using fS apply simp
          using h xy x y f unfolding subset-eq image-iff
          apply auto
          apply (case-tac xa = f x)
          apply (rule bexI[where x=x])
          apply auto
```

```
        done
      also have  ... ≤ card (S -{y})
        apply (rule card-image-le)
        using fS by simp
      also have ... ≤ card S - 1 using y fS by simp
      finally have False  using S0 by arith }
    then have x = y by blast}
  then have ?rhs unfolding inj-on-def by blast}
 moreover
 {assume h: ?rhs
  have f ' S = T
    apply (rule card-subset-eq[OF fT ST])
    unfolding card-image[OF h] using c .
  then have ?lhs by blast}
 ultimately show ?thesis by blast
qed

lemma linear-surjective-imp-injective:
 assumes lf: linear (f::real ^'n => real ^'n) and sf: surj f
 shows inj f
proof-
 let ?U = UNIV :: (real ^'n) set
 from basis-exists[of ?U] obtain B
   where B: B ⊆ ?U independent B ?U ⊆ span B and d: card B = dim ?U
   by blast
 {fix x assume x: x ∈ span B and fx: f x = 0
  from B(2) have fB: finite B using independent-bound by auto
  have fBi: independent (f ' B)
    apply (rule card-le-dim-spanning[of f ' B ?U])
    apply blast
    using sf B(3)
    unfolding span-linear-image[OF lf] surj-def subset-eq image-iff
    apply blast
    using fB apply (blast intro: finite-imageI)
    unfolding d[symmetric]
    apply (rule card-image-le)
    apply (rule fB)
    done
  have th0: dim ?U ≤ card (f ' B)
    apply (rule span-card-ge-dim)
    apply blast
    unfolding span-linear-image[OF lf]
    apply (rule subset-trans[where B = f ' UNIV])
    using sf unfolding surj-def apply blast
    apply (rule image-mono)
    apply (rule B(3))
    apply (metis finite-imageI fB)
    done
```

moreover have *card* (*f* ' *B*) ≤ *card B*
by (*rule card-image-le*, *rule fB*)
ultimately have *th1*: *card B* = *card* (*f* ' *B*) **unfolding** *d* **by** *arith*
have *fiB*: *inj-on f B*
unfolding *surjective-iff-injective-gen*[*OF fB finite-imageI*[*OF fB*] *th1 subset-refl*, *symmetric*] **by** *blast*
from *linear-indep-image-lemma*[*OF lf fB fBi fiB x*] *fx*
have *x* = *0* **by** *blast*}
note *th* = *this*
from *th* **show** *?thesis* **unfolding** *linear-injective-0*[*OF lf*]
using *B*(*3*) **by** *blast*
qed

Hence either is enough for isomorphism.

lemma *left-right-inverse-eq*:
assumes *fg*: *f o g* = *id* **and** *gh*: *g o h* = *id*
shows *f* = *h*
proof−
have *f* = *f o* (*g o h*) **unfolding** *gh* **by** *simp*
also have ... = (*f o g*) *o h* **by** (*simp add*: *o-assoc*)
finally show *f* = *h* **unfolding** *fg* **by** *simp*
qed

lemma *isomorphism-expand*:
f o g = *id* ∧ *g o f* = *id* ⟷ (∀ *x*. *f*(*g x*) = *x*) ∧ (∀ *x*. *g*(*f x*) = *x*)
by (*simp add*: *expand-fun-eq o-def id-def*)

lemma *linear-injective-isomorphism*:
assumes *lf*: *linear* (*f* :: *real*^'*n* ⇒ *real* ^'*n*) **and** *fi*: *inj f*
shows ∃*f* '. *linear f* ' ∧ (∀ *x*. *f* ' (*f x*) = *x*) ∧ (∀ *x*. *f* (*f* ' *x*) = *x*)
unfolding *isomorphism-expand*[*symmetric*]
using *linear-surjective-right-inverse*[*OF lf linear-injective-imp-surjective*[*OF lf fi*]]
linear-injective-left-inverse[*OF lf fi*]
by (*metis left-right-inverse-eq*)

lemma *linear-surjective-isomorphism*:
assumes *lf*: *linear* (*f*::*real* ^'*n* ⇒ *real* ^'*n*) **and** *sf*: *surj f*
shows ∃*f* '. *linear f* ' ∧ (∀ *x*. *f* ' (*f x*) = *x*) ∧ (∀ *x*. *f* (*f* ' *x*) = *x*)
unfolding *isomorphism-expand*[*symmetric*]
using *linear-surjective-right-inverse*[*OF lf sf*] *linear-injective-left-inverse*[*OF lf linear-surjective-imp-injective*[*O*
lf sf]]
by (*metis left-right-inverse-eq*)

Left and right inverses are the same for (*real*, *'n*) *cart* ⇒ (*real*, *'n*) *cart*.

lemma *linear-inverse-left*:
assumes *lf*: *linear* (*f*::*real* ^'*n* ⇒ *real* ^'*n*) **and** *lf* ': *linear f* '
shows *f o f* ' = *id* ⟷ *f* ' *o f* = *id*
proof−
{**fix** *f f* '':: *real* ^'*n* ⇒ *real* ^'*n*

```
    assume lf: linear f linear f' and f: f o f' = id
    from f have sf: surj f

      apply (auto simp add: o-def stupid-ext[symmetric] id-def surj-def)
      by metis
    from linear-surjective-isomorphism[OF lf(1) sf] lf f
    have f' o f = id unfolding stupid-ext[symmetric] o-def id-def
      by metis}
  then show ?thesis using lf lf' by metis
qed
```

Moreover, a one-sided inverse is automatically linear.

```
lemma left-inverse-linear:
  assumes lf: linear (f::real ^'n ⇒ real ^'n) and gf: g o f = id
  shows linear g
proof−
 from gf have fi: inj f apply (auto simp add: inj-on-def o-def id-def stupid-ext[symmetric])
    by metis
  from linear-injective-isomorphism[OF lf fi]
  obtain h:: real ^'n ⇒ real ^'n where
    h: linear h ∀ x. h (f x) = x ∀ x. f (h x) = x by blast
  have h = g apply (rule ext) using gf h(2,3)
    apply (simp add: o-def id-def stupid-ext[symmetric])
    by metis
  with h(1) show ?thesis by blast
qed

lemma right-inverse-linear:
  assumes lf: linear (f:: real ^'n ⇒ real ^'n) and gf: f o g = id
  shows linear g
proof−
 from gf have fi: surj f apply (auto simp add: surj-def o-def id-def stupid-ext[symmetric])
    by metis
  from linear-surjective-isomorphism[OF lf fi]
  obtain h:: real ^'n ⇒ real ^'n where
    h: linear h ∀ x. h (f x) = x ∀ x. f (h x) = x by blast
  have h = g apply (rule ext) using gf h(2,3)
    apply (simp add: o-def id-def stupid-ext[symmetric])
    by metis
  with h(1) show ?thesis by blast
qed
```

The same result in terms of square matrices.

```
lemma matrix-left-right-inverse:
  fixes A A' :: real ^'n^'n
  shows A ** A' = mat 1 ⟷ A' ** A = mat 1
proof−
  {fix A A' :: real ^'n^'n assume AA': A ** A' = mat 1
    have sA: surj (op *v A)
```

unfolding *surj-def*
apply *clarify*
apply (*rule-tac x*=($A' *v y$) **in** *exI*)
by (*simp add*: *matrix-vector-mul-assoc AA' matrix-vector-mul-lid*)
from *linear-surjective-isomorphism*[*OF matrix-vector-mul-linear sA*]
obtain f' :: *real ^'n* ⇒ *real ^'n*
where f': *linear* f' ∀ *x*. f' (*A *v x*) = *x* ∀ *x*. *A *v* f' *x* = *x* **by** *blast*
have *th*: *matrix* f' ** *A* = *mat 1*
by (*simp add*: *matrix-eq matrix-works*[*OF* f'(*1*)] *matrix-vector-mul-assoc*[*symmetric*] *matrix-vector-mul-lid* f'(*2*)[*rule-format*])
hence (*matrix* f' ** *A*) ** A' = *mat 1* ** A' **by** *simp*
hence *matrix* f' = A' **by** (*simp add*: *matrix-mul-assoc*[*symmetric*] *AA' matrix-mul-rid matrix-mul-lid*)
hence *matrix* f' ** *A* = A' ** *A* **by** *simp*
hence A' ** *A* = *mat 1* **by** (*simp add*: *th*)**}**
then show *?thesis* **by** *blast*
qed

Considering an n-element vector as an n-by-1 or 1-by-n matrix.

definition *rowvector v* = (χ *i j*. (*v*$*j*))

definition *columnvector v* = (χ *i j*. (*v*$*i*))

lemma *transpose-columnvector*:
transpose(*columnvector v*) = *rowvector v*
by (*simp add*: *transpose-def rowvector-def columnvector-def Cart-eq*)

lemma *transpose-rowvector*: *transpose*(*rowvector v*) = *columnvector v*
by (*simp add*: *transpose-def columnvector-def rowvector-def Cart-eq*)

lemma *dot-rowvector-columnvector*:
columnvector (*A *v v*) = *A ** *columnvector v*
by (*vector columnvector-def matrix-matrix-mult-def matrix-vector-mult-def*)

lemma *dot-matrix-product*: (*x*::*real^'n*) · *y* = (((*rowvector x* ::*real^'n^1*) ** (*columnvector y* :: *real^1^'n*))$*1*)$*1*
by (*vector matrix-matrix-mult-def rowvector-def columnvector-def inner-vector-def*)

lemma *dot-matrix-vector-mul*:
fixes *A B* :: *real ^'n ^'n* **and** *x y* :: *real ^'n*
shows (*A *v x*) · (*B *v y*) =
(((*rowvector x* :: *real^'n^1*) ** ((*transpose A ** B*) ** (*columnvector y* :: *real ^1^'n*)))$*1*)$*1*
unfolding *dot-matrix-product transpose-columnvector*[*symmetric*]
dot-rowvector-columnvector matrix-transpose-mul matrix-mul-assoc **..**

13.16 Infinity norm

definition *infnorm* (*x*::*real^'n*) = *Sup* {*abs*(*x*$*i*) |*i*. *i*∈ (*UNIV* :: *'n set*)}

lemma *numseg-dimindex-nonempty*: $\exists i.\ i \in (UNIV :: 'n\ set)$
by *auto*

lemma *infnorm-set-image*:
$\{abs(x\$i) \mid i.\ i \in (UNIV :: -\ set)\} =$
$(\lambda i.\ abs(x\$i))\ `\ (UNIV)$ **by** *blast*

lemma *infnorm-set-lemma*:
shows *finite* $\{abs((x::'a::abs\ \hat{}\ 'n)\$i) \mid i.\ i \in (UNIV :: 'n\ set)\}$
and $\{abs(x\$i) \mid i.\ i \in (UNIV :: 'n::finite\ set)\} \neq \{\}$
unfolding *infnorm-set-image*
by (*auto intro*: *finite-imageI*)

lemma *infnorm-pos-le*: $0 \leq infnorm\ (x::real\hat{}\ 'n)$
unfolding *infnorm-def*
unfolding *Sup-finite-ge-iff*[*OF infnorm-set-lemma*]
unfolding *infnorm-set-image*
by *auto*

lemma *infnorm-triangle*: $infnorm\ ((x::real\hat{}\ 'n) + y) \leq infnorm\ x + infnorm\ y$
proof−
have *th*: $\bigwedge x\ y\ (z::real).\ x - y <= z \longleftrightarrow x - z <= y$ **by** *arith*
have *th1*: $\bigwedge S\ f.\ f\ `\ S = \{\ f\ i \mid\ i.\ i \in S\}$ **by** *blast*
have *th2*: $\bigwedge x\ (y::real).\ abs(x + y) - abs(x) <= abs(y)$ **by** *arith*
show *?thesis*
unfolding *infnorm-def*
unfolding *Sup-finite-le-iff*[*OF infnorm-set-lemma*]
apply (*subst diff-le-eq*[*symmetric*])
unfolding *Sup-finite-ge-iff*[*OF infnorm-set-lemma*]
unfolding *infnorm-set-image bex-simps*
apply (*subst th*)
unfolding *th1*
unfolding *Sup-finite-ge-iff*[*OF infnorm-set-lemma*]

unfolding *infnorm-set-image ball-simps bex-simps*
apply *simp*
apply (*metis th2*)
done
qed

lemma *infnorm-eq-0*: $infnorm\ x = 0 \longleftrightarrow (x::real\ \hat{}\ 'n) = 0$
proof−
have $infnorm\ x <= 0 \longleftrightarrow x = 0$
unfolding *infnorm-def*
unfolding *Sup-finite-le-iff*[*OF infnorm-set-lemma*]
unfolding *infnorm-set-image ball-simps*
by *vector*
then show *?thesis* **using** *infnorm-pos-le*[*of x*] **by** *simp*

qed

lemma *infnorm-0*: *infnorm 0 = 0*
by (*simp add*: *infnorm-eq-0*)

lemma *infnorm-neg*: *infnorm (− x) = infnorm x*
unfolding *infnorm-def*
apply (*rule cong*[*of Sup Sup*])
apply *blast*
apply (*rule set-ext*)
apply *auto*
done

lemma *infnorm-sub*: *infnorm (x − y) = infnorm (y − x)*
proof−
have *y − x = − (x − y)* **by** *simp*
then show *?thesis* **by** (*metis infnorm-neg*)
qed

lemma *real-abs-sub-infnorm*: *| infnorm x − infnorm y| ≤ infnorm (x − y)*
proof−
have *th*: *⋀*(*nx*::*real*) *n ny. nx <= n + ny ⟹ ny <= n + nx ==> |nx − ny|*
<= n
by *arith*
from *infnorm-triangle*[*of x − y y*] *infnorm-triangle*[*of x − y −x*]
have *ths*: *infnorm x ≤ infnorm (x − y) + infnorm y*
infnorm y ≤ infnorm (x − y) + infnorm x
by (*simp-all add*: *field-simps infnorm-neg diff-def*[*symmetric*])
from *th*[*OF ths*] **show** *?thesis* **.**
qed

lemma *real-abs-infnorm*: *|infnorm x| = infnorm x*
using *infnorm-pos-le*[*of x*] **by** *arith*

lemma *component-le-infnorm*:
shows *|x$i| ≤ infnorm* (*x*::*real^'n*)
proof−
let *?U = UNIV* :: *'n set*
let *?S* = {*|x$i| |i. i∈ ?U*}
have *fS*: *finite ?S* **unfolding** *image-Collect*[*symmetric*]
apply (*rule finite-imageI*) **unfolding** *Collect-def mem-def* **by** *simp*
have *S0*: *?S ≠* {} **by** *blast*
have *th1*: *⋀S f. f ‘ S =* { *f i| i. i ∈ S*} **by** *blast*
from *Sup-finite-in*[*OF fS S0*]
show *?thesis* **unfolding** *infnorm-def infnorm-set-image*
by (*metis Sup-finite-ge-iff finite finite-imageI UNIV-not-empty image-is-empty*
rangeI order-refl)
qed

lemma *infnorm-mul-lemma*: *infnorm*(*a* ∗*s x*) <= |*a*| ∗ *infnorm x*
apply (*subst infnorm-def*)
unfolding *Sup-finite-le-iff*[*OF infnorm-set-lemma*]
unfolding *infnorm-set-image ball-simps*
apply (*simp add*: *abs-mult*)
apply (*rule allI*)
apply (*cut-tac component-le-infnorm*[*of x*])
apply (*rule mult-mono*)
apply *auto*
done

lemma *infnorm-mul*: *infnorm*(*a* ∗*s x*) = *abs a* ∗ *infnorm x*
proof−
{assume *a0*: *a* = *0* **hence** *?thesis* **by** (*simp add*: *infnorm-0*) **}**
moreover
{assume *a0*: *a* ≠ *0*
from *a0* **have** *th*: (*1*/*a*) ∗*s* (*a* ∗*s x*) = *x*
by (*simp add*: *vector-smult-assoc*)
from *a0* **have** *ap*: |*a*| > *0* **by** *arith*
from *infnorm-mul-lemma*[*of 1*/*a a* ∗*s x*]
have *infnorm x* ≤ *1*/|*a*| ∗ *infnorm* (*a*∗*s x*)
unfolding *th* **by** *simp*
with *ap* **have** |*a*| ∗ *infnorm x* ≤ |*a*| ∗ (*1*/|*a*| ∗ *infnorm* (*a* ∗*s x*)) **by** (*simp add*: *field-simps*)
then have |*a*| ∗ *infnorm x* ≤ *infnorm* (*a*∗*s x*)
using *ap* **by** (*simp add*: *field-simps*)
with *infnorm-mul-lemma*[*of a x*] **have** *?thesis* **by** *arith* **}**
ultimately show *?thesis* **by** *blast*
qed

lemma *infnorm-pos-lt*: *infnorm x* > *0* ⟷ *x* ≠ *0*
using *infnorm-pos-le*[*of x*] *infnorm-eq-0*[*of x*] **by** *arith*

Prove that it differs only up to a bound from Euclidean norm.

lemma *infnorm-le-norm*: *infnorm x* ≤ *norm x*
unfolding *infnorm-def Sup-finite-le-iff*[*OF infnorm-set-lemma*]
unfolding *infnorm-set-image ball-simps*
by (*metis component-le-norm*)
lemma *card-enum*: *card* {*1* .. *n*} = *n* **by** *auto*
lemma *norm-le-infnorm*: *norm*(*x*) <= *sqrt*(*real CARD*('*n*)) ∗ *infnorm*(*x*::*real* ^'*n*)
proof−
let *?d* = *CARD*('*n*)
have *real ?d* ≥ *0* **by** *simp*
hence *d2*: (*sqrt* (*real ?d*))^*2* = *real ?d*
by (*auto intro*: *real-sqrt-pow2*)
have *th*: *sqrt* (*real ?d*) ∗ *infnorm x* ≥ *0*
by (*simp add*: *zero-le-mult-iff infnorm-pos-le*)
have *th1*: *x* · *x* ≤ (*sqrt* (*real ?d*) ∗ *infnorm x*)^*2*
unfolding *power-mult-distrib d2*

unfolding *real-of-nat-def inner-vector-def*
apply (*subst power2-abs*[*symmetric*])
apply (*rule setsum-bounded*)
apply(*auto simp add*: *power2-eq-square*[*symmetric*])
apply (*subst power2-abs*[*symmetric*])
apply (*rule power-mono*)
unfolding *infnorm-def Sup-finite-ge-iff*[*OF infnorm-set-lemma*]
unfolding *infnorm-set-image bex-simps* **apply**(*rule-tac x*=*i* **in** *exI*) **by** *auto*
from *real-le-lsqrt*[*OF inner-ge-zero th th1*]
show *?thesis* **unfolding** *norm-eq-sqrt-inner id-def* **.**
qed

Equality in Cauchy-Schwarz and triangle inequalities.

lemma *norm-cauchy-schwarz-eq*: $x \cdot y = norm\ x * norm\ y \longleftrightarrow norm\ x *_R y = norm\ y *_R x$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof−
{assume *h*: $x = 0$
hence *?thesis* **by** *simp***}**
moreover
{assume *h*: $y = 0$
hence *?thesis* **by** *simp***}**
moreover
{assume *x*: $x \neq 0$ **and** *y*: $y \neq 0$
from *inner-eq-zero-iff*[*of norm y* $*_R$ *x* − *norm x* $*_R$ *y*]
have *?rhs* $\longleftrightarrow$ $(norm\ y * (norm\ y * norm\ x * norm\ x - norm\ x * (x \cdot y)) - norm\ x * (norm\ y * (y \cdot x) - norm\ x * norm\ y * norm\ y) = 0)$
using *x y*
unfolding *inner-simps smult-conv-scaleR*
unfolding *power2-norm-eq-inner*[*symmetric*] *power2-eq-square diff-eq-0-iff-eq*
apply (*simp add*: *inner-commute*)
apply (*simp add*: *field-simps*) **by** *metis*
also have ... $\longleftrightarrow$ $(2 * norm\ x * norm\ y * (norm\ x * norm\ y - x \cdot y) = 0)$
using *x y*
by (*simp add*: *field-simps inner-commute*)
also have ... $\longleftrightarrow$ *?lhs* **using** *x y*
apply *simp*
by *metis*
finally have *?thesis* **by** *blast***}**
ultimately show *?thesis* **by** *blast*
qed

lemma *norm-cauchy-schwarz-abs-eq*:
shows $abs(x \cdot y) = norm\ x * norm\ y \longleftrightarrow$
$norm\ x *_R y = norm\ y *_R x \vee norm(x) *_R y = - norm\ y *_R x$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof−
have *th*: $\bigwedge(x::real)\ a.\ a \geq 0 \Longrightarrow abs\ x = a \longleftrightarrow x = a \vee x = - a$ **by** *arith*
have *?rhs* $\longleftrightarrow$ $norm\ x *_R y = norm\ y *_R x \vee norm\ (- x) *_R y = norm\ y *_R (- x)$

by *simp*
also have ... $\longleftrightarrow$ $(x \cdot y = norm\ x * norm\ y \vee$
$(-x) \cdot y = norm\ x * norm\ y)$
unfolding *norm-cauchy-schwarz-eq*[*symmetric*]
unfolding *norm-minus-cancel norm-scaleR* **..**
also have ... $\longleftrightarrow$ *?lhs*
unfolding *th*[*OF mult-nonneg-nonneg*, *OF norm-ge-zero*[*of x*] *norm-ge-zero*[*of y*]] *inner-simps* **by** *auto*
finally show *?thesis* **..**
qed

lemma *norm-triangle-eq*:
fixes *x y* :: *'a*::*real-inner*
shows $norm(x + y) = norm\ x + norm\ y \longleftrightarrow norm\ x *_R y = norm\ y *_R x$
proof−
{assume *x*: $x = 0 \vee y = 0$
hence *?thesis* **by** (*cases x=0*, *simp-all*)**}**
moreover
{assume *x*: $x \neq 0$ **and** *y*: $y \neq 0$
hence $norm\ x \neq 0$ $norm\ y \neq 0$
by *simp-all*
hence *n*: $norm\ x > 0$ $norm\ y > 0$
using *norm-ge-zero*[*of x*] *norm-ge-zero*[*of y*]
by *arith*+
have *th*: $\bigwedge(a::real)\ b\ c.\ a + b + c \neq 0 ==> (a = b + c \longleftrightarrow a\hat{}2 = (b + c)\hat{}2)$ **by** *algebra*
have $norm(x + y) = norm\ x + norm\ y \longleftrightarrow norm(x + y)\hat{}\ 2 = (norm\ x + norm\ y)\ \hat{}2$
apply (*rule th*) **using** *n norm-ge-zero*[*of x + y*]
by *arith*
also have ... $\longleftrightarrow$ $norm\ x *_R y = norm\ y *_R x$
unfolding *norm-cauchy-schwarz-eq*[*symmetric*]
unfolding *power2-norm-eq-inner inner-simps*
by (*simp add*: *power2-norm-eq-inner*[*symmetric*] *power2-eq-square inner-commute field-simps*)
finally have *?thesis* **.}**
ultimately show *?thesis* **by** *blast*
qed

13.17 Collinearity

definition
collinear :: *'a*::*real-vector set* ⇒ *bool* **where**
$collinear\ S \longleftrightarrow (\exists u.\ \forall x \in S.\ \forall\ y \in S.\ \exists c.\ x - y = c *_R u)$

lemma *collinear-empty*: *collinear* {} **by** (*simp add*: *collinear-def*)

lemma *collinear-sing*: *collinear* {*x*}
by (*simp add*: *collinear-def*)

lemma *collinear-2*: *collinear* {*x*, *y*}
apply (*simp add*: *collinear-def*)
apply (*rule exI*[**where** *x*=*x* − *y*])
apply *auto*
apply (*rule exI*[**where** *x*=*1*], *simp*)
apply (*rule exI*[**where** *x*=− *1*], *simp*)
done

lemma *collinear-lemma*: *collinear* {*0*,*x*,*y*} ⟷ *x* = *0* ∨ *y* = *0* ∨ (∃ *c*. *y* = *c* ∗$_R$ *x*) (**is** *?lhs* ⟷ *?rhs*)
proof−
{assume *x*=*0* ∨ *y* = *0* **hence** *?thesis*
by (*cases x* = *0*, *simp-all add*: *collinear-2 insert-commute*)**}**
moreover
{assume *x*: *x* ≠ *0* **and** *y*: *y* ≠ *0*
{assume *h*: *?lhs*
then obtain *u* **where** *u*: ∀ *x*∈ {*0*,*x*,*y*}. ∀ *y*∈ {*0*,*x*,*y*}. ∃ *c*. *x* − *y* = *c* ∗$_R$ *u*
unfolding *collinear-def* **by** *blast*
from *u*[*rule-format*, *of x 0*] *u*[*rule-format*, *of y 0*]
obtain *cx* **and** *cy* **where**
cx: *x* = *cx* ∗$_R$ *u* **and** *cy*: *y* = *cy* ∗$_R$ *u*
by *auto*
from *cx x* **have** *cx0*: *cx* ≠ *0* **by** *auto*
from *cy y* **have** *cy0*: *cy* ≠ *0* **by** *auto*
let *?d* = *cy* / *cx*
from *cx cy cx0* **have** *y* = *?d* ∗$_R$ *x*
by (*simp add*: *vector-smult-assoc*)
hence *?rhs* **using** *x y* **by** *blast***}**
moreover
{assume *h*: *?rhs*
then obtain *c* **where** *c*: *y* = *c* ∗$_R$ *x* **using** *x y* **by** *blast*
have *?lhs* **unfolding** *collinear-def c*
apply (*rule exI*[**where** *x*=*x*])
apply *auto*
apply (*rule exI*[**where** *x*=− *1*], *simp*)
apply (*rule exI*[**where** *x*= −*c*], *simp*)
apply (*rule exI*[**where** *x*=*1*], *simp*)
apply (*rule exI*[**where** *x*=*1* − *c*], *simp add*: *scaleR-left-diff-distrib*)
apply (*rule exI*[**where** *x*=*c* − *1*], *simp add*: *scaleR-left-diff-distrib*)
done}
ultimately have *?thesis* **by** *blast***}**
ultimately show *?thesis* **by** *blast*
qed

lemma *norm-cauchy-schwarz-equal*:
shows *abs*(*x* · *y*) = *norm x* ∗ *norm y* ⟷ *collinear* {*0*,*x*,*y*}
unfolding *norm-cauchy-schwarz-abs-eq*
apply (*cases x*=*0*, *simp-all add*: *collinear-2*)

apply (*cases y=0*, *simp-all add*: *collinear-2 insert-commute*)
unfolding *collinear-lemma*
apply *simp*
apply (*subgoal-tac norm x* $\neq$ *0*)
apply (*subgoal-tac norm y* $\neq$ *0*)
apply (*rule iffI*)
apply (*cases norm x* $*_R$ *y* = *norm y* $*_R$ *x*)
apply (*rule exI*[**where** *x*=(*1*/*norm x*) * *norm y*])
apply (*drule sym*)
unfolding *scaleR-scaleR*[*symmetric*]
apply (*simp add*: *vector-smult-assoc field-simps*)
apply (*rule exI*[**where** *x*=(*1*/*norm x*) * − *norm y*])
apply *clarify*
apply (*drule sym*)
unfolding *scaleR-scaleR*[*symmetric*]
apply (*simp add*: *vector-smult-assoc field-simps*)
apply (*erule exE*)
apply (*erule ssubst*)
unfolding *scaleR-scaleR*
unfolding *norm-scaleR*
apply (*subgoal-tac norm x* * *c* = |*c*| * *norm x* ∨ *norm x* * *c* = − |*c*| * *norm x*)
apply (*case-tac c* <= *0*, *simp add*: *field-simps*)
apply (*simp add*: *field-simps*)
apply (*case-tac c* <= *0*, *simp add*: *field-simps*)
apply (*simp add*: *field-simps*)
apply *simp*
apply *simp*
done

end

14 Permutations: Permutations, both general and specifically on finite sets.

theory *Permutations*
imports *Parity Fact*
begin

definition *permutes* (**infixr** *permutes 41*) **where**
(*p permutes S*) ⟷ (∀ *x*. *x* ∉ *S* ⟶ *p x* = *x*) ∧ (∀ *y*. ∃!*x*. *p x* = *y*)

lemma *swapid-sym*: *Fun.swap a b id* = *Fun.swap b a id*
by (*auto simp add*: *expand-fun-eq swap-def fun-upd-def*)

lemma *swap-id-refl*: *Fun.swap a a id = id* **by** *simp*
lemma *swap-id-sym*: *Fun.swap a b id = Fun.swap b a id*
by (*rule ext*, *simp add*: *swap-def*)
lemma *swap-id-idempotent*[*simp*]: *Fun.swap a b id o Fun.swap a b id = id*
by (*rule ext*, *auto simp add*: *swap-def*)

lemma *inv-unique-comp*: **assumes** *fg*: *f o g = id* **and** *gf*: *g o f = id*
shows *inv f = g*
using *fg gf inv-equality*[*of g f*] **by** (*auto simp add*: *expand-fun-eq*)

lemma *inverse-swap-id*: *inv* (*Fun.swap a b id*) = *Fun.swap a b id*
by (*rule inv-unique-comp*, *simp-all*)

lemma *swap-id-eq*: *Fun.swap a b id x* = (*if x = a then b else if x = b then a else x*)
by (*simp add*: *swap-def*)

lemma *permutes-in-image*: *p permutes S* ⟹ *p x* ∈ *S* ⟷ *x* ∈ *S*
unfolding *permutes-def* **by** *metis*

lemma *permutes-image*: **assumes** *pS*: *p permutes S* **shows** *p* ' *S = S*
using *pS*
unfolding *permutes-def*
apply −
apply (*rule set-ext*)
apply (*simp add*: *image-iff*)
apply *metis*
done

lemma *permutes-inj*: *p permutes S* ==> *inj p*
unfolding *permutes-def inj-on-def* **by** *blast*

lemma *permutes-surj*: *p permutes s* ==> *surj p*
unfolding *permutes-def surj-def* **by** *metis*

lemma *permutes-inv-o*: **assumes** *pS*: *p permutes S*
shows *p o inv p = id*
and *inv p o p = id*
using *permutes-inj*[*OF pS*] *permutes-surj*[*OF pS*]
unfolding *inj-iff*[*symmetric*] *surj-iff*[*symmetric*] **by** *blast+*

lemma *permutes-inverses*:
fixes *p* :: *'a* ⇒ *'a*
assumes *pS*: *p permutes S*

shows $p\ (inv\ p\ x) = x$
and $inv\ p\ (p\ x) = x$
using *permutes-inv-o*[*OF pS*, *unfolded expand-fun-eq o-def*] **by** *auto*

lemma *permutes-subset*: p *permutes* $S \Longrightarrow S \subseteq T$ ==> p *permutes* T
unfolding *permutes-def* **by** *blast*

lemma *permutes-empty*[*simp*]: p *permutes* $\{\} \longleftrightarrow p = id$
unfolding *expand-fun-eq permutes-def* **apply** *simp* **by** *metis*

lemma *permutes-sing*[*simp*]: p *permutes* $\{a\} \longleftrightarrow p = id$
unfolding *expand-fun-eq permutes-def* **apply** *simp* **by** *metis*

lemma *permutes-univ*: p *permutes UNIV* $\longleftrightarrow (\forall y.\ \exists!x.\ p\ x = y)$
unfolding *permutes-def* **by** *simp*

lemma *permutes-inv-eq*: p *permutes* S ==> $inv\ p\ y = x \longleftrightarrow p\ x = y$
unfolding *permutes-def inv-def* **apply** *auto*
apply (*erule allE*[**where** x=y])
apply (*erule allE*[**where** x=y])
apply (*rule someI-ex*) **apply** *blast*
apply (*rule some1-equality*)
apply *blast*
apply *blast*
done

lemma *permutes-swap-id*: $a \in S \Longrightarrow b \in S$ ==> *Fun.swap a b id permutes* S
unfolding *permutes-def swap-def fun-upd-def* **by** *auto metis*

lemma *permutes-superset*:
p *permutes* $S \Longrightarrow (\forall x \in S - T.\ p\ x = x) \Longrightarrow p$ *permutes* T
by (*simp add*: *Ball-def permutes-def*) *metis*

lemma *permutes-id*: *id permutes* S **unfolding** *permutes-def* **by** *simp*

lemma *permutes-compose*: p *permutes* $S \Longrightarrow q$ *permutes* S ==> $q\ o\ p$ *permutes* S
unfolding *permutes-def o-def* **by** *metis*

lemma *permutes-inv*: **assumes** *pS*: p *permutes* S **shows** *inv p permutes* S
using *pS* **unfolding** *permutes-def permutes-inv-eq*[*OF pS*] **by** *metis*

lemma *permutes-inv-inv*: **assumes** *pS*: p *permutes* S **shows** $inv\ (inv\ p) = p$
unfolding *expand-fun-eq permutes-inv-eq*[*OF pS*] *permutes-inv-eq*[*OF permutes-inv*[*OF pS*]]

by *blast*

lemma *permutes-insert-lemma*:
assumes *pS*: *p permutes* (*insert a S*)
shows *Fun.swap a* (*p a*) *id o p permutes S*
apply (*rule permutes-superset*[**where** *S* = *insert a S*])
apply (*rule permutes-compose*[*OF pS*])
apply (*rule permutes-swap-id*, *simp*)
using *permutes-in-image*[*OF pS*, *of a*] **apply** *simp*
apply (*auto simp add*: *Ball-def swap-def*)
done

lemma *permutes-insert*: {*p*. *p permutes* (*insert a S*)} =
(λ(*b*,*p*). *Fun.swap a b id o p*) ' {(*b*,*p*). *b* ∈ *insert a S* ∧ *p* ∈ {*p*. *p permutes S*}}
proof−

{fix *p*
{assume *pS*: *p permutes insert a S*
let *?b* = *p a*
let *?q* = *Fun.swap a* (*p a*) *id o p*
have *th0*: *p* = *Fun.swap a ?b id o ?q* **unfolding** *expand-fun-eq o-assoc* **by** *simp*
have *th1*: *?b* ∈ *insert a S* **unfolding** *permutes-in-image*[*OF pS*] **by** *simp*
from *permutes-insert-lemma*[*OF pS*] *th0 th1*
have ∃ *b q*. *p* = *Fun.swap a b id o q* ∧ *b* ∈ *insert a S* ∧ *q permutes S* **by** *blast*}
moreover
{fix *b q* **assume** *bq*: *p* = *Fun.swap a b id o q b* ∈ *insert a S q permutes S*
from *permutes-subset*[*OF bq*(*3*), *of insert a S*]
have *qS*: *q permutes insert a S* **by** *auto*
have *aS*: *a* ∈ *insert a S* **by** *simp*
from *bq*(*1*) *permutes-compose*[*OF qS permutes-swap-id*[*OF aS bq*(*2*)]]
have *p permutes insert a S* **by** *simp* **}**
ultimately have *p permutes insert a S* ⟷ (∃ *b q*. *p* = *Fun.swap a b id o q* ∧ *b* ∈ *insert a S* ∧ *q permutes S*) **by** *blast*}
thus *?thesis* **by** *auto*
qed

lemma *card-permutations*: **assumes** *Sn*: *card S* = *n* **and** *fS*: *finite S*
shows *card* {*p*. *p permutes S*} = *fact n*
using *fS Sn* **proof** (*induct arbitrary*: *n*)
case *empty* **thus** *?case* **by** *simp*
next
case (*insert x F*)

```
 { fix n assume H0: card (insert x F) = n
  let ?xF = {p. p permutes insert x F}
  let ?pF = {p. p permutes F}
  let ?pF' = {(b, p). b ∈ insert x F ∧ p ∈ ?pF}
  let ?g = (λ(b, p). Fun.swap x b id ∘ p)
  from permutes-insert[of x F]
  have xfgpF': ?xF = ?g ` ?pF' .
  have Fs: card F = n − 1 using ‹x ∉ F› H0 ‹finite F› by auto
  from insert.hyps Fs have pFs: card ?pF = fact (n − 1) using ‹finite F› by
auto
 moreover hence finite ?pF using fact-gt-zero-nat by (auto intro: card-ge-0-finite)
  ultimately have pF'f: finite ?pF' using H0 ‹finite F›
   apply (simp only: Collect-split Collect-mem-eq)
   apply (rule finite-cartesian-product)
   apply simp-all
   done

  have ginj: inj-on ?g ?pF'
  proof−
   {
    fix b p c q assume bp: (b,p) ∈ ?pF' and cq: (c,q) ∈ ?pF'
     and eq: ?g (b,p) = ?g (c,q)
    from bp cq have ths: b ∈ insert x F c ∈ insert x F x ∈ insert x F p permutes
F q permutes F by auto
    from ths(4) ‹x ∉ F› eq have b = ?g (b,p) x unfolding permutes-def
     by (auto simp add: swap-def fun-upd-def expand-fun-eq)
    also have ... = ?g (c,q) x using ths(5) ‹x ∉ F› eq
     by (auto simp add: swap-def fun-upd-def expand-fun-eq)
    also have ... = cusing ths(5) ‹x ∉ F› unfolding permutes-def
     by (auto simp add: swap-def fun-upd-def expand-fun-eq)
    finally have bc: b = c .
    hence Fun.swap x b id = Fun.swap x c id by simp
    with eq have Fun.swap x b id o p = Fun.swap x b id o q by simp
     hence Fun.swap x b id o (Fun.swap x b id o p) = Fun.swap x b id o
(Fun.swap x b id o q) by simp
    hence p = q by (simp add: o-assoc)
    with bc have (b,p) = (c,q) by simp
   }
   thus ?thesis  unfolding inj-on-def by blast
  qed
  from ‹x ∉ F› H0 have n0: n ≠ 0  using ‹finite F› by auto
  hence ∃ m. n = Suc m by arith
  then obtain m where n[simp]: n = Suc m by blast
  from pFs H0 have xFc: card ?xF = fact n
   unfolding xfgpF' card-image[OF ginj] using ‹finite F› ‹finite ?pF›
   apply (simp only: Collect-split Collect-mem-eq card-cartesian-product)
   by simp
 from finite-imageI[OF pF'f, of ?g] have xFf: finite ?xF unfolding xfgpF' by
simp
```

```
    have card ?xF = fact n
      using xFf xFc unfolding xFf by blast
  }
  thus ?case using insert by simp
qed

lemma finite-permutations: assumes fS: finite S shows finite {p. p permutes S}
  using card-permutations[OF refl fS] fact-gt-zero-nat
  by (auto intro: card-ge-0-finite)




lemma (in ab-semigroup-mult) fold-image-permute: assumes fS: finite S and pS:
p permutes S
  shows fold-image times f z S = fold-image times (f o p) z S
   using fold-image-reindex[OF fS subset-inj-on[OF permutes-inj[OF pS], of S,
simplified], of f z]
  unfolding permutes-image[OF pS] .
lemma (in ab-semigroup-add) fold-image-permute: assumes fS: finite S and pS:
p permutes S
  shows fold-image plus f z S = fold-image plus (f o p) z S
proof−
 interpret ab-semigroup-mult plus apply unfold-locales apply (simp add: add-assoc)
    apply (simp add: add-commute) done
  from fold-image-reindex[OF fS subset-inj-on[OF permutes-inj[OF pS], of S, sim-
plified], of f z]
  show ?thesis
  unfolding permutes-image[OF pS] .
qed

lemma setsum-permute: assumes pS: p permutes S
  shows setsum f S = setsum (f o p) S
  unfolding setsum-def using fold-image-permute[of S p f 0] pS by clarsimp

lemma setsum-permute-natseg:assumes pS: p permutes {m .. n}
  shows setsum f {m .. n} = setsum (f o p) {m .. n}
  using setsum-permute[OF pS, of f ] pS by blast

lemma setprod-permute: assumes pS: p permutes S
  shows setprod f S = setprod (f o p) S
  unfolding setprod-def
  using ab-semigroup-mult-class.fold-image-permute[of S p f 1] pS by clarsimp

lemma setprod-permute-natseg:assumes pS: p permutes {m .. n}
  shows setprod f {m .. n} = setprod (f o p) {m .. n}
  using setprod-permute[OF pS, of f ] pS by blast
```

lemma *swap-id-common*: $a \neq c \Longrightarrow b \neq c \Longrightarrow$ *Fun.swap a b id o Fun.swap a c id = Fun.swap b c id o Fun.swap a b id* **by** (*simp add*: *expand-fun-eq swap-def*)

lemma *swap-id-common'*: $\sim(a = b) \Longrightarrow \sim(a = c) \Longrightarrow$ *Fun.swap a c id o Fun.swap b c id = Fun.swap b c id o Fun.swap a b id* **by** (*simp add*: *expand-fun-eq swap-def*)

lemma *swap-id-independent*: $\sim(a = c) \Longrightarrow \sim(a = d) \Longrightarrow \sim(b = c) \Longrightarrow \sim(b = d)$ $\Longrightarrow$ *Fun.swap a b id o Fun.swap c d id = Fun.swap c d id o Fun.swap a b id*
by (*simp add*: *swap-def expand-fun-eq*)

inductive *swapidseq* :: *nat* $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool* **where**
id[*simp*]: *swapidseq 0 id*
| *comp-Suc*: *swapidseq n p* $\Longrightarrow$ *a* $\neq$ *b* $\Longrightarrow$ *swapidseq* (*Suc n*) (*Fun.swap a b id o p*)

declare *id*[*unfolded id-def*, *simp*]
definition *permutation p* $\longleftrightarrow$ ($\exists$ *n*. *swapidseq n p*)

lemma *permutation-id*[*simp*]: *permutation id***unfolding** *permutation-def*
by (*rule exI*[**where** *x=0*], *simp*)
declare *permutation-id*[*unfolded id-def*, *simp*]

lemma *swapidseq-swap*: *swapidseq* (*if a = b then 0 else 1*) (*Fun.swap a b id*)
apply *clarsimp*
using *comp-Suc*[*of 0 id a b*] **by** *simp*

lemma *permutation-swap-id*: *permutation* (*Fun.swap a b id*)
apply (*cases a=b*, *simp-all*)
unfolding *permutation-def* **using** *swapidseq-swap*[*of a b*] **by** *blast*

lemma *swapidseq-comp-add*: *swapidseq n p* $\Longrightarrow$ *swapidseq m q* $\Longrightarrow$ *swapidseq* (*n + m*) (*p o q*)
proof (*induct n p arbitrary*: *m q rule*: *swapidseq.induct*)
case (*id m q*) **thus** *?case* **by** *simp*
next
case (*comp-Suc n p a b m q*)

have *th*: *Suc n + m = Suc (n + m)* **by** *arith*
show *?case* **unfolding** *th o-assoc*[*symmetric*]
apply (*rule swapidseq.comp-Suc*) **using** *comp-Suc.hyps*(*2*)[*OF comp-Suc.prems*] *comp-Suc.hyps*(*3*) **by** *blast+*
qed

lemma *permutation-compose*: *permutation p ⟹ permutation q ==> permutation*(*p o q*)
unfolding *permutation-def* **using** *swapidseq-comp-add*[*of - p - q*] **by** *metis*

lemma *swapidseq-endswap*: *swapidseq n p ⟹ a ≠ b ==> swapidseq* (*Suc n*) (*p o Fun.swap a b id*)
apply (*induct n p rule*: *swapidseq.induct*)
using *swapidseq-swap*[*of a b*]
by (*auto simp add*: *o-assoc*[*symmetric*] *intro*: *swapidseq.comp-Suc*)

lemma *swapidseq-inverse-exists*: *swapidseq n p ==> ∃ q. swapidseq n q ∧ p o q = id ∧ q o p = id*
proof(*induct n p rule*: *swapidseq.induct*)
case *id* **thus** *?case* **by** (*rule exI*[**where** *x*=*id*], *simp*)
next
case (*comp-Suc n p a b*)
from *comp-Suc.hyps* **obtain** *q* **where** *q*: *swapidseq n q p ∘ q = id q ∘ p = id* **by** *blast*
let *?q = q o Fun.swap a b id*
note *H = comp-Suc.hyps*
from *swapidseq-swap*[*of a b*] *H*(*3*) **have** *th0*: *swapidseq 1* (*Fun.swap a b id*) **by** *simp*
from *swapidseq-comp-add*[*OF q*(*1*) *th0*] **have** *th1*:*swapidseq* (*Suc n*) *?q* **by** *simp*
have *Fun.swap a b id o p o ?q = Fun.swap a b id o* (*p o q*) *o Fun.swap a b id* **by** (*simp add*: *o-assoc*)
also have *... = id* **by** (*simp add*: *q*(*2*))
finally have *th2*: *Fun.swap a b id o p o ?q = id* **.**
have *?q ∘* (*Fun.swap a b id ∘ p*) *= q ∘* (*Fun.swap a b id o Fun.swap a b id*) *∘ p* **by** (*simp only*: *o-assoc*)
hence *?q ∘* (*Fun.swap a b id ∘ p*) *= id* **by** (*simp add*: *q*(*3*))
with *th1 th2* **show** *?case* **by** *blast*
qed

lemma *swapidseq-inverse*: **assumes** *H*: *swapidseq n p* **shows** *swapidseq n* (*inv p*)
using *swapidseq-inverse-exists*[*OF H*] *inv-unique-comp*[*of p*] **by** *auto*

lemma *permutation-inverse*: *permutation p ==> permutation* (*inv p*)
using *permutation-def swapidseq-inverse* **by** *blast*

lemma *symmetry-lemma*:$(\bigwedge a\ b\ c\ d.\ P\ a\ b\ c\ d ==> P\ a\ b\ d\ c) \Longrightarrow$
$(\bigwedge a\ b\ c\ d.\ a \neq b \Longrightarrow c \neq d \Longrightarrow (a = c \wedge b = d \vee\ a = c \wedge b \neq d \vee a \neq c \wedge b = d \vee a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d) ==> P\ a\ b\ c\ d)$
$==> (\bigwedge a\ b\ c\ d.\ a \neq b \longrightarrow c \neq d \longrightarrow\ P\ a\ b\ c\ d)$ **by** *metis*

lemma *swap-general*: $a \neq b \Longrightarrow c \neq d \Longrightarrow$ *Fun.swap a b id o Fun.swap c d id = id* $\vee$
$(\exists x\ y\ z.\ x \neq a \wedge y \neq a \wedge z \neq a \wedge x \neq y \wedge$ *Fun.swap a b id o Fun.swap c d id = Fun.swap x y id o Fun.swap a z id*)
proof−
assume *H*: *a≠b c≠d*
have $a \neq b \longrightarrow c \neq d \longrightarrow$
(*Fun.swap a b id o Fun.swap c d id = id* $\vee$
$(\exists x\ y\ z.\ x \neq a \wedge y \neq a \wedge z \neq a \wedge x \neq y \wedge$ *Fun.swap a b id o Fun.swap c d id = Fun.swap x y id o Fun.swap a z id*))
apply (*rule symmetry-lemma*[**where** *a=a* **and** *b=b* **and** *c=c* **and** *d=d*])
apply (*simp-all only*: *swapid-sym*)
apply (*case-tac* $a = c \wedge b = d$, *clarsimp simp only*: *swapid-sym swap-id-idempotent*)
apply (*case-tac* $a = c \wedge b \neq d$)
apply (*rule disjI2*)
apply (*rule-tac x=b* **in** *exI*)
apply (*rule-tac x=d* **in** *exI*)
apply (*rule-tac x=b* **in** *exI*)
apply (*clarsimp simp add*: *expand-fun-eq swap-def*)
apply (*case-tac* $a \neq c \wedge b = d$)
apply (*rule disjI2*)
apply (*rule-tac x=c* **in** *exI*)
apply (*rule-tac x=d* **in** *exI*)
apply (*rule-tac x=c* **in** *exI*)
apply (*clarsimp simp add*: *expand-fun-eq swap-def*)
apply (*rule disjI2*)
apply (*rule-tac x=c* **in** *exI*)
apply (*rule-tac x=d* **in** *exI*)
apply (*rule-tac x=b* **in** *exI*)
apply (*clarsimp simp add*: *expand-fun-eq swap-def*)
done
with *H* **show** *?thesis* **by** *metis*
qed

lemma *swapidseq-id-iff* [*simp*]: *swapidseq 0 p* $\longleftrightarrow$ $p = id$
using *swapidseq.cases*[*of 0 p p = id*]
by *auto*

lemma *swapidseq-cases*: *swapidseq n p* $\longleftrightarrow$ (*n=0* $\wedge$ $p = id \vee (\exists a\ b\ q\ m.\ n = Suc\ m \wedge p =$ *Fun.swap a b id o q* $\wedge$ *swapidseq m q* $\wedge$ *a≠ b*))
apply (*rule iffI*)
apply (*erule swapidseq.cases*[*of n p*])
apply *simp*

apply (*rule disjI2*)
apply (*rule-tac x*= *a* **in** *exI*)
apply (*rule-tac x*= *b* **in** *exI*)
apply (*rule-tac x*= *pa* **in** *exI*)
apply (*rule-tac x*= *na* **in** *exI*)
apply *simp*
apply *auto*
apply (*rule comp-Suc*, *simp-all*)
done
lemma *fixing-swapidseq-decrease*:
assumes *spn*: *swapidseq n p* **and** *ab*: *a*≠*b* **and** *pa*: (*Fun.swap a b id o p*) *a* = *a*
shows *n* ≠ *0* ∧ *swapidseq* (*n* − *1*) (*Fun.swap a b id o p*)
using *spn ab pa*
proof(*induct n arbitrary*: *p a b*)
case *0* **thus** *?case* **by** (*auto simp add*: *swap-def fun-upd-def*)
next
case (*Suc n p a b*)
from *Suc.prems*(*1*) *swapidseq-cases*[*of Suc n p*] **obtain**
c d q m **where** *cdqm*: *Suc n* = *Suc m p* = *Fun.swap c d id o q swapidseq m q c* ≠ *d n* = *m*
by *auto*
{assume *H*: *Fun.swap a b id o Fun.swap c d id* = *id*

have *?case* **apply** (*simp only*: *cdqm o-assoc H*)
by (*simp add*: *cdqm*)**}**
moreover
{ fix *x y z*
assume *H*: *x*≠*a y*≠*a z* ≠*a x* ≠*y*
Fun.swap a b id o Fun.swap c d id = *Fun.swap x y id o Fun.swap a z id*
from *H* **have** *az*: *a* ≠ *z* **by** *simp*

{fix *h* **have** (*Fun.swap x y id o h*) *a* = *a* ⟷ *h a* = *a*
using *H* **by** (*simp add*: *swap-def*)**}**
note *th3* = *this*
from *cdqm*(*2*) **have** *Fun.swap a b id o p* = *Fun.swap a b id o* (*Fun.swap c d id o q*) **by** *simp*
hence *Fun.swap a b id o p* = *Fun.swap x y id o* (*Fun.swap a z id o q*) **by** (*simp add*: *o-assoc H*)
hence (*Fun.swap a b id o p*) *a* = (*Fun.swap x y id o* (*Fun.swap a z id o q*)) *a* **by** *simp*
hence (*Fun.swap x y id o* (*Fun.swap a z id o q*)) *a* = *a* **unfolding** *Suc* **by** *metis*
hence *th1*: (*Fun.swap a z id o q*) *a* = *a* **unfolding** *th3* **.**
from *Suc.hyps*[*OF cdqm*(*3*)[*unfolded cdqm*(*5*)[*symmetric*]] *az th1*]
have *th2*: *swapidseq* (*n* − *1*) (*Fun.swap a z id o q*) *n* ≠ *0* **by** *blast*+
have *th*: *Suc n* − *1* = *Suc* (*n* − *1*) **using** *th2*(*2*) **by** *auto*
have *?case* **unfolding** *cdqm*(*2*) *H o-assoc th*
apply (*simp only*: *Suc-not-Zero simp-thms o-assoc*[*symmetric*])
apply (*rule comp-Suc*)

using *th2 H* **apply** *blast+*
done}
ultimately show *?case* **using** *swap-general*[*OF Suc.prems*(*2*) *cdqm*(*4*)] **by** *metis*
qed

lemma *swapidseq-identity-even*:
assumes *swapidseq n* (*id* :: *'a* ⇒ *'a*) **shows** *even n*
using ‹*swapidseq n id*›
proof(*induct n rule*: *nat-less-induct*)
fix *n*
assume *H*: ∀ *m*<*n*. *swapidseq m* (*id*::*'a* ⇒ *'a*) ⟶ *even m swapidseq n* (*id* :: *'a* ⇒ *'a*)
{assume *n* = *0* **hence** *even n* **by** *arith***}**
moreover
{fix *a b* :: *'a* **and** *q m*
assume *h*: *n* = *Suc m* (*id* :: *'a* ⇒ *'a*) = *Fun.swap a b id* ∘ *q swapidseq m q a* ≠ *b*
from *fixing-swapidseq-decrease*[*OF h*(*3*,*4*), *unfolded h*(*2*)[*symmetric*]]
have *m*: *m* ≠ *0 swapidseq* (*m* − *1*) (*id* :: *'a* ⇒ *'a*) **by** *auto*
from *h m* **have** *mn*: *m* − *1* < *n* **by** *arith*
from *H*(*1*)[*rule-format*, *OF mn m*(*2*)] *h*(*1*) *m*(*1*) **have** *even n* **apply** *arith*
done}
ultimately show *even n* **using** *H*(*2*)[*unfolded swapidseq-cases*[*of n id*]] **by** *auto*
qed

definition *evenperm p* = *even* (*SOME n. swapidseq n p*)

lemma *swapidseq-even-even*: **assumes**
m: *swapidseq m p* **and** *n*: *swapidseq n p*
shows *even m* ⟷ *even n*
proof−
from *swapidseq-inverse-exists*[*OF n*]
obtain *q* **where** *q*: *swapidseq n q p* ∘ *q* = *id q* ∘ *p* = *id* **by** *blast*

from *swapidseq-identity-even*[*OF swapidseq-comp-add*[*OF m q*(*1*), *unfolded q*]]
show *?thesis* **by** *arith*
qed

lemma *evenperm-unique*: **assumes** *p*: *swapidseq n p* **and** *n*:*even n* = *b*
shows *evenperm p* = *b*
unfolding *n*[*symmetric*] *evenperm-def*
apply (*rule swapidseq-even-even*[**where** *p* = *p*])
apply (*rule someI*[**where** *x* = *n*])
using *p* **by** *blast+*

lemma *evenperm-id*[*simp*]: *evenperm id = True*
apply (*rule evenperm-unique*[**where** *n = 0*]) **by** *simp-all*

lemma *evenperm-swap*: *evenperm* (*Fun.swap a b id*) = (*a = b*)
apply (*rule evenperm-unique*[**where** *n=if a = b then 0 else 1*])
by (*simp-all add*: *swapidseq-swap*)

lemma *evenperm-comp*:
assumes *p*: *permutation p* **and** *q*:*permutation q*
shows *evenperm* (*p o q*) = (*evenperm p = evenperm q*)
proof−
from *p q* **obtain**
n m **where** *n*: *swapidseq n p* **and** *m*: *swapidseq m q*
unfolding *permutation-def* **by** *blast*
note *nm = swapidseq-comp-add*[*OF n m*]
have *th*: *even* (*n + m*) = (*even n* ⟷ *even m*) **by** *arith*
from *evenperm-unique*[*OF n refl*] *evenperm-unique*[*OF m refl*]
evenperm-unique[*OF nm th*]
show *?thesis* **by** *blast*
qed

lemma *evenperm-inv*: **assumes** *p*: *permutation p*
shows *evenperm* (*inv p*) = *evenperm p*
proof−
from *p* **obtain** *n* **where** *n*: *swapidseq n p* **unfolding** *permutation-def* **by** *blast*
from *evenperm-unique*[*OF swapidseq-inverse*[*OF n*] *evenperm-unique*[*OF n refl*, *symmetric*]]
show *?thesis* **.**
qed

lemma *bij-iff*: *bij f* ⟷ (∀ *x*. ∃!*y*. *f y = x*)
unfolding *bij-def inj-on-def surj-def*
apply *auto*
apply *metis*
apply *metis*
done

lemma *permutation-bijective*:
assumes *p*: *permutation p*
shows *bij p*

proof−
from *p* **obtain** *n* **where** *n*: *swapidseq n p* **unfolding** *permutation-def* **by** *blast*
from *swapidseq-inverse-exists*[*OF n*] **obtain** *q* **where**
q: *swapidseq n q p* ∘ *q* = *id q* ∘ *p* = *id* **by** *blast*
thus *?thesis* **unfolding** *bij-iff* **apply** (*auto simp add*: *expand-fun-eq*) **apply** *metis* **done**
qed

lemma *permutation-finite-support*: **assumes** *p*: *permutation p*
shows *finite* {*x*. *p x* ≠ *x*}
proof−
from *p* **obtain** *n* **where** *n*: *swapidseq n p* **unfolding** *permutation-def* **by** *blast*
from *n* **show** *?thesis*
proof(*induct n p rule*: *swapidseq.induct*)
case *id* **thus** *?case* **by** *simp*
next
case (*comp-Suc n p a b*)
let *?S* = *insert a* (*insert b* {*x*. *p x* ≠ *x*})
from *comp-Suc.hyps*(*2*) **have** *fS*: *finite ?S* **by** *simp*
from ‹*a* ≠ *b*› **have** *th*: {*x*. (*Fun.swap a b id o p*) *x* ≠ *x*} ⊆ *?S*
by (*auto simp add*: *swap-def*)
from *finite-subset*[*OF th fS*] **show** *?case* **.**
qed
qed

lemma *bij-inv-eq-iff*: *bij p* ==> *x* = *inv p y* ⟷ *p x* = *y*
using *surj-f-inv-f*[*of p*] *inv-f-f*[*of f*] **by** (*auto simp add*: *bij-def*)

lemma *bij-swap-comp*:
assumes *bp*: *bij p* **shows** *Fun.swap a b id o p* = *Fun.swap* (*inv p a*) (*inv p b*) *p*
using *surj-f-inv-f*[*OF bij-is-surj*[*OF bp*]]
by (*simp add*: *expand-fun-eq swap-def bij-inv-eq-iff*[*OF bp*])

lemma *bij-swap-ompose-bij*: *bij p* ⟹ *bij* (*Fun.swap a b id o p*)
proof−
assume *H*: *bij p*
show *?thesis*
unfolding *bij-swap-comp*[*OF H*] *bij-swap-iff*
using *H* **.**
qed

lemma *permutation-lemma*:
assumes *fS*: *finite S* **and** *p*: *bij p* **and** *pS*: ∀ *x*. *x*∉ *S* ⟶ *p x* = *x*
shows *permutation p*
using *fS p pS*
proof(*induct S arbitrary*: *p rule*: *finite-induct*)
case (*empty p*) **thus** *?case* **by** *simp*
next
case (*insert a F p*)

```
  let ?r = Fun.swap a (p a) id o p
  let ?q = Fun.swap a (p a) id o ?r
  have raa: ?r a = a by (simp add: swap-def)
  from bij-swap-ompose-bij[OF insert(4)]
  have br: bij ?r .

  from insert raa have th: ∀ x. x ∉ F ⟶ ?r x = x
    apply (clarsimp simp add: swap-def)
    apply (erule-tac x=x in allE)
    apply auto
    unfolding bij-iff apply metis
    done
  from insert(3)[OF br th]
  have rp: permutation ?r .
   have permutation ?q by (simp add: permutation-compose permutation-swap-id
rp)
  thus ?case by (simp add: o-assoc)
qed

lemma permutation: permutation p ⟷ bij p ∧ finite {x. p x ≠ x}
  (is ?lhs ⟷ ?b ∧ ?f)
proof
  assume p: ?lhs
  from p permutation-bijective permutation-finite-support show ?b ∧ ?f by auto
next
  assume bf: ?b ∧ ?f
  hence bf: ?f ?b by blast+
  from permutation-lemma[OF bf] show ?lhs by blast
qed

lemma permutation-inverse-works: assumes p: permutation p
  shows inv p o p = id p o inv p = id
using permutation-bijective[OF p] surj-iff bij-def inj-iff by auto

lemma permutation-inverse-compose:
  assumes p: permutation p and q: permutation q
  shows inv (p o q) = inv q o inv p
proof−
  note ps = permutation-inverse-works[OF p]
  note qs = permutation-inverse-works[OF q]
  have p o q o (inv q o inv p) = p o (q o inv q) o inv p by (simp add: o-assoc)
  also have ... = id by (simp add: ps qs)
  finally have th0: p o q o (inv q o inv p) = id .
  have inv q o inv p o (p o q) = inv q o (inv p o p) o q by (simp add: o-assoc)
  also have ... = id by (simp add: ps qs)
  finally have th1: inv q o inv p o (p o q) = id .
  from inv-unique-comp[OF th0 th1] show ?thesis .
qed
```

lemma *permutation-permutes*: *permutation p* $\longleftrightarrow$ ($\exists$ *S*. *finite S* $\wedge$ *p permutes S*)
unfolding *permutation permutes-def bij-iff* [*symmetric*]
apply (*rule iffI*, *clarify*)
apply (*rule exI*[**where** *x*={*x*. *p x* $\neq$ *x*}])
apply *simp*
apply *clarsimp*
apply (*rule-tac B*=*S* **in** *finite-subset*)
apply *auto*
done

lemma *permutes-induct*: *finite S* $\Longrightarrow$ *P id* $\Longrightarrow$ ($\bigwedge$ *a b p*. *a* $\in$ *S* $\Longrightarrow$ *b* $\in$ *S* $\Longrightarrow$ *P p* $\Longrightarrow$ *P p* $\Longrightarrow$ *permutation p* ==> *P* (*Fun.swap a b id o p*))
==> ($\bigwedge$*p*. *p permutes S* ==> *P p*)
proof(*induct S rule*: *finite-induct*)
case *empty* **thus** *?case* **by** *auto*
next
case (*insert x F p*)
let *?r* = *Fun.swap x* (*p x*) *id o p*
let *?q* = *Fun.swap x* (*p x*) *id o ?r*
have *qp*: *?q* = *p* **by** (*simp add*: *o-assoc*)
from *permutes-insert-lemma*[*OF insert.prems*(*3*)] *insert* **have** *Pr*: *P ?r* **by** *blast*
from *permutes-in-image*[*OF insert.prems*(*3*), *of x*]
have *pxF*: *p x* $\in$ *insert x F* **by** *simp*
have *xF*: *x* $\in$ *insert x F* **by** *simp*
have *rp*: *permutation ?r*
unfolding *permutation-permutes* **using** *insert.hyps*(*1*)
permutes-insert-lemma[*OF insert.prems*(*3*)] **by** *blast*
from *insert.prems*(*2*)[*OF xF pxF Pr Pr rp*]
show *?case* **unfolding** *qp* **.**
qed

definition *sign p* = (*if evenperm p then* (*1*::*int*) *else* $-$*1*)

lemma *sign-nz*: *sign p* $\neq$ *0* **by** (*simp add*: *sign-def*)
lemma *sign-id*: *sign id* = *1* **by** (*simp add*: *sign-def*)
lemma *sign-inverse*: *permutation p* ==> *sign* (*inv p*) = *sign p*
by (*simp add*: *sign-def evenperm-inv*)

lemma *sign-compose*: *permutation p* $\Longrightarrow$ *permutation q* ==> *sign* (*p o q*) = *sign*(*p*) $*$ *sign*(*q*) **by** (*simp add*: *sign-def evenperm-comp*)
lemma *sign-swap-id*: *sign* (*Fun.swap a b id*) = (*if a* = *b then 1 else* −*1*)
by (*simp add*: *sign-def evenperm-swap*)
lemma *sign-idempotent*: *sign p* $*$ *sign p* = *1* **by** (*simp add*: *sign-def*)

lemma *permutes-natset-le*:
assumes *p*: *p permutes* (*S*::'*a*::*wellorder set*) **and** *le*: $\forall\, i \in S.$ *p i* <= *i* **shows** *p* = *id*
proof−
{fix *n*
have *p n* = *n*
using *p le*
proof(*induct n arbitrary*: *S rule*: *less-induct*)
fix *n S* **assume** *H*: $\bigwedge m\ S.\ [\![m < n;\ p\ permutes\ S;\ \forall\, i{\in}S.\ p\ i \leq i]\!] \Longrightarrow p\ m = m$
p permutes S $\forall\, i \in S.$ *p i* $\leq$ *i*
{assume *n* $\notin$ *S*
with *H*(*2*) **have** *p n* = *n* **unfolding** *permutes-def* **by** *metis***}**
moreover
{assume *ns*: *n* $\in$ *S*
from *H*(*3*) *ns* **have** *p n* < *n* $\vee$ *p n* = *n* **by** *auto*
moreover{assume *h*: *p n* < *n*
from *H h* **have** *p* (*p n*) = *p n* **by** *metis*
with *permutes-inj*[*OF H*(*2*)] **have** *p n* = *n* **unfolding** *inj-on-def* **by** *blast*
with *h* **have** *False* **by** *simp***}**
ultimately have *p n* = *n* **by** *blast* **}**
ultimately show *p n* = *n* **by** *blast*
qed}
thus *?thesis* **by** (*auto simp add*: *expand-fun-eq*)
qed

lemma *permutes-natset-ge*:
assumes *p*: *p permutes* (*S*::'*a*::*wellorder set*) **and** *le*: $\forall\, i \in S.$ *p i* $\geq$ *i* **shows** *p* = *id*
proof−
{fix *i* **assume** *i*: *i* $\in$ *S*
from *i permutes-in-image*[*OF permutes-inv*[*OF p*]] **have** *inv p i* $\in$ *S* **by** *simp*
with *le* **have** *p* (*inv p i*) $\geq$ *inv p i* **by** *blast*
with *permutes-inverses*[*OF p*] **have** *i* $\geq$ *inv p i* **by** *simp***}**
then have *th*: $\forall\, i{\in}S.$ *inv p i* $\leq$ *i* **by** *blast*
from *permutes-natset-le*[*OF permutes-inv*[*OF p*] *th*]
have *inv p* = *inv id* **by** *simp*
then show *?thesis*
apply (*subst permutes-inv-inv*[*OF p*, *symmetric*])

apply (*rule inv-unique-comp*)
apply *simp-all*
done
qed

lemma *image-inverse-permutations*: {*inv p* |*p*. *p permutes S*} = {*p*. *p permutes S*}
apply (*rule set-ext*)
apply *auto*
using *permutes-inv-inv permutes-inv* **apply** *auto*
apply (*rule-tac x*=*inv x* **in** *exI*)
apply *auto*
done

lemma *image-compose-permutations-left*:
assumes *q*: *q permutes S* **shows** {*q o p* | *p*. *p permutes S*} = {*p* . *p permutes S*}
apply (*rule set-ext*)
apply *auto*
apply (*rule permutes-compose*)
using *q* **apply** *auto*
apply (*rule-tac x* = *inv q o x* **in** *exI*)
by (*simp add*: *o-assoc permutes-inv permutes-compose permutes-inv-o*)

lemma *image-compose-permutations-right*:
assumes *q*: *q permutes S*
shows {*p o q* | *p*. *p permutes S*} = {*p* . *p permutes S*}
apply (*rule set-ext*)
apply *auto*
apply (*rule permutes-compose*)
using *q* **apply** *auto*
apply (*rule-tac x* = *x o inv q* **in** *exI*)
by (*simp add*: *o-assoc permutes-inv permutes-compose permutes-inv-o o-assoc*[*symmetric*])

lemma *permutes-in-seg*: *p permutes* {*1* ..*n*} ⟹ *i* ∈ {*1*..*n*} ==> *1* <= *p i* ∧ *p i* <= *n*

apply (*simp add*: *permutes-def*)
apply *metis*
done

term *setsum*
lemma *setsum-permutations-inverse*: *setsum f* {*p*. *p permutes S*} = *setsum* (λ*p*. *f*(*inv p*)) {*p*. *p permutes S*} (**is** *?lhs* = *?rhs*)
proof−
let *?S* = {*p* . *p permutes S*}
have *th0*: *inj-on inv ?S*
proof(*auto simp add*: *inj-on-def*)
fix *q r*

```
  assume q: q permutes S and r: r permutes S and qr: inv q = inv r
  hence inv (inv q) = inv (inv r) by simp
  with permutes-inv-inv[OF q] permutes-inv-inv[OF r]
  show q = r by metis
qed
  have th1: inv ' ?S = ?S using image-inverse-permutations by blast
  have th2: ?rhs = setsum (f o inv) ?S by (simp add: o-def)
  from setsum-reindex[OF th0, of f]  show ?thesis unfolding th1 th2 .
qed

lemma setum-permutations-compose-left:
  assumes q: q permutes S
  shows setsum f {p. p permutes S} =
          setsum (λp. f(q o p)) {p. p permutes S} (is ?lhs = ?rhs)
proof−
  let ?S = {p. p permutes S}
  have th0: ?rhs = setsum (f o (op o q)) ?S by (simp add: o-def)
  have th1: inj-on (op o q) ?S
    apply (auto simp add: inj-on-def)
  proof−
    fix p r
    assume p permutes S and r:r permutes S and rp: q ∘ p = q ∘ r
    hence inv q o q o p = inv q o q o r by (simp add: o-assoc[symmetric])
    with permutes-inj[OF q, unfolded inj-iff]

    show p = r by simp
  qed
  have th3: (op o q) ' ?S = ?S using image-compose-permutations-left[OF q] by
auto
  from setsum-reindex[OF th1, of f]
  show ?thesis unfolding th0 th1 th3 .
qed

lemma sum-permutations-compose-right:
  assumes q: q permutes S
  shows setsum f {p. p permutes S} =
          setsum (λp. f(p o q)) {p. p permutes S} (is ?lhs = ?rhs)
proof−
  let ?S = {p. p permutes S}
  have th0: ?rhs = setsum (f o (λp. p o q)) ?S by (simp add: o-def)
  have th1: inj-on (λp. p o q) ?S
    apply (auto simp add: inj-on-def)
  proof−
    fix p r
    assume p permutes S and r:r permutes S and rp: p o q = r o q
    hence p o (q o inv q)  = r o (q o inv q) by (simp add: o-assoc)
    with permutes-surj[OF q, unfolded surj-iff]

    show p = r by simp
```

qed
have *th3*: $(\lambda p.\ p\ o\ q)$ ' *?S* = *?S* **using** *image-compose-permutations-right*[*OF q*]
by *auto*
from *setsum-reindex*[*OF th1*, *of f*]
show *?thesis* **unfolding** *th0 th1 th3* **.**
qed

lemma *setsum-over-permutations-insert*:
assumes *fS*: *finite S* **and** *aS*: $a \notin S$
shows *setsum f* $\{p.\ p$ *permutes* $(insert\ a\ S)\}$ = *setsum* $(\lambda b.\ setsum\ (\lambda q.\ f\ (Fun.swap\ a\ b\ id\ o\ q))\ \{p.\ p\ permutes\ S\})\ (insert\ a\ S)$
proof−
have *th0*: $\bigwedge f\ a\ b.\ (\lambda(b,p).\ f\ (Fun.swap\ a\ b\ id\ o\ p)) = f\ o\ (\lambda(b,p).\ Fun.swap\ a\ b\ id\ o\ p)$
by (*simp add*: *expand-fun-eq*)
have *th1*: $\bigwedge P\ Q.\ P \times Q = \{(a,b).\ a \in P \wedge b \in Q\}$ **by** *blast*
have *th2*: $\bigwedge P\ Q.\ P \Longrightarrow (P \Longrightarrow Q) \Longrightarrow P \wedge Q$ **by** *blast*
show *?thesis*
unfolding *permutes-insert*
unfolding *setsum-cartesian-product*
unfolding *th1*[*symmetric*]
unfolding *th0*
proof(*rule setsum-reindex*)
let *?f* = $(\lambda(b,\ y).\ Fun.swap\ a\ b\ id \circ y)$
let *?P* = $\{p.\ p\ permutes\ S\}$
{fix *b c p q* **assume** *b*: $b \in insert\ a\ S$ **and** *c*: $c \in insert\ a\ S$
and *p*: *p permutes S* **and** *q*: *q permutes S*
and *eq*: *Fun.swap a b id o p* = *Fun.swap a c id o q*
from *p q aS* **have** *pa*: *p a* = *a* **and** *qa*: *q a* = *a*
unfolding *permutes-def* **by** *metis*+
from *eq* **have** (*Fun.swap a b id o p*) *a* = (*Fun.swap a c id o q*) *a* **by** *simp*
hence *bc*: *b* = *c*
by (*simp add*: *permutes-def pa qa o-def fun-upd-def swap-def id-def cong del*: *if-weak-cong split*: *split-if-asm*)
from *eq*[*unfolded bc*] **have** $(\lambda p.\ Fun.swap\ a\ c\ id\ o\ p)\ (Fun.swap\ a\ c\ id\ o\ p) = (\lambda p.\ Fun.swap\ a\ c\ id\ o\ p)\ (Fun.swap\ a\ c\ id\ o\ q)$ **by** *simp*
hence *p* = *q* **unfolding** *o-assoc swap-id-idempotent*
by (*simp add*: *o-def*)
with *bc* **have** $b = c \wedge p = q$ **by** *blast*
}

then show *inj-on ?f* (*insert a S* × *?P*)
unfolding *inj-on-def*
apply *clarify* **by** *metis*
qed

qed

end

15 Glbs: Definitions of Lower Bounds and Greatest Lower Bounds, analogous to Lubs

theory *Glbs*
imports *Lubs*
begin

definition
greatestP :: ['*a* =>*bool*,'*a*::*ord*] => *bool* **where**
greatestP P x = (*P x* & *Collect P* *<= *x*)

definition
isLb :: ['*a set*, '*a set*, '*a*::*ord*] => *bool* **where**
isLb R S x = (*x* <=* *S* & *x*: *R*)

definition
isGlb :: ['*a set*, '*a set*, '*a*::*ord*] => *bool* **where**
isGlb R S x = *greatestP* (*isLb R S*) *x*

definition
lbs :: ['*a set*, '*a*::*ord set*] => '*a set* **where**
lbs R S = *Collect* (*isLb R S*)

15.1 Rules about the Operators *greatestP*, *isLb* and *isGlb*

lemma *leastPD1*: *greatestP P x* ==> *P x*
by (*simp add*: *greatestP-def*)

lemma *greatestPD2*: *greatestP P x* ==> *Collect P* *<= *x*
by (*simp add*: *greatestP-def*)

lemma *greatestPD3*: [| *greatestP P x*; *y*: *Collect P* |] ==> *x* >= *y*
by (*blast dest*!: *greatestPD2 setleD*)

lemma *isGlbD1*: *isGlb R S x* ==> *x* <=* *S*
by (*simp add*: *isGlb-def isLb-def greatestP-def*)

lemma *isGlbD1a*: *isGlb R S x* ==> *x*: *R*
by (*simp add*: *isGlb-def isLb-def greatestP-def*)

lemma *isGlb-isLb*: *isGlb R S x* ==> *isLb R S x*
apply (*simp add*: *isLb-def*)
apply (*blast dest*: *isGlbD1 isGlbD1a*)

done

lemma *isGlbD2*: [| *isGlb R S x*; *y* : *S* |] ==> *y* >= *x*
by (*blast dest*!: *isGlbD1 setgeD*)

lemma *isGlbD3*: *isGlb R S x* ==> *greatestP*(*isLb R S*) *x*
by (*simp add*: *isGlb-def*)

lemma *isGlbI1*: *greatestP*(*isLb R S*) *x* ==> *isGlb R S x*
by (*simp add*: *isGlb-def*)

lemma *isGlbI2*: [| *isLb R S x*; *Collect* (*isLb R S*) *<=* *x* |] ==> *isGlb R S x*
by (*simp add*: *isGlb-def greatestP-def*)

lemma *isLbD*: [| *isLb R S x*; *y* : *S* |] ==> *y* >= *x*
by (*simp add*: *isLb-def setge-def*)

lemma *isLbD2*: *isLb R S x* ==> *x* <=* *S*
by (*simp add*: *isLb-def*)

lemma *isLbD2a*: *isLb R S x* ==> *x*: *R*
by (*simp add*: *isLb-def*)

lemma *isLbI*: [| *x* <=* *S* ; *x*: *R* |] ==> *isLb R S x*
by (*simp add*: *isLb-def*)

lemma *isGlb-le-isLb*: [| *isGlb R S x*; *isLb R S y* |] ==> *x* >= *y*
apply (*simp add*: *isGlb-def*)
apply (*blast intro*!: *greatestPD3*)
done

lemma *isGlb-ubs*: *isGlb R S x* ==> *lbs R S* *<= *x*
apply (*simp add*: *lbs-def isGlb-def*)
apply (*erule greatestPD2*)
done

end

16 Topology-Euclidean-Space: Elementary topology in Euclidean space.

theory *Topology-Euclidean-Space*
imports *SEQ Euclidean-Space Glbs*
begin

16.1 General notion of a topology

definition *istopology* $L \longleftrightarrow \{\} \in L \wedge (\forall S \in L.\ \forall T \in L.\ S \cap T \in L) \wedge (\forall K.\ K \subseteq L \longrightarrow \bigcup K \in L)$
typedef (**open**) *'a topology* = $\{L::('a\ set)\ set.\ istopology\ L\}$
morphisms *openin topology*
unfolding *istopology-def* **by** *blast*

lemma *istopology-open-in*[*intro*]: *istopology*(*openin U*)
using *openin*[*of U*] **by** *blast*

lemma *topology-inverse'*: *istopology U* $\Longrightarrow$ *openin* (*topology U*) = *U*
using *topology-inverse*[*unfolded mem-def Collect-def*] **.**

lemma *topology-inverse-iff*: *istopology U* $\longleftrightarrow$ *openin* (*topology U*) = *U*
using *topology-inverse*[*of U*] *istopology-open-in*[*of topology U*] **by** *auto*

lemma *topology-eq*: *T1* = *T2* $\longleftrightarrow$ ($\forall S.$ *openin T1 S* $\longleftrightarrow$ *openin T2 S*)
proof−
{assume *T1*=*T2* **hence** $\forall S.$ *openin T1 S* $\longleftrightarrow$ *openin T2 S* **by** *simp***}**
moreover
{assume *H*: $\forall S.$ *openin T1 S* $\longleftrightarrow$ *openin T2 S*
hence *openin T1* = *openin T2* **by** (*metis mem-def set-ext*)
hence *topology* (*openin T1*) = *topology* (*openin T2*) **by** *simp*
hence *T1* = *T2* **unfolding** *openin-inverse* **.}**
ultimately show *?thesis* **by** *blast*
qed

Infer the "universe" from union of all sets in the topology.

definition *topspace T* = $\bigcup\{S.\ openin\ T\ S\}$

16.2 Main properties of open sets

lemma *openin-clauses*:
fixes *U* :: *'a topology*
shows *openin U* {}
$\bigwedge S\ T.$ *openin U S* $\Longrightarrow$ *openin U T* $\Longrightarrow$ *openin U* ($S \cap T$)
$\bigwedge K.$ ($\forall S \in K.$ *openin U S*) $\Longrightarrow$ *openin U* ($\bigcup K$)
using *openin*[*of U*] **unfolding** *istopology-def Collect-def mem-def*
unfolding *subset-eq Ball-def mem-def* **by** *auto*

lemma *openin-subset*[*intro*]: *openin U S* $\Longrightarrow$ $S \subseteq$ *topspace U*
unfolding *topspace-def* **by** *blast*
lemma *openin-empty*[*simp*]: *openin U* {} **by** (*simp add*: *openin-clauses*)

lemma *openin-Int*[*intro*]: *openin U S* $\Longrightarrow$ *openin U T* $\Longrightarrow$ *openin U* ($S \cap T$)
using *openin-clauses* **by** *simp*

lemma *openin-Union*[*intro*]: ($\forall S \in K.$ *openin U S*) $\Longrightarrow$ *openin U* ($\bigcup K$)
using *openin-clauses* **by** *simp*

lemma *openin-Un*[*intro*]: *openin U S* $\Longrightarrow$ *openin U T* $\Longrightarrow$ *openin U* (*S* $\cup$ *T*)
using *openin-Union*[*of* {*S*,*T*} *U*] **by** *auto*

lemma *openin-topspace*[*intro*, *simp*]: *openin U* (*topspace U*) **by** (*simp add*: *openin-Union topspace-def*)

lemma *openin-subopen*: *openin U S* $\longleftrightarrow$ ($\forall$ *x* $\in$ *S*. $\exists$ *T*. *openin U T* $\wedge$ *x* $\in$ *T* $\wedge$ *T* $\subseteq$ *S*) (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *?lhs* **then show** *?rhs* **by** *auto*
next
assume *H*: *?rhs*
let *?t* = $\bigcup$ {*T*. *openin U T* $\wedge$ *T* $\subseteq$ *S*}
have *openin U ?t* **by** (*simp add*: *openin-Union*)
also have *?t* = *S* **using** *H* **by** *auto*
finally show *openin U S* **.**
qed

16.3 Closed sets

definition *closedin U S* $\longleftrightarrow$ *S* $\subseteq$ *topspace U* $\wedge$ *openin U* (*topspace U* $-$ *S*)

lemma *closedin-subset*: *closedin U S* $\Longrightarrow$ *S* $\subseteq$ *topspace U* **by** (*metis closedin-def*)
lemma *closedin-empty*[*simp*]: *closedin U* {} **by** (*simp add*: *closedin-def*)
lemma *closedin-topspace*[*intro*,*simp*]:
closedin U (*topspace U*) **by** (*simp add*: *closedin-def*)
lemma *closedin-Un*[*intro*]: *closedin U S* $\Longrightarrow$ *closedin U T* $\Longrightarrow$ *closedin U* (*S* $\cup$ *T*)
by (*auto simp add*: *Diff-Un closedin-def*)

lemma *Diff-Inter*[*intro*]: *A* $-$ $\bigcap$ *S* = $\bigcup$ {*A* $-$ *s*|*s*. *s*$\in$*S*} **by** *auto*
lemma *closedin-Inter*[*intro*]: **assumes** *Ke*: *K* $\neq$ {} **and** *Kc*: $\forall$ *S* $\in$*K*. *closedin U S*
shows *closedin U* ($\bigcap$ *K*) **using** *Ke Kc* **unfolding** *closedin-def Diff-Inter* **by** *auto*

lemma *closedin-Int*[*intro*]: *closedin U S* $\Longrightarrow$ *closedin U T* $\Longrightarrow$ *closedin U* (*S* $\cap$ *T*)
using *closedin-Inter*[*of* {*S*,*T*} *U*] **by** *auto*

lemma *Diff-Diff-Int*: *A* $-$ (*A* $-$ *B*) = *A* $\cap$ *B* **by** *blast*
lemma *openin-closedin-eq*: *openin U S* $\longleftrightarrow$ *S* $\subseteq$ *topspace U* $\wedge$ *closedin U* (*topspace U* $-$ *S*)
apply (*auto simp add*: *closedin-def Diff-Diff-Int inf-absorb2*)
apply (*metis openin-subset subset-eq*)
done

lemma *openin-closedin*: *S* $\subseteq$ *topspace U* $\Longrightarrow$ (*openin U S* $\longleftrightarrow$ *closedin U* (*topspace*

$U - S$))
by (*simp add*: *openin-closedin-eq*)

lemma *openin-diff*[*intro*]: **assumes** *oS*: *openin U S* **and** *cT*: *closedin U T* **shows** *openin U* (*S* − *T*)
proof−
have *S* − *T* = *S* ∩ (*topspace U* − *T*) **using** *openin-subset*[*of U S*] *oS cT*
by (*auto simp add*: *topspace-def openin-subset*)
then show *?thesis* **using** *oS cT* **by** (*auto simp add*: *closedin-def*)
qed

lemma *closedin-diff*[*intro*]: **assumes** *oS*: *closedin U S* **and** *cT*: *openin U T* **shows** *closedin U* (*S* − *T*)
proof−
have *S* − *T* = *S* ∩ (*topspace U* − *T*) **using** *closedin-subset*[*of U S*] *oS cT*
by (*auto simp add*: *topspace-def*)
then show *?thesis* **using** *oS cT* **by** (*auto simp add*: *openin-closedin-eq*)
qed

16.4 Subspace topology.

definition *subtopology U V* = *topology* {*S* ∩ *V* |*S*. *openin U S*}

lemma *istopology-subtopology*: *istopology* {*S* ∩ *V* |*S*. *openin U S*} (**is** *istopology ?L*)
proof−
have {} ∈ *?L* **by** *blast*
{fix *A B* **assume** *A*: *A* ∈ *?L* **and** *B*: *B* ∈ *?L*
from *A B* **obtain** *Sa* **and** *Sb* **where** *Sa*: *openin U Sa A* = *Sa* ∩ *V* **and** *Sb*: *openin U Sb B* = *Sb* ∩ *V* **by** *blast*
have *A*∩*B* = (*Sa* ∩ *Sb*) ∩ *V openin U* (*Sa* ∩ *Sb*) **using** *Sa Sb* **by** *blast*+
then have *A* ∩ *B* ∈ *?L* **by** *blast***}**
moreover
{fix *K* **assume** *K*: *K* ⊆ *?L*
have *th0*: *?L* = (λ*S*. *S* ∩ *V*) ‘ *openin U*
apply (*rule set-ext*)
apply (*simp add*: *Ball-def image-iff*)
by (*metis mem-def*)
from *K*[*unfolded th0 subset-image-iff*]
obtain *Sk* **where** *Sk*: *Sk* ⊆ *openin U K* = (λ*S*. *S* ∩ *V*) ‘ *Sk* **by** *blast*
have ⋃*K* = (⋃*Sk*) ∩ *V* **using** *Sk* **by** *auto*
moreover have *openin U* (⋃ *Sk*) **using** *Sk* **by** (*auto simp add*: *subset-eq mem-def*)
ultimately have ⋃*K* ∈ *?L* **by** *blast***}**
ultimately show *?thesis* **unfolding** *istopology-def* **by** *blast*
qed

lemma *openin-subtopology*:
openin (*subtopology U V*) *S* ⟷ (∃ *T*. (*openin U T*) ∧ (*S* = *T* ∩ *V*))

unfolding *subtopology-def topology-inverse'*[*OF istopology-subtopology*]
by (*auto simp add*: *Collect-def*)

lemma *topspace-subtopology*: *topspace*(*subtopology U V*) = *topspace U* ∩ *V*
by (*auto simp add*: *topspace-def openin-subtopology*)

lemma *closedin-subtopology*:
closedin (*subtopology U V*) *S* ⟷ (∃ *T*. *closedin U T* ∧ *S* = *T* ∩ *V*)
unfolding *closedin-def topspace-subtopology*
apply (*simp add*: *openin-subtopology*)
apply (*rule iffI*)
apply *clarify*
apply (*rule-tac x*=*topspace U* − *T* **in** *exI*)
by *auto*

lemma *openin-subtopology-refl*: *openin* (*subtopology U V*) *V* ⟷ *V* ⊆ *topspace U*
unfolding *openin-subtopology*
apply (*rule iffI*, *clarify*)
apply (*frule openin-subset*[*of U*]) **apply** *blast*
apply (*rule exI*[**where** *x*=*topspace U*])
by *auto*

lemma *subtopology-superset*: **assumes** *UV*: *topspace U* ⊆ *V*
shows *subtopology U V* = *U*
proof−
{fix *S*
{fix *T* **assume** *T*: *openin U T S* = *T* ∩ *V*
from *T openin-subset*[*OF T*(*1*)] *UV* **have** *eq*: *S* = *T* **by** *blast*
have *openin U S* **unfolding** *eq* **using** *T* **by** *blast***}**
moreover
{assume *S*: *openin U S*
hence ∃ *T*. *openin U T* ∧ *S* = *T* ∩ *V*
using *openin-subset*[*OF S*] *UV* **by** *auto***}**
ultimately have (∃ *T*. *openin U T* ∧ *S* = *T* ∩ *V*) ⟷ *openin U S* **by** *blast***}**
then show *?thesis* **unfolding** *topology-eq openin-subtopology* **by** *blast*
qed

lemma *subtopology-topspace*[*simp*]: *subtopology U* (*topspace U*) = *U*
by (*simp add*: *subtopology-superset*)

lemma *subtopology-UNIV*[*simp*]: *subtopology U UNIV* = *U*
by (*simp add*: *subtopology-superset*)

16.5 The universal Euclidean versions are what we use most of the time

definition

euclidean :: *'a*::*topological-space topology* **where**
euclidean = *topology open*

lemma *open-openin*: *open S* ⟷ *openin euclidean S*
unfolding *euclidean-def*
apply (*rule cong*[**where** *x*=*S* **and** *y*=*S*])
apply (*rule topology-inverse*[*symmetric*])
apply (*auto simp add*: *istopology-def*)
by (*auto simp add*: *mem-def subset-eq*)

lemma *topspace-euclidean*: *topspace euclidean* = *UNIV*
apply (*simp add*: *topspace-def*)
apply (*rule set-ext*)
by (*auto simp add*: *open-openin*[*symmetric*])

lemma *topspace-euclidean-subtopology*[*simp*]: *topspace* (*subtopology euclidean S*)
= *S*
by (*simp add*: *topspace-euclidean topspace-subtopology*)

lemma *closed-closedin*: *closed S* ⟷ *closedin euclidean S*
by (*simp add*: *closed-def closedin-def topspace-euclidean open-openin Compl-eq-Diff-UNIV*)

lemma *open-subopen*: *open S* ⟷ (∀ *x*∈*S*. ∃ *T*. *open T* ∧ *x* ∈ *T* ∧ *T* ⊆ *S*)
by (*simp add*: *open-openin openin-subopen*[*symmetric*])

16.6 Open and closed balls.

definition
ball :: *'a*::*metric-space* ⇒ *real* ⇒ *'a set* **where**
ball x e = {*y*. *dist x y* < *e*}

definition
cball :: *'a*::*metric-space* ⇒ *real* ⇒ *'a set* **where**
cball x e = {*y*. *dist x y* ≤ *e*}

lemma *mem-ball*[*simp*]: *y* ∈ *ball x e* ⟷ *dist x y* < *e* **by** (*simp add*: *ball-def*)
lemma *mem-cball*[*simp*]: *y* ∈ *cball x e* ⟷ *dist x y* ≤ *e* **by** (*simp add*: *cball-def*)

lemma *mem-ball-0* [*simp*]:
fixes *x* :: *'a*::*real-normed-vector*
shows *x* ∈ *ball 0 e* ⟷ *norm x* < *e*
by (*simp add*: *dist-norm*)

lemma *mem-cball-0* [*simp*]:
fixes *x* :: *'a*::*real-normed-vector*
shows *x* ∈ *cball 0 e* ⟷ *norm x* ≤ *e*
by (*simp add*: *dist-norm*)

lemma *centre-in-cball*[*simp*]: *x* ∈ *cball x e* ⟷ *0*≤ *e* **by** *simp*

lemma *ball-subset-cball*[*simp*,*intro*]: *ball x e* ⊆ *cball x e* **by** (*simp add*: *subset-eq*)
lemma *subset-ball*[*intro*]: *d* <= *e* ==> *ball x d* ⊆ *ball x e* **by** (*simp add*: *subset-eq*)
lemma *subset-cball*[*intro*]: *d* <= *e* ==> *cball x d* ⊆ *cball x e* **by** (*simp add*: *subset-eq*)
lemma *ball-max-Un*: *ball a* (*max r s*) = *ball a r* ∪ *ball a s*
by (*simp add*: *expand-set-eq*) *arith*

lemma *ball-min-Int*: *ball a* (*min r s*) = *ball a r* ∩ *ball a s*
by (*simp add*: *expand-set-eq*)

lemma *diff-less-iff*: (*a*::*real*) − *b* > *0* ⟷ *a* > *b*
(*a*::*real*) − *b* < *0* ⟷ *a* < *b*
a − *b* < *c* ⟷ *a* < *c* +*b* *a* − *b* > *c* ⟷ *a* > *c* +*b* **by** *arith*+
lemma *diff-le-iff*: (*a*::*real*) − *b* ≥ *0* ⟷ *a* ≥ *b* (*a*::*real*) − *b* ≤ *0* ⟷ *a* ≤ *b*
a − *b* ≤ *c* ⟷ *a* ≤ *c* +*b* *a* − *b* ≥ *c* ⟷ *a* ≥ *c* +*b* **by** *arith*+

lemma *open-ball*[*intro*, *simp*]: *open* (*ball x e*)
unfolding *open-dist ball-def Collect-def Ball-def mem-def*
unfolding *dist-commute*
apply *clarify*
apply (*rule-tac x*=*e* − *dist xa x* **in** *exI*)
using *dist-triangle-alt*[**where** *z*=*x*]
apply (*clarsimp simp add*: *diff-less-iff*)
apply *atomize*
apply (*erule-tac x*=*y* **in** *allE*)
apply (*erule-tac x*=*xa* **in** *allE*)
by *arith*

lemma *centre-in-ball*[*simp*]: *x* ∈ *ball x e* ⟷ *e* > *0* **by** (*metis mem-ball dist-self*)
lemma *open-contains-ball*: *open S* ⟷ (∀ *x*∈*S*. ∃ *e*>*0*. *ball x e* ⊆ *S*)
unfolding *open-dist subset-eq mem-ball Ball-def dist-commute* **..**

lemma *openE*[*elim?*]:
assumes *open S x*∈*S*
obtains *e* **where** *e*>*0 ball x e* ⊆ *S*
using *assms* **unfolding** *open-contains-ball* **by** *auto*

lemma *open-contains-ball-eq*: *open S* ⟹ ∀ *x*. *x*∈*S* ⟷ (∃ *e*>*0*. *ball x e* ⊆ *S*)
by (*metis open-contains-ball subset-eq centre-in-ball*)

lemma *ball-eq-empty*[*simp*]: *ball x e* = {} ⟷ *e* ≤ *0*
unfolding *mem-ball expand-set-eq*
apply (*simp add*: *not-less*)
by (*metis zero-le-dist order-trans dist-self*)

lemma *ball-empty*[*intro*]: *e* ≤ *0* ==> *ball x e* = {} **by** *simp*

16.7 Basic "localization" results are handy for connectedness.

lemma *openin-open*: *openin* (*subtopology euclidean U*) *S* ⟷ (∃ *T*. *open T* ∧ (*S* = *U* ∩ *T*))
by (*auto simp add*: *openin-subtopology open-openin*[*symmetric*])

lemma *openin-open-Int*[*intro*]: *open S* ⟹ *openin* (*subtopology euclidean U*) (*U* ∩ *S*)
by (*auto simp add*: *openin-open*)

lemma *open-openin-trans*[*trans*]:
open S ⟹ *open T* ⟹ *T* ⊆ *S* ⟹ *openin* (*subtopology euclidean S*) *T*
by (*metis Int-absorb1 openin-open-Int*)

lemma *open-subset*: *S* ⊆ *T* ⟹ *open S* ⟹ *openin* (*subtopology euclidean T*) *S*
by (*auto simp add*: *openin-open*)

lemma *closedin-closed*: *closedin* (*subtopology euclidean U*) *S* ⟷ (∃ *T*. *closed T* ∧ *S* = *U* ∩ *T*)
by (*simp add*: *closedin-subtopology closed-closedin Int-ac*)

lemma *closedin-closed-Int*: *closed S* ==> *closedin* (*subtopology euclidean U*) (*U* ∩ *S*)
by (*metis closedin-closed*)

lemma *closed-closedin-trans*: *closed S* ⟹ *closed T* ⟹ *T* ⊆ *S* ⟹ *closedin* (*subtopology euclidean S*) *T*
apply (*subgoal-tac S* ∩ *T* = *T*)
apply *auto*
apply (*frule closedin-closed-Int*[*of T S*])
by *simp*

lemma *closed-subset*: *S* ⊆ *T* ⟹ *closed S* ⟹ *closedin* (*subtopology euclidean T*) *S*
by (*auto simp add*: *closedin-closed*)

lemma *openin-euclidean-subtopology-iff*:
fixes *S U* :: *'a*::*metric-space set*
shows *openin* (*subtopology euclidean U*) *S*
⟷ *S* ⊆ *U* ∧ (∀ *x*∈*S*. ∃ *e*>*0*. ∀ *x'*∈*U*. *dist x' x* < *e* ⟶ *x'*∈ *S*) (**is** *?lhs* ⟷ *?rhs*)
proof−
{assume *?lhs* **hence** *?rhs* **unfolding** *openin-subtopology open-openin*[*symmetric*]
by (*simp add*: *open-dist*) *blast***}**
moreover
{assume *SU*: *S* ⊆ *U* **and** *H*: ⋀*x*. *x* ∈ *S* ⟹ ∃ *e*>*0*. ∀ *x'*∈*U*. *dist x' x* < *e* ⟶ *x'* ∈ *S*
from *H* **obtain** *d* **where** *d*: ⋀*x* . *x*∈ *S* ⟹ *d x* > *0* ∧ (∀ *x'* ∈ *U*. *dist x' x* < *d x* ⟶ *x'* ∈ *S*)

by *metis*
let *?T* = $\bigcup${*B*. ∃ *x*∈*S*. *B* = *ball x* (*d x*)}
have *oT*: *open ?T* **by** *auto*
{ fix *x* **assume** *x*∈*S*
hence *x* ∈ $\bigcup${*B*. ∃ *x*∈*S*. *B* = *ball x* (*d x*)}
apply *simp* **apply**(*rule-tac x*=*ball x*(*d x*) **in** *exI*) **apply** *auto*
by (*rule d* [*THEN conjunct1*])
hence *x*∈ *?T* ∩ *U* **using** *SU* **and** ‹*x*∈*S*› **by** *auto* **}**
moreover
{ fix *y* **assume** *y*∈*?T*
then obtain *B* **where** *y*∈*B B*∈{*B*. ∃ *x*∈*S*. *B* = *ball x* (*d x*)} **by** *auto*
then obtain *x* **where** *x*∈*S* **and** *x*:*y* ∈ *ball x* (*d x*) **by** *auto*
assume *y*∈*U*
hence *y*∈*S* **using** *d*[*OF* ‹*x*∈*S*›] **and** *x* **by**(*auto simp add*: *dist-commute*) **}**
ultimately have *S* = *?T* ∩ *U* **by** *blast*
with *oT* **have** *?lhs* **unfolding** *openin-subtopology open-openin*[*symmetric*] **by** *blast*}
ultimately show *?thesis* **by** *blast*
qed

These "transitivity" results are handy too.

lemma *openin-trans*[*trans*]: *openin* (*subtopology euclidean T*) *S* ⟹ *openin* (*subtopology euclidean U*) *T*
⟹ *openin* (*subtopology euclidean U*) *S*
unfolding *open-openin openin-open* **by** *blast*

lemma *openin-open-trans*: *openin* (*subtopology euclidean T*) *S* ⟹ *open T* ⟹ *open S*
by (*auto simp add*: *openin-open intro*: *openin-trans*)

lemma *closedin-trans*[*trans*]:
closedin (*subtopology euclidean T*) *S* ⟹
closedin (*subtopology euclidean U*) *T*
==> *closedin* (*subtopology euclidean U*) *S*
by (*auto simp add*: *closedin-closed closed-closedin closed-Inter Int-assoc*)

lemma *closedin-closed-trans*: *closedin* (*subtopology euclidean T*) *S* ⟹ *closed T*
⟹ *closed S*
by (*auto simp add*: *closedin-closed intro*: *closedin-trans*)

16.8 Connectedness

definition *connected S* ⟷
~(∃ *e1 e2*. *open e1* ∧ *open e2* ∧ *S* ⊆ (*e1* ∪ *e2*) ∧ (*e1* ∩ *e2* ∩ *S* = {})
∧ ~(*e1* ∩ *S* = {}) ∧ ~(*e2* ∩ *S* = {}))

lemma *connected-local*:
connected S ⟷ ~(∃ *e1 e2*.
openin (*subtopology euclidean S*) *e1* ∧

openin (*subtopology euclidean S*) *e2* ∧
S ⊆ *e1* ∪ *e2* ∧
e1 ∩ *e2* = {} ∧
~(*e1* = {}) ∧
~(*e2* = {}))
unfolding *connected-def openin-open* **by** (*safe*, *blast*+)

lemma *exists-diff*:
fixes *P* :: *'a set* ⇒ *bool*
shows (∃ *S*. *P*(− *S*)) ⟷ (∃ *S*. *P S*) (**is** *?lhs* ⟷ *?rhs*)
proof−
{assume *?lhs* **hence** *?rhs* **by** *blast* **}**
moreover
{fix *S* **assume** *H*: *P S*
have *S* = − (− *S*) **by** *auto*
with *H* **have** *P* (− (− *S*)) **by** *metis* **}**
ultimately show *?thesis* **by** *metis*
qed

lemma *connected-clopen*: *connected S* ⟷
(∀ *T*. *openin* (*subtopology euclidean S*) *T* ∧
closedin (*subtopology euclidean S*) *T* ⟶ *T* = {} ∨ *T* = *S*) (**is** *?lhs* ⟷ *?rhs*)
proof−
have ¬ *connected S* ⟷ (∃ *e1 e2*. *open e1* ∧ *open* (− *e2*) ∧ *S* ⊆ *e1* ∪ (− *e2*) ∧ *e1* ∩ (− *e2*) ∩ *S* = {} ∧ *e1* ∩ *S* ≠ {} ∧ (− *e2*) ∩ *S* ≠ {})
unfolding *connected-def openin-open closedin-closed*
apply (*subst exists-diff*) **by** *blast*
hence *th0*: *connected S* ⟷ ¬ (∃ *e2 e1*. *closed e2* ∧ *open e1* ∧ *S* ⊆ *e1* ∪ (− *e2*) ∧ *e1* ∩ (− *e2*) ∩ *S* = {} ∧ *e1* ∩ *S* ≠ {} ∧ (− *e2*) ∩ *S* ≠ {})
(**is** - ⟷ ¬ (∃ *e2 e1*. *?P e2 e1*)) **apply** (*simp add*: *closed-def*) **by** *metis*

have *th1*: *?rhs* ⟷ ¬ (∃ *t' t*. *closed t'*∧*t* = *S*∩*t'* ∧ *t*≠{} ∧ *t*≠*S* ∧ (∃ *t'*. *open t'* ∧ *t* = *S* ∩ *t'*))
(**is** - ⟷ ¬ (∃ *t' t*. *?Q t' t*))
unfolding *connected-def openin-open closedin-closed* **by** *auto*
{fix *e2*
{fix *e1* **have** *?P e2 e1* ⟷ (∃ *t*. *closed e2* ∧ *t* = *S*∩*e2* ∧ *open e1* ∧ *t* = *S*∩*e1* ∧ *t*≠{} ∧ *t*≠*S*)
by *auto***}**
then have (∃ *e1*. *?P e2 e1*) ⟷ (∃ *t*. *?Q e2 t*) **by** *metis***}**
then have ∀ *e2*. (∃ *e1*. *?P e2 e1*) ⟷ (∃ *t*. *?Q e2 t*) **by** *blast*
then show *?thesis* **unfolding** *th0 th1* **by** *simp*
qed

lemma *connected-empty*[*simp*, *intro*]: *connected* {}
by (*simp add*: *connected-def*)

16.9 Hausdorff and other separation properties

class *t0-space* = *topological-space* +
  assumes *t0-space*: $x \neq y \Longrightarrow \exists\, U.\ \mathit{open}\ U \wedge \neg\ (x \in U \longleftrightarrow y \in U)$

class *t1-space* = *topological-space* +
  assumes *t1-space*: $x \neq y \Longrightarrow \exists\, U.\ \mathit{open}\ U \wedge x \in U \wedge y \notin U$

instance *t1-space* $\subseteq$ *t0-space*
proof qed (*fast dest*: *t1-space*)

lemma *separation-t1*:
  fixes $x\ y :: 'a$::*t1-space*
  shows $x \neq y \longleftrightarrow (\exists\, U.\ \mathit{open}\ U \wedge x \in U \wedge y \notin U)$
  using *t1-space*[*of* $x\ y$] **by** *blast*

lemma *closed-sing*:
  fixes $a :: 'a$::*t1-space*
  shows *closed* $\{a\}$
proof −
  let *?T* $= \bigcup\{S.\ \mathit{open}\ S \wedge a \notin S\}$
  have *open ?T* **by** (*simp add*: *open-Union*)
  also have *?T* $= -\ \{a\}$
    by (*simp add*: *expand-set-eq separation-t1*, *auto*)
  finally show *closed* $\{a\}$ **unfolding** *closed-def* **.**
qed

lemma *closed-insert* [*simp*]:
  fixes $a :: 'a$::*t1-space*
  assumes *closed* S **shows** *closed* (*insert* $a\ S$)
proof −
  from *closed-sing assms*
  have *closed* $(\{a\} \cup S)$ **by** (*rule closed-Un*)
  thus *closed* (*insert* $a\ S$) **by** *simp*
qed

lemma *finite-imp-closed*:
  fixes $S :: 'a$::*t1-space set*
  shows *finite* $S \Longrightarrow$ *closed* S
by (*induct set*: *finite*, *simp-all*)

T2 spaces are also known as Hausdorff spaces.

class *t2-space* = *topological-space* +
  assumes *hausdorff*: $x \neq y \Longrightarrow \exists\, U\ V.\ \mathit{open}\ U \wedge \mathit{open}\ V \wedge x \in U \wedge y \in V \wedge U \cap V = \{\}$

instance *t2-space* $\subseteq$ *t1-space*
proof qed (*fast dest*: *hausdorff*)

instance *metric-space* $\subseteq$ *t2-space*

proof
fix *x y* :: *'a*::*metric-space*
assume *xy*: $x \neq y$
let *?U* = *ball x* (*dist x y* / *2*)
let *?V* = *ball y* (*dist x y* / *2*)
have *th0*: $\bigwedge d\ x\ y\ z.\ (d\ x\ z :: real) <= d\ x\ y + d\ y\ z \Longrightarrow d\ y\ z = d\ z\ y$
$==> \sim(d\ x\ y * 2 < d\ x\ z \wedge d\ z\ y * 2 < d\ x\ z)$ **by** *arith*
have *open ?U* $\wedge$ *open ?V* $\wedge$ $x \in ?U \wedge y \in ?V \wedge ?U \cap ?V = \{\}$
using *dist-pos-lt*[*OF xy*] *th0*[*of dist*,*OF dist-triangle dist-commute*]
by (*auto simp add*: *expand-set-eq*)
then show $\exists\ U\ V.$ *open U* $\wedge$ *open V* $\wedge$ $x \in U \wedge y \in V \wedge U \cap V = \{\}$
by *blast*
qed

lemma *separation-t2*:
fixes *x y* :: *'a*::*t2-space*
shows $x \neq y \longleftrightarrow (\exists\ U\ V.$ *open U* $\wedge$ *open V* $\wedge$ $x \in U \wedge y \in V \wedge U \cap V = \{\})$
using *hausdorff*[*of x y*] **by** *blast*

lemma *separation-t0*:
fixes *x y* :: *'a*::*t0-space*
shows $x \neq y \longleftrightarrow (\exists\ U.$ *open U* $\wedge\ \sim(x \in U \longleftrightarrow y \in U))$
using *t0-space*[*of x y*] **by** *blast*

16.10 Limit points

definition
islimpt:: *'a*::*topological-space* $\Rightarrow$ *'a set* $\Rightarrow$ *bool*
(**infixr** *islimpt 60*) **where**
x islimpt S $\longleftrightarrow (\forall\ T.\ x \in T \longrightarrow$ *open T* $\longrightarrow (\exists\ y \in S.\ y \in T \wedge y \neq x))$

lemma *islimptI*:
assumes $\bigwedge T.\ x \in T \Longrightarrow$ *open T* $\Longrightarrow \exists\ y \in S.\ y \in T \wedge y \neq x$
shows *x islimpt S*
using *assms* **unfolding** *islimpt-def* **by** *auto*

lemma *islimptE*:
assumes *x islimpt S* **and** $x \in T$ **and** *open T*
obtains *y* **where** $y \in S$ **and** $y \in T$ **and** $y \neq x$
using *assms* **unfolding** *islimpt-def* **by** *auto*

lemma *islimpt-subset*: *x islimpt S* $\Longrightarrow S \subseteq T ==>$ *x islimpt T* **by** (*auto simp add*: *islimpt-def*)

lemma *islimpt-approachable*:
fixes *x* :: *'a*::*metric-space*
shows *x islimpt S* $\longleftrightarrow (\forall\ e>0.\ \exists\ x' \in S.\ x' \neq x \wedge$ *dist x' x* $< e)$
unfolding *islimpt-def*
apply *auto*

apply(*erule-tac x=ball x e* **in** *allE*)
apply *auto*
apply(*rule-tac x=y* **in** *bexI*)
apply (*auto simp add*: *dist-commute*)
apply (*simp add*: *open-dist*, *drule* (*1*) *bspec*)
apply (*clarify*, *drule spec*, *drule* (*1*) *mp*, *auto*)
done

lemma *islimpt-approachable-le*:
fixes $x :: 'a$::*metric-space*
shows x *islimpt* $S \longleftrightarrow (\forall e{>}0.\ \exists x'{\in}\ S.\ x' \neq x \wedge$ *dist* $x'\ x <= e)$
unfolding *islimpt-approachable*
using *approachable-lt-le*[**where** $f{=}\lambda x'.$ *dist* $x'\ x$ **and** $P{=}\lambda x'.\ \neg\ (x'{\in}S \wedge x'{\neq}x)$]
by *metis*

class *perfect-space* =

assumes *islimpt-UNIV* [*simp*, *intro*]: (x::$'a$::*metric-space*) *islimpt UNIV*

lemma *perfect-choose-dist*:
fixes $x :: 'a$::*perfect-space*
shows $0 < r \Longrightarrow \exists a.\ a \neq x \wedge$ *dist* $a\ x < r$
using *islimpt-UNIV* [*of x*]
by (*simp add*: *islimpt-approachable*)

instance *real* :: *perfect-space*
apply *default*
apply (*rule islimpt-approachable* [*THEN iffD2*])
apply (*clarify*, *rule-tac x=x + e/2* **in** *bexI*)
apply (*auto simp add*: *dist-norm*)
done

instance *cart* :: (*perfect-space*, *finite*) *perfect-space*
proof
fix $x :: 'a \hat{} 'b$
{
fix *e* :: *real* **assume** $0 < e$
def $a \equiv x\ \$$ *undefined*
have *a islimpt UNIV* **by** (*rule islimpt-UNIV*)
with ‹$0 < e$› **obtain** *b* **where** $b \neq a$ **and** *dist b a* $< e$
unfolding *islimpt-approachable* **by** *auto*
def $y \equiv$ *Cart-lambda* ((*Cart-nth x*)(*undefined* := *b*))
from ‹$b \neq a$› **have** $y \neq x$
unfolding *a-def y-def* **by** (*simp add*: *Cart-eq*)
from ‹*dist b a* $< e$› **have** *dist y x* $< e$
unfolding *dist-vector-def a-def y-def*
apply *simp*
apply (*rule le-less-trans* [*OF setL2-le-setsum* [*OF zero-le-dist*]])
apply (*subst setsum-diff1'* [**where** *a=undefined*], *simp*, *simp*, *simp*)

```
    done
   from ‹y ≠ x› and ‹dist y x < e›
   have ∃ y∈UNIV. y ≠ x ∧ dist y x < e by auto
  }
  then show x islimpt UNIV unfolding islimpt-approachable by blast
qed

lemma closed-limpt: closed S ⟷ (∀ x. x islimpt S ⟶ x ∈ S)
  unfolding closed-def
  apply (subst open-subopen)
  apply (simp add: islimpt-def subset-eq)
  by (metis ComplE ComplI insertCI insert-absorb mem-def)

lemma islimpt-EMPTY[simp]: ¬ x islimpt {}
  unfolding islimpt-def by auto

lemma closed-positive-orthant: closed {x::real^'n. ∀ i. 0 ≤x$i}
proof−
  let ?U = UNIV :: 'n set
  let ?O = {x::real^'n. ∀ i. x$i≥0}
  {fix x:: real^'n and i::'n assume H: ∀ e>0. ∃ x'∈?O. x' ≠ x ∧ dist x' x < e
    and xi: x$i < 0
    from xi have th0: −x$i > 0 by arith
    from H[rule-format, OF th0] obtain x' where x': x' ∈?O x' ≠ x dist x' x <
−x $ i by blast
      have th: ⋀b a (x::real). abs x <= b ⟹ b <= a ==> ~(a + x < 0) by
arith
     have th': ⋀x (y::real). x < 0 ⟹ 0 <= y ==> abs x <= abs (y − x) by
arith
     have th1: |x$i| ≤ |(x' − x)$i| using x'(1) xi
       apply (simp only: vector-component)
       by (rule th') auto
     have th2: |dist x x'| ≥ |(x' − x)$i| using  component-le-norm[of x'−x i]
       apply (simp add: dist-norm) by norm
     from th[OF th1 th2] x'(3) have False by (simp add: dist-commute) }
  then show ?thesis unfolding closed-limpt islimpt-approachable
    unfolding not-le[symmetric] by blast
qed

lemma finite-set-avoid:
  fixes a :: 'a::metric-space
  assumes fS: finite S shows  ∃ d>0. ∀ x∈S. x ≠ a ⟶ d <= dist a x
proof(induct rule: finite-induct[OF fS])
  case 1 thus ?case apply auto by ferrack
next
  case (2 x F)
  from 2 obtain d where d: d >0 ∀ x∈F. x≠a ⟶ d ≤ dist a x by blast
  {assume x = a hence ?case using d by auto  }
  moreover
```

{assume *xa*: $x \neq a$
let *?d* = *min d* (*dist a x*)
have *dp*: *?d* > *0* **using** *xa d*(*1*) **using** *dist-nz* **by** *auto*
from *d* **have** *d'*: $\forall x \in F.\ x \neq a \longrightarrow$ *?d* $\leq$ *dist a x* **by** *auto*
with *dp xa* **have** *?case* **by**(*auto intro*!: *exI*[**where** *x*=*?d*]) **}**
ultimately show *?case* **by** *blast*
qed

lemma *islimpt-finite*:
fixes *S* :: *'a*::*metric-space set*
assumes *fS*: *finite S* **shows** ¬ *a islimpt S*
unfolding *islimpt-approachable*
using *finite-set-avoid*[*OF fS*, *of a*] **by** (*metis dist-commute not-le*)

lemma *islimpt-Un*: *x islimpt* $(S \cup T) \longleftrightarrow$ *x islimpt S* $\vee$ *x islimpt T*
apply (*rule iffI*)
defer
apply (*metis Un-upper1 Un-upper2 islimpt-subset*)
unfolding *islimpt-def*
apply (*rule ccontr*, *clarsimp*, *rename-tac A B*)
apply (*drule-tac x*=*A* $\cap$ *B* **in** *spec*)
apply (*auto simp add*: *open-Int*)
done

lemma *discrete-imp-closed*:
fixes *S* :: *'a*::*metric-space set*
assumes *e*: *0* < *e* **and** *d*: $\forall x \in S.\ \forall y \in S.$ *dist y x* < *e* $\longrightarrow$ *y* = *x*
shows *closed S*
proof−
{fix *x* **assume** *C*: $\forall$ *e*>*0*. $\exists x' \in S.\ x' \neq x \wedge$ *dist x' x* < *e*
from *e* **have** *e2*: *e*/*2* > *0* **by** *arith*
from *C*[*rule-format*, *OF e2*] **obtain** *y* **where** *y*: *y* $\in$ *S y*$\neq$*x dist y x* < *e*/*2* **by** *blast*
let *?m* = *min* (*e*/*2*) (*dist x y*)
from *e2 y*(*2*) **have** *mp*: *?m* > *0* **by** (*simp add*: *dist-nz*[*THEN sym*])
from *C*[*rule-format*, *OF mp*] **obtain** *z* **where** *z*: *z* $\in$ *S z*$\neq$*x dist z x* < *?m* **by** *blast*
have *th*: *dist z y* < *e* **using** *z y*
by (*intro dist-triangle-lt* [**where** *z*=*x*], *simp*)
from *d*[*rule-format*, *OF y*(*1*) *z*(*1*) *th*] *y z*
have *False* **by** (*auto simp add*: *dist-commute*)**}**
then show *?thesis* **by** (*metis islimpt-approachable closed-limpt* [**where** *'a*=*'a*])
qed

16.11 Interior of a Set

definition *interior S* = $\{x.\ \exists T.$ *open T* $\wedge$ *x* $\in$ *T* $\wedge$ *T* $\subseteq$ *S*$\}$

lemma *interior-eq*: *interior S* = *S* $\longleftrightarrow$ *open S*

apply (*simp add*: *expand-set-eq interior-def*)
apply (*subst* (*2*) *open-subopen*) **by** (*safe*, *blast*+)

lemma *interior-open*: *open S* ==> (*interior S* = *S*) **by** (*metis interior-eq*)

lemma *interior-empty*[*simp*]: *interior* {} = {} **by** (*simp add*: *interior-def*)

lemma *open-interior*[*simp*, *intro*]: *open*(*interior S*)
apply (*simp add*: *interior-def*)
apply (*subst open-subopen*) **by** *blast*

lemma *interior-interior*[*simp*]: *interior*(*interior S*) = *interior S* **by** (*metis interior-eq open-interior*)
lemma *interior-subset*: *interior S* ⊆ *S* **by** (*auto simp add*: *interior-def*)
lemma *subset-interior*: *S* ⊆ *T* ==> (*interior S*) ⊆ (*interior T*) **by** (*auto simp add*: *interior-def*)
lemma *interior-maximal*: *T* ⊆ *S* ⟹ *open T* ==> *T* ⊆ (*interior S*) **by** (*auto simp add*: *interior-def*)
lemma *interior-unique*: *T* ⊆ *S* ⟹ *open T* ⟹ (∀ *T'*. *T'* ⊆ *S* ∧ *open T'* ⟶ *T'* ⊆ *T*) ⟹ *interior S* = *T*
by (*metis equalityI interior-maximal interior-subset open-interior*)
lemma *mem-interior*: *x* ∈ *interior S* ⟷ (∃ *e*. 0 < *e* ∧ *ball x e* ⊆ *S*)
apply (*simp add*: *interior-def*)
by (*metis open-contains-ball centre-in-ball open-ball subset-trans*)

lemma *open-subset-interior*: *open S* ==> *S* ⊆ *interior T* ⟷ *S* ⊆ *T*
by (*metis interior-maximal interior-subset subset-trans*)

lemma *interior-inter*[*simp*]: *interior*(*S* ∩ *T*) = *interior S* ∩ *interior T*
apply (*rule equalityI*, *simp*)
apply (*metis Int-lower1 Int-lower2 subset-interior*)
by (*metis Int-mono interior-subset open-Int open-interior open-subset-interior*)

lemma *interior-limit-point* [*intro*]:
fixes *x* :: *'a*::*perfect-space*
assumes *x*: *x* ∈ *interior S* **shows** *x islimpt S*
proof−
from *x* **obtain** *e* **where** *e*: *e*>*0* ∀ *x'*. *dist x x'* < *e* ⟶ *x'* ∈ *S*
unfolding *mem-interior subset-eq Ball-def mem-ball* **by** *blast*
{
fix *d*::*real* **assume** *d*: *d*>*0*
let *?m* = *min d e*
have *mde2*: *0* < *?m* **using** *e*(*1*) *d*(*1*) **by** *simp*
from *perfect-choose-dist* [*OF mde2*, *of x*]
obtain *y* **where** *y* ≠ *x* **and** *dist y x* < *?m* **by** *blast*
then have *dist y x* < *e dist y x* < *d* **by** *simp-all*
from ‹*dist y x* < *e*› *e*(*2*) **have** *y* ∈ *S* **by** (*simp add*: *dist-commute*)
have ∃ *x'*∈*S*. *x'*≠ *x* ∧ *dist x' x* < *d*
using ‹*y* ∈ *S*› ‹*y* ≠ *x*› ‹*dist y x* < *d*› **by** *fast*

```
  }
  then show ?thesis unfolding islimpt-approachable by blast
qed

lemma interior-closed-Un-empty-interior:
  assumes cS: closed S and iT: interior T = {}
  shows interior(S ∪ T) = interior S
proof
  show interior S ⊆ interior (S∪T)
    by (rule subset-interior, blast)
next
  show interior (S ∪ T) ⊆ interior S
  proof
    fix x assume x ∈ interior (S ∪ T)
    then obtain R where open R x ∈ R R ⊆ S ∪ T
      unfolding interior-def by fast
    show x ∈ interior S
    proof (rule ccontr)
      assume x ∉ interior S
      with ‹x ∈ R› ‹open R› obtain y where y ∈ R − S
        unfolding interior-def expand-set-eq by fast
      from ‹open R› ‹closed S› have open (R − S) by (rule open-Diff)
      from ‹R ⊆ S ∪ T› have R − S ⊆ T by fast
      from ‹y ∈ R − S› ‹open (R − S)› ‹R − S ⊆ T› ‹interior T = {}›
      show False unfolding interior-def by fast
    qed
  qed
qed
```

16.12 Closure of a Set

```
definition closure S = S ∪ {x | x. x islimpt S}

lemma closure-interior: closure S = − interior (− S)
proof−
  { fix x
    have x∈− interior (− S) ⟷ x ∈ closure S  (is ?lhs = ?rhs)
    proof
      let ?exT = λ y. (∃ T. open T ∧ y ∈ T ∧ T ⊆ − S)
      assume ?lhs
      hence ∗:¬ ?exT x
        unfolding interior-def
        by simp
      { assume ¬ ?rhs
        hence False using ∗
          unfolding closure-def islimpt-def
          by blast
      }
      thus ?rhs
```

```
      by blast
    next
      assume ?rhs thus ?lhs
        unfolding closure-def interior-def islimpt-def
        by blast
    qed
  }
  thus ?thesis
    by blast
qed

lemma interior-closure: interior S = - (closure (- S))
proof-
  { fix x
    have x ∈ interior S ⟷ x ∈ - (closure (- S))
      unfolding interior-def closure-def islimpt-def
      by auto
  }
  thus ?thesis
    by blast
qed

lemma closed-closure[simp, intro]: closed (closure S)
proof-
  have closed (- interior (-S)) by blast
  thus ?thesis using closure-interior[of S] by simp
qed

lemma closure-hull: closure S = closed hull S
proof-
  have S ⊆ closure S
    unfolding closure-def
    by blast
  moreover
  have closed (closure S)
    using closed-closure[of S]
    by assumption
  moreover
  { fix t
    assume *:S ⊆ t closed t
    { fix x
      assume x islimpt S
      hence x islimpt t using *(1)
        using islimpt-subset[of x, of S, of t]
        by blast
    }
    with * have closure S ⊆ t
      unfolding closure-def
      using closed-limpt[of t]
```

by *auto*
}
ultimately show *?thesis*
using *hull-unique*[*of S*, *of closure S*, *of closed*]
unfolding *mem-def*
by *simp*
qed

lemma *closure-eq*: *closure S* = *S* ⟷ *closed S*
unfolding *closure-hull*
using *hull-eq*[*of closed*, *unfolded mem-def*, *OF closed-Inter*, *of S*]
by (*metis mem-def subset-eq*)

lemma *closure-closed*[*simp*]: *closed S* ⟹ *closure S* = *S*
using *closure-eq*[*of S*]
by *simp*

lemma *closure-closure*[*simp*]: *closure* (*closure S*) = *closure S*
unfolding *closure-hull*
using *hull-hull*[*of closed S*]
by *assumption*

lemma *closure-subset*: *S* ⊆ *closure S*
unfolding *closure-hull*
using *hull-subset*[*of S closed*]
by *assumption*

lemma *subset-closure*: *S* ⊆ *T* ⟹ *closure S* ⊆ *closure T*
unfolding *closure-hull*
using *hull-mono*[*of S T closed*]
by *assumption*

lemma *closure-minimal*: *S* ⊆ *T* ⟹ *closed T* ⟹ *closure S* ⊆ *T*
using *hull-minimal*[*of S T closed*]
unfolding *closure-hull mem-def*
by *simp*

lemma *closure-unique*: *S* ⊆ *T* ∧ *closed T* ∧ (∀ *T'*. *S* ⊆ *T'* ∧ *closed T'* ⟶ *T* ⊆ *T'*) ⟹ *closure S* = *T*
using *hull-unique*[*of S T closed*]
unfolding *closure-hull mem-def*
by *simp*

lemma *closure-empty*[*simp*]: *closure* {} = {}
using *closed-empty closure-closed*[*of* {}]
by *simp*

lemma *closure-univ*[*simp*]: *closure UNIV* = *UNIV*
using *closure-closed*[*of UNIV*]

by *simp*

lemma *closure-eq-empty*: *closure* $S = \{\} \longleftrightarrow S = \{\}$
using *closure-empty closure-subset*[*of* S]
by *blast*

lemma *closure-subset-eq*: *closure* $S \subseteq S \longleftrightarrow$ *closed* S
using *closure-eq*[*of* S] *closure-subset*[*of* S]
by *simp*

lemma *open-inter-closure-eq-empty*:
open $S \Longrightarrow (S \cap$ *closure* $T) = \{\} \longleftrightarrow S \cap T = \{\}$
using *open-subset-interior*[*of* $S - T$]
using *interior-subset*[*of* $- T$]
unfolding *closure-interior*
by *auto*

lemma *open-inter-closure-subset*:
open $S \Longrightarrow (S \cap (closure\ T)) \subseteq closure(S \cap T)$
proof
fix x
assume *as*: *open* S $x \in S \cap$ *closure* T
{ assume $*$:x *islimpt* T
have x *islimpt* $(S \cap T)$
proof (*rule islimptI*)
fix A
assume $x \in A$ *open* A
with *as* **have** $x \in A \cap S$ *open* $(A \cap S)$
by (*simp-all add*: *open-Int*)
with $*$ **obtain** y **where** $y \in T$ $y \in A \cap S$ $y \neq x$
by (*rule islimptE*)
hence $y \in S \cap T$ $y \in A \wedge y \neq x$
by *simp-all*
thus $\exists\, y{\in}(S \cap T).\ y \in A \wedge y \neq x$ **..**
qed
}
then show $x \in$ *closure* $(S \cap T)$ **using** *as*
unfolding *closure-def*
by *blast*
qed

lemma *closure-complement*: $closure(-S) = -$ *interior*(S)
proof−
have $S = -(-S)$
by *auto*
thus *?thesis*
unfolding *closure-interior*
by *auto*
qed

lemma *interior-complement*: *interior*(− *S*) = − *closure*(*S*)
unfolding *closure-interior*
by *blast*

16.13 Frontier (aka boundary)

definition *frontier S* = *closure S* − *interior S*

lemma *frontier-closed*: *closed*(*frontier S*)
by (*simp add*: *frontier-def closed-Diff*)

lemma *frontier-closures*: *frontier S* = (*closure S*) ∩ (*closure*(− *S*))
by (*auto simp add*: *frontier-def interior-closure*)

lemma *frontier-straddle*:
fixes *a* :: *'a*::*metric-space*
shows *a* ∈ *frontier S* ⟷ (∀ *e*>*0*. (∃ *x*∈*S*. *dist a x* < *e*) ∧ (∃ *x*. *x* ∉ *S* ∧ *dist a x* < *e*)) (**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs*
{ fix *e*::*real*
assume *e* > *0*
let *?rhse* = (∃ *x*∈*S*. *dist a x* < *e*) ∧ (∃ *x*. *x* ∉ *S* ∧ *dist a x* < *e*)
{ assume *a*∈*S*
have ∃ *x*∈*S*. *dist a x* < *e* **using** ‹*e*>*0*› ‹*a*∈*S*› **by**(*rule-tac x*=*a* **in** *bexI*) *auto*
moreover have ∃ *x*. *x* ∉ *S* ∧ *dist a x* < *e* **using** ‹*?lhs*› ‹*a*∈*S*›
unfolding *frontier-closures closure-def islimpt-def* **using** ‹*e*>*0*›
by (*auto*, *erule-tac x*=*ball a e* **in** *allE*, *auto*)
ultimately have *?rhse* **by** *auto*
}
moreover
{ assume *a*∉*S*
hence *?rhse* **using** ‹*?lhs*›
unfolding *frontier-closures closure-def islimpt-def*
using *open-ball*[*of a e*] ‹*e* > *0*›
by *simp* (*metis centre-in-ball mem-ball open-ball*)
}
ultimately have *?rhse* **by** *auto*
}
thus *?rhs* **by** *auto*
next
assume *?rhs*
moreover
{ fix *T* **assume** *a*∉*S* **and**
as:∀ *e*>*0*. (∃ *x*∈*S*. *dist a x* < *e*) ∧ (∃ *x*. *x* ∉ *S* ∧ *dist a x* < *e*) *a* ∉ *S a* ∈ *T open T*
from ‹*open T*› ‹*a* ∈ *T*› **have** ∃ *e*>*0*. *ball a e* ⊆ *T* **unfolding** *open-contains-ball*[*of T*] **by** *auto*

then obtain *e* **where** *e>0 ball a e* $\subseteq$ *T* **by** *auto*
then obtain *y* **where** *y*:*y*$\in$*S dist a y* $<$ *e* **using** *as*(*1*) **by** *auto*
have $\exists$ *y*$\in$*S*. *y* $\in$ *T* $\wedge$ *y* $\neq$ *a*
using ‹*dist a y* $<$ *e*› ‹*ball a e* $\subseteq$ *T*› **unfolding** *ball-def* **using** ‹*y*$\in$*S*› ‹*a*$\notin$*S*›
by *auto*
}
hence *a* $\in$ *closure S* **unfolding** *closure-def islimpt-def* **using** ‹*?rhs*› **by** *auto*
moreover
{ fix *T* **assume** *a* $\in$ *T open T a*$\in$*S*
then obtain *e* **where** *e>0* **and** *balle*: *ball a e* $\subseteq$ *T* **unfolding** *open-contains-ball*
using ‹*?rhs*› **by** *auto*
obtain *x* **where** *x* $\notin$ *S dist a x* $<$ *e* **using** ‹*?rhs*› **using** ‹*e>0*› **by** *auto*
hence $\exists$ *y*$\in-$ *S*. *y* $\in$ *T* $\wedge$ *y* $\neq$ *a* **using** *balle* ‹*a*$\in$*S*› **unfolding** *ball-def* **by**
(*rule-tac x*=*x* **in** *bexI*)*auto*
}
hence *a islimpt* (*-* *S*) $\vee$ *a*$\notin$*S* **unfolding** *islimpt-def* **by** *auto*
ultimately show *?lhs* **unfolding** *frontier-closures* **using** *closure-def*[*of* *-* *S*]
by *auto*
qed

lemma *frontier-subset-closed*: *closed S* $\Longrightarrow$ *frontier S* $\subseteq$ *S*
by (*metis frontier-def closure-closed Diff-subset*)

lemma *frontier-empty*[*simp*]: *frontier* {} = {}
by (*simp add*: *frontier-def*)

lemma *frontier-subset-eq*: *frontier S* $\subseteq$ *S* $\longleftrightarrow$ *closed S*
proof−
{ assume *frontier S* $\subseteq$ *S*
hence *closure S* $\subseteq$ *S* **using** *interior-subset* **unfolding** *frontier-def* **by** *auto*
hence *closed S* **using** *closure-subset-eq* **by** *auto*
}
thus *?thesis* **using** *frontier-subset-closed*[*of S*] **..**
qed

lemma *frontier-complement*: *frontier*(*-* *S*) = *frontier S*
by (*auto simp add*: *frontier-def closure-complement interior-complement*)

lemma *frontier-disjoint-eq*: *frontier S* $\cap$ *S* = {} $\longleftrightarrow$ *open S*
using *frontier-complement frontier-subset-eq*[*of* *-* *S*]
unfolding *open-closed* **by** *auto*

16.14 Nets and the "eventually true" quantifier

Common nets and The "within" modifier for nets.

definition
at-infinity :: *'a*::*real-normed-vector net* **where**
at-infinity = *Abs-net* (λ*P*. $\exists$ *r*. $\forall$ *x*. *r* $\leq$ *norm x* $\longrightarrow$ *P x*)

definition
indirection :: *'a*::*real-normed-vector* ⇒ *'a* ⇒ *'a net* (**infixr** *indirection 70*) **where**
a indirection v = (*at a*) *within* {*b*. ∃ *c*≥*0*. *b* − *a* = *scaleR c v*}

Prove That They are all nets.

lemma *eventually-at-infinity*:
eventually P at-infinity ⟷ (∃ *b*. ∀ *x*. *norm x* >= *b* ⟶ *P x*)
unfolding *at-infinity-def*
proof (*rule eventually-Abs-net*, *rule is-filter.intro*)
fix *P Q* :: *'a* ⇒ *bool*
assume ∃ *r*. ∀ *x*. *r* ≤ *norm x* ⟶ *P x* **and** ∃ *s*. ∀ *x*. *s* ≤ *norm x* ⟶ *Q x*
then obtain *r s* **where**
∀ *x*. *r* ≤ *norm x* ⟶ *P x* **and** ∀ *x*. *s* ≤ *norm x* ⟶ *Q x* **by** *auto*
then have ∀ *x*. *max r s* ≤ *norm x* ⟶ *P x* ∧ *Q x* **by** *simp*
then show ∃ *r*. ∀ *x*. *r* ≤ *norm x* ⟶ *P x* ∧ *Q x* **..**
qed *auto*

Identify Trivial limits, where we can't approach arbitrarily closely.

definition
trivial-limit :: *'a net* ⇒ *bool* **where**
trivial-limit net ⟷ *eventually* (λ*x*. *False*) *net*

lemma *trivial-limit-within*:
shows *trivial-limit* (*at a within S*) ⟷ ¬ *a islimpt S*
proof
assume *trivial-limit* (*at a within S*)
thus ¬ *a islimpt S*
unfolding *trivial-limit-def*
unfolding *eventually-within eventually-at-topological*
unfolding *islimpt-def*
apply (*clarsimp simp add*: *expand-set-eq*)
apply (*rename-tac T*, *rule-tac x*=*T* **in** *exI*)
apply (*clarsimp*, *drule-tac x*=*y* **in** *bspec*, *simp-all*)
done
next
assume ¬ *a islimpt S*
thus *trivial-limit* (*at a within S*)
unfolding *trivial-limit-def*
unfolding *eventually-within eventually-at-topological*
unfolding *islimpt-def*
apply *clarsimp*
apply (*rule-tac x*=*T* **in** *exI*)
apply *auto*
done
qed

lemma *trivial-limit-at-iff*: *trivial-limit* (*at a*) ⟷ ¬ *a islimpt UNIV*
using *trivial-limit-within* [*of a UNIV*]
by (*simp add*: *within-UNIV*)

lemma *trivial-limit-at*:
fixes *a* :: *'a*::*perfect-space*
shows ¬ *trivial-limit* (*at a*)
by (*simp add*: *trivial-limit-at-iff*)

lemma *trivial-limit-at-infinity*:
¬ *trivial-limit* (*at-infinity* :: (*'a*::{*real-normed-vector*,*zero-neq-one*}) *net*)

unfolding *trivial-limit-def eventually-at-infinity*
apply *clarsimp*
apply (*rule-tac x*=*scaleR b* (*sgn 1*) **in** *exI*)
apply (*simp add*: *norm-sgn*)
done

lemma *trivial-limit-sequentially*[*intro*]: ¬ *trivial-limit sequentially*
by (*auto simp add*: *trivial-limit-def eventually-sequentially*)

Some property holds "sufficiently close" to the limit point.

lemma *eventually-at*:
eventually P (*at a*) ⟷ (∃ *d*>*0*. ∀ *x*. *0* < *dist x a* ∧ *dist x a* < *d* ⟶ *P x*)
unfolding *eventually-at dist-nz* **by** *auto*

lemma *eventually-within*: *eventually P* (*at a within S*) ⟷
(∃ *d*>*0*. ∀ *x*∈*S*. *0* < *dist x a* ∧ *dist x a* < *d* ⟶ *P x*)
unfolding *eventually-within eventually-at dist-nz* **by** *auto*

lemma *eventually-within-le*: *eventually P* (*at a within S*) ⟷
(∃ *d*>*0*. ∀ *x*∈*S*. *0* < *dist x a* ∧ *dist x a* <= *d* ⟶ *P x*) (**is** *?lhs* = *?rhs*)
unfolding *eventually-within*
by *auto* (*metis Rats-dense-in-nn-real order-le-less-trans order-refl*)

lemma *eventually-happens*: *eventually P net* ==> *trivial-limit net* ∨ (∃ *x*. *P x*)
unfolding *trivial-limit-def*
by (*auto elim*: *eventually-rev-mp*)

lemma *always-eventually*: (∀ *x*. *P x*) ==> *eventually P net*
proof −
assume ∀ *x*. *P x* **hence** *P* = (λ*x*. *True*) **by** (*simp add*: *ext*)
thus *eventually P net* **by** *simp*
qed

lemma *trivial-limit-eventually*: *trivial-limit net* ⟹ *eventually P net*
unfolding *trivial-limit-def* **by** (*auto elim*: *eventually-rev-mp*)

lemma *eventually-False*: *eventually* (λ*x*. *False*) *net* ⟷ *trivial-limit net*
unfolding *trivial-limit-def* **..**

lemma *trivial-limit-eq*: *trivial-limit net* ⟷ (∀ *P*. *eventually P net*)

apply (*safe elim*!: *trivial-limit-eventually*)
apply (*simp add*: *eventually-False* [*symmetric*])
done

Combining theorems for "eventually"

lemma *eventually-conjI*:
⟦*eventually* (λ*x*. *P x*) *net*; *eventually* (λ*x*. *Q x*) *net*⟧
⟹ *eventually* (λ*x*. *P x* ∧ *Q x*) *net*
by (*rule eventually-conj*)

lemma *eventually-rev-mono*:
eventually P net ⟹ (∀ *x*. *P x* ⟶ *Q x*) ⟹ *eventually Q net*
using *eventually-mono* [*of P Q*] **by** *fast*

lemma *eventually-and*: *eventually* (λ*x*. *P x* ∧ *Q x*) *net* ⟷ *eventually P net* ∧ *eventually Q net*
by (*auto intro*!: *eventually-conjI elim*: *eventually-rev-mono*)

lemma *eventually-false*: *eventually* (λ*x*. *False*) *net* ⟷ *trivial-limit net*
by (*auto simp add*: *eventually-False*)

lemma *not-eventually*: (∀ *x*. ¬ *P x*) ⟹ ~(*trivial-limit net*) ==> ~(*eventually* (λ*x*. *P x*) *net*)
by (*simp add*: *eventually-False*)

16.15 Limits

Notation Lim to avoid collition with lim defined in analysis

definition
Lim :: *'a net* ⇒ (*'a* ⇒ *'b*::*t2-space*) ⇒ *'b* **where**
Lim net f = (*THE l*. (*f* −−−> *l*) *net*)

lemma *Lim*:
(*f* −−−> *l*) *net* ⟷
trivial-limit net ∨
(∀ *e*>*0*. *eventually* (λ*x*. *dist* (*f x*) *l* < *e*) *net*)
unfolding *tendsto-iff trivial-limit-eq* **by** *auto*

Show that they yield usual definitions in the various cases.

lemma *Lim-within-le*: (*f* −−−> *l*)(*at a within S*) ⟷
(∀ *e*>*0*. ∃ *d*>*0*. ∀ *x*∈*S*. *0* < *dist x a* ∧ *dist x a* <= *d* ⟶ *dist* (*f x*) *l* < *e*)
by (*auto simp add*: *tendsto-iff eventually-within-le*)

lemma *Lim-within*: (*f* −−−> *l*) (*at a within S*) ⟷
(∀ *e* >*0*. ∃ *d*>*0*. ∀ *x* ∈ *S*. *0* < *dist x a* ∧ *dist x a* < *d* ⟶ *dist* (*f x*) *l* < *e*)
by (*auto simp add*: *tendsto-iff eventually-within*)

lemma *Lim-at*: $(f \longrightarrow l)\ (at\ a) \longleftrightarrow$
$(\forall e>0.\ \exists d>0.\ \forall x.\ 0 < dist\ x\ a \ \wedge\ dist\ x\ a < d \longrightarrow dist\ (f\ x)\ l < e)$
by (*auto simp add*: *tendsto-iff eventually-at*)

lemma *Lim-at-iff-LIM*: $(f \longrightarrow l)\ (at\ a) \longleftrightarrow f \dashrightarrow a \longrightarrow l$
unfolding *Lim-at LIM-def* **by** (*simp only*: *zero-less-dist-iff*)

lemma *Lim-at-infinity*:
$(f \longrightarrow l)\ at\text{-}infinity \longleftrightarrow (\forall e>0.\ \exists b.\ \forall x.\ norm\ x >= b \longrightarrow dist\ (f\ x)\ l < e)$
by (*auto simp add*: *tendsto-iff eventually-at-infinity*)

lemma *Lim-sequentially*:
$(S \longrightarrow l)\ sequentially \longleftrightarrow$
$(\forall e>0.\ \exists N.\ \forall n\geq N.\ dist\ (S\ n)\ l < e)$
by (*auto simp add*: *tendsto-iff eventually-sequentially*)

lemma *Lim-sequentially-iff-LIMSEQ*: $(S \longrightarrow l)\ sequentially \longleftrightarrow S \longrightarrow l$
unfolding *Lim-sequentially LIMSEQ-def* **..**

lemma *Lim-eventually*: $eventually\ (\lambda x.\ f\ x = l)\ net \Longrightarrow (f \longrightarrow l)\ net$
by (*rule topological-tendstoI*, *auto elim*: *eventually-rev-mono*)

The expected monotonicity property.

lemma *Lim-within-empty*: $(f \longrightarrow l)\ (net\ within\ \{\})$
unfolding *tendsto-def Limits.eventually-within* **by** *simp*

lemma *Lim-within-subset*: $(f \longrightarrow l)\ (net\ within\ S) \Longrightarrow T \subseteq S \Longrightarrow (f \longrightarrow l)\ (net\ within\ T)$
unfolding *tendsto-def Limits.eventually-within*
by (*auto elim*!: *eventually-elim1*)

lemma *Lim-Un*: **assumes** $(f \longrightarrow l)\ (net\ within\ S)\ (f \longrightarrow l)\ (net\ within\ T)$
shows $(f \longrightarrow l)\ (net\ within\ (S \cup T))$
using *assms* **unfolding** *tendsto-def Limits.eventually-within*
apply *clarify*
apply (*drule spec*, *drule* (*1*) *mp*, *drule* (*1*) *mp*)
apply (*drule spec*, *drule* (*1*) *mp*, *drule* (*1*) *mp*)
apply (*auto elim*: *eventually-elim2*)
done

lemma *Lim-Un-univ*:
$(f \longrightarrow l)\ (net\ within\ S) \Longrightarrow (f \longrightarrow l)\ (net\ within\ T) \Longrightarrow S \cup T = UNIV$
$\Longrightarrow (f \longrightarrow l)\ net$
by (*metis Lim-Un within-UNIV*)

Interrelations between restricted and unrestricted limits.

lemma *Lim-at-within*: $(f \longrightarrow l)\ net \Longrightarrow (f \longrightarrow l)(net\ within\ S)$

unfolding *tendsto-def Limits.eventually-within*
apply (*clarify*, *drule spec*, *drule* (*1*) *mp*, *drule* (*1*) *mp*)
by (*auto elim*!: *eventually-elim1*)

lemma *Lim-within-open*:
fixes f :: $'a$::*topological-space* ⇒ $'b$::*topological-space*
assumes $a \in S$ *open* S
shows $(f \longrightarrow l)(at\ a\ within\ S) \longleftrightarrow (f \longrightarrow l)(at\ a)$ (**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs*
{ fix A **assume** *open* A $l \in A$
with ‹*?lhs*› **have** *eventually* $(\lambda x.\ f\ x \in A)$ $(at\ a\ within\ S)$
by (*rule topological-tendstoD*)
hence *eventually* $(\lambda x.\ x \in S \longrightarrow f\ x \in A)$ $(at\ a)$
unfolding *Limits.eventually-within* **.**
then obtain T **where** *open* T $a \in T$ $\forall x{\in}T.\ x \neq a \longrightarrow x \in S \longrightarrow f\ x \in A$
unfolding *eventually-at-topological* **by** *fast*
hence *open* $(T \cap S)$ $a \in T \cap S$ $\forall x{\in}(T \cap S).\ x \neq a \longrightarrow f\ x \in A$
using *assms* **by** *auto*
hence $\exists T.$ *open* $T \wedge a \in T \wedge (\forall x{\in}T.\ x \neq a \longrightarrow f\ x \in A)$
by *fast*
hence *eventually* $(\lambda x.\ f\ x \in A)$ $(at\ a)$
unfolding *eventually-at-topological* **.**
}
thus *?rhs* **by** (*rule topological-tendstoI*)
next
assume *?rhs*
thus *?lhs* **by** (*rule Lim-at-within*)
qed

Another limit point characterization.

lemma *islimpt-sequential*:
fixes x :: $'a$::*metric-space*
shows x *islimpt* $S \longleftrightarrow (\exists f.\ (\forall n{::}nat.\ f\ n \in S - \{x\}) \wedge (f \longrightarrow x)\ sequentially)$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain f **where** $f{:}\forall y.\ y{>}0 \longrightarrow f\ y \in S \wedge f\ y \neq x \wedge dist\ (f\ y)\ x < y$
unfolding *islimpt-approachable* **using** *choice*[*of* $\lambda e\ y.\ e{>}0 \longrightarrow y{\in}S \wedge y{\neq}x \wedge$ *dist* $y\ x < e$] **by** *auto*
{ fix n::*nat*
have $f\ (inverse\ (real\ n + 1)) \in S - \{x\}$ **using** f **by** *auto*
}
moreover
{ fix e::*real* **assume** $e{>}0$
hence $\exists N{::}nat.\ inverse\ (real\ (N + 1)) < e$ **using** *real-arch-inv*[*of* e] **apply**
(*auto simp add*: *Suc-pred'*) **apply**(*rule-tac* $x{=}n - 1$ **in** *exI*) **by** *auto*
then obtain N::*nat* **where** *inverse* $(real\ (N + 1)) < e$ **by** *auto*
hence $\forall n{\geq}N.$ *inverse* $(real\ n + 1) < e$ **by** (*auto*, *metis Suc-le-mono le-SucE*

less-imp-inverse-less nat-le-real-less order-less-trans real-of-nat-Suc real-of-nat-Suc-gt-zero)
moreover have $\forall n \geq N.$ *dist* (*f* (*inverse* (*real n* + *1*))) *x* < (*inverse* (*real n* + *1*)) **using** *f* ‹*e*>*0*› **by** *auto*
ultimately have $\exists N$::*nat*. $\forall n \geq N.$ *dist* (*f* (*inverse* (*real n* + *1*))) *x* < *e* **apply**(*rule-tac x*=*N* **in** *exI*) **apply** *auto* **apply**(*erule-tac x*=*n* **in** *allE*)+ **by** *auto*
}
hence ((λ*n*. *f* (*inverse* (*real n* + *1*))) ---> *x*) *sequentially*
unfolding *Lim-sequentially* **using** *f* **by** *auto*
ultimately show *?rhs* **apply** (*rule-tac x*=(λ*n*::*nat*. *f* (*inverse* (*real n* + *1*))) **in** *exI*) **by** *auto*
next
assume *?rhs*
then obtain *f*::*nat*⇒'*a* **where** *f*:($\forall n.$ *f n* ∈ *S* − {*x*}) ($\forall e{>}0.$ $\exists N.$ $\forall n \geq N.$ *dist* (*f n*) *x* < *e*) **unfolding** *Lim-sequentially* **by** *auto*
{ fix *e*::*real* **assume** *e*>*0*
then obtain *N* **where** *dist* (*f N*) *x* < *e* **using** *f*(*2*) **by** *auto*
moreover have *f N*∈*S f N* ≠ *x* **using** *f*(*1*) **by** *auto*
ultimately have $\exists x'{\in}S.$ *x'* ≠ *x* ∧ *dist x' x* < *e* **by** *auto*
}
thus *?lhs* **unfolding** *islimpt-approachable* **by** *auto*
qed

Basic arithmetical combining theorems for limits.

lemma *Lim-linear*:
assumes (*f* ---> *l*) *net bounded-linear h*
shows ((λ*x*. *h* (*f x*)) ---> *h l*) *net*
using ‹*bounded-linear h*› ‹(*f* ---> *l*) *net*›
by (*rule bounded-linear.tendsto*)

lemma *Lim-ident-at*: ((λ*x*. *x*) ---> *a*) (*at a*)
unfolding *tendsto-def Limits.eventually-at-topological* **by** *fast*

lemma *Lim-const*[*intro*]: ((λ*x*. *a*) ---> *a*) *net* **by** (*rule tendsto-const*)

lemma *Lim-cmul*[*intro*]:
fixes *f* :: '*a* ⇒ '*b*::*real-normed-vector*
shows (*f* ---> *l*) *net* ==> ((λ*x*. *c* *$_R$ *f x*) ---> *c* *$_R$ *l*) *net*
by (*intro tendsto-intros*)

lemma *Lim-neg*:
fixes *f* :: '*a* ⇒ '*b*::*real-normed-vector*
shows (*f* ---> *l*) *net* ==> ((λ*x*. −(*f x*)) ---> −*l*) *net*
by (*rule tendsto-minus*)

lemma *Lim-add*: **fixes** *f* :: '*a* ⇒ '*b*::*real-normed-vector* **shows**
(*f* ---> *l*) *net* ⟹ (*g* ---> *m*) *net* ⟹ ((λ*x*. *f*(*x*) + *g*(*x*)) ---> *l* + *m*) *net*
by (*rule tendsto-add*)

lemma *Lim-sub*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
shows $(f \longrightarrow l)\ net \Longrightarrow (g \longrightarrow m)\ net \Longrightarrow ((\lambda x.\ f(x) - g(x)) \longrightarrow l - m)\ net$
by (*rule tendsto-diff*)

lemma *Lim-mul*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
assumes $(c \longrightarrow d)\ net\ \ (f \longrightarrow l)\ net$
shows $((\lambda x.\ c(x) *_R f\ x) \longrightarrow (d *_R l))\ net$
using *assms* **by** (*rule scaleR.tendsto*)

lemma *Lim-inv*:
fixes $f :: 'a \Rightarrow real$
assumes $(f \longrightarrow l)\ (net::'a\ net)\ \ l \neq 0$
shows $((inverse\ o\ f) \longrightarrow inverse\ l)\ net$
unfolding *o-def* **using** *assms* **by** (*rule tendsto-inverse*)

lemma *Lim-vmul*:
fixes $c :: 'a \Rightarrow real$ **and** $v :: 'b$::*real-normed-vector*
shows $(c \longrightarrow d)\ net \Longrightarrow ((\lambda x.\ c(x) *_R v) \longrightarrow d *_R v)\ net$
by (*intro tendsto-intros*)

lemma *Lim-null*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
shows $(f \longrightarrow l)\ net \longleftrightarrow ((\lambda x.\ f(x) - l) \longrightarrow 0)\ net$ **by** (*simp add*: *Lim dist-norm*)

lemma *Lim-null-norm*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
shows $(f \longrightarrow 0)\ net \longleftrightarrow ((\lambda x.\ norm(f\ x)) \longrightarrow 0)\ net$
by (*simp add*: *Lim dist-norm*)

lemma *Lim-null-comparison*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
assumes *eventually* $(\lambda x.\ norm\ (f\ x) \leq g\ x)\ net\ (g \longrightarrow 0)\ net$
shows $(f \longrightarrow 0)\ net$
proof(*simp add*: *tendsto-iff*, *rule*+)
fix *e*::*real* **assume** $0<e$
{ fix x
assume $norm\ (f\ x) \leq g\ x\ dist\ (g\ x)\ 0 < e$
hence $dist\ (f\ x)\ 0 < e$ **by** (*simp add*: *dist-norm*)
}
thus *eventually* $(\lambda x.\ dist\ (f\ x)\ 0 < e)\ net$
using *eventually-and*[*of* $\lambda x.\ norm(f\ x) <= g\ x\ \lambda x.\ dist\ (g\ x)\ 0 < e\ net$]
using *eventually-mono*[*of* $(\lambda x.\ norm\ (f\ x) \leq g\ x \wedge dist\ (g\ x)\ 0 < e)\ (\lambda x.\ dist\ (f\ x)\ 0 < e)\ net$]
using *assms* ‹*e*>*0*› **unfolding** *tendsto-iff* **by** *auto*
qed

lemma *Lim-component*:
fixes $f :: 'a \Rightarrow 'b$::*metric-space* ^ $'n$
shows $(f \longrightarrow l)$ *net* $\Longrightarrow ((\lambda a.\ f\ a\ \$i) \longrightarrow l\$i)$ *net*
unfolding *tendsto-iff*
apply (*clarify*)
apply (*drule spec*, *drule* (*1*) *mp*)
apply (*erule eventually-elim1*)
apply (*erule le-less-trans* [*OF dist-nth-le*])
done

lemma *Lim-transform-bound*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
fixes $g :: 'a \Rightarrow 'c$::*real-normed-vector*
assumes *eventually* $(\lambda n.\ norm(f\ n) <= norm(g\ n))$ *net* $(g \longrightarrow 0)$ *net*
shows $(f \longrightarrow 0)$ *net*
proof (*rule tendstoI*)
fix *e*::*real* **assume** *e>0*
{ fix *x*
assume $norm\ (f\ x) \leq norm\ (g\ x)$ $dist\ (g\ x)\ 0 < e$
hence $dist\ (f\ x)\ 0 < e$ **by** (*simp add*: *dist-norm*)**}**
thus *eventually* $(\lambda x.\ dist\ (f\ x)\ 0 < e)$ *net*
using *eventually-and*[*of* $\lambda x.\ norm\ (f\ x) \leq norm\ (g\ x)$ $\lambda x.\ dist\ (g\ x)\ 0 < e$ *net*]
using *eventually-mono*[*of* $\lambda x.\ norm\ (f\ x) \leq norm\ (g\ x) \wedge dist\ (g\ x)\ 0 < e$ $\lambda x.$ *dist* $(f\ x)\ 0 < e$ *net*]
using *assms* ‹*e>0*› **unfolding** *tendsto-iff* **by** *blast*
qed

Deducing things about the limit from the elements.

lemma *Lim-in-closed-set*:
assumes *closed S eventually* $(\lambda x.\ f(x) \in S)$ *net* $\neg$(*trivial-limit net*) $(f \longrightarrow l)$ *net*
shows $l \in S$
proof (*rule ccontr*)
assume $l \notin S$
with ‹*closed S*› **have** *open* $(-\ S)$ $l \in -\ S$
by (*simp-all add*: *open-Compl*)
with *assms*(*4*) **have** *eventually* $(\lambda x.\ f\ x \in -\ S)$ *net*
by (*rule topological-tendstoD*)
with *assms*(*2*) **have** *eventually* $(\lambda x.\ False)$ *net*
by (*rule eventually-elim2*) *simp*
with *assms*(*3*) **show** *False*
by (*simp add*: *eventually-False*)
qed

Need to prove closed(cball(x,e)) before deducing this as a corollary.

lemma *Lim-dist-ubound*:
assumes $\neg$(*trivial-limit net*) $(f \longrightarrow l)$ *net eventually* $(\lambda x.\ dist\ a\ (f\ x) <= e)$ *net*

shows *dist a l <= e*
proof (*rule ccontr*)
assume ¬ *dist a l ≤ e*
then have *0 < dist a l − e* **by** *simp*
with *assms(2)* **have** *eventually* (λ*x*. *dist* (*f x*) *l < dist a l − e*) *net*
by (*rule tendstoD*)
with *assms(3)* **have** *eventually* (λ*x*. *dist a* (*f x*) ≤ *e* ∧ *dist* (*f x*) *l < dist a l − e*) *net*
by (*rule eventually-conjI*)
then obtain *w* **where** *dist a* (*f w*) ≤ *e dist* (*f w*) *l < dist a l − e*
using *assms(1) eventually-happens* **by** *auto*
hence *dist a* (*f w*) + *dist* (*f w*) *l < e* + (*dist a l − e*)
by (*rule add-le-less-mono*)
hence *dist a* (*f w*) + *dist* (*f w*) *l < dist a l*
by *simp*
also have ... ≤ *dist a* (*f w*) + *dist* (*f w*) *l*
by (*rule dist-triangle*)
finally show *False* **by** *simp*
qed

lemma *Lim-norm-ubound*:
fixes *f* :: *'a* ⇒ *'b*::*real-normed-vector*
assumes ¬(*trivial-limit net*) (*f* −−−> *l*) *net eventually* (λ*x*. *norm*(*f x*) <= *e*) *net*
shows *norm*(*l*) <= *e*
proof (*rule ccontr*)
assume ¬ *norm l ≤ e*
then have *0 < norm l − e* **by** *simp*
with *assms(2)* **have** *eventually* (λ*x*. *dist* (*f x*) *l < norm l − e*) *net*
by (*rule tendstoD*)
with *assms(3)* **have** *eventually* (λ*x*. *norm* (*f x*) ≤ *e* ∧ *dist* (*f x*) *l < norm l − e*) *net*
by (*rule eventually-conjI*)
then obtain *w* **where** *norm* (*f w*) ≤ *e dist* (*f w*) *l < norm l − e*
using *assms(1) eventually-happens* **by** *auto*
hence *norm* (*f w − l*) < *norm l − e norm* (*f w*) ≤ *e* **by** (*simp-all add*: *dist-norm*)
hence *norm* (*f w − l*) + *norm* (*f w*) < *norm l* **by** *simp*
hence *norm* (*f w − l − f w*) < *norm l* **by** (*rule le-less-trans* [*OF norm-triangle-ineq4*])
thus *False* **using** ‹¬ *norm l ≤ e*› **by** *simp*
qed

lemma *Lim-norm-lbound*:
fixes *f* :: *'a* ⇒ *'b*::*real-normed-vector*
assumes ¬ (*trivial-limit net*) (*f* −−−> *l*) *net eventually* (λ*x*. *e* <= *norm*(*f x*)) *net*
shows *e ≤ norm l*
proof (*rule ccontr*)
assume ¬ *e ≤ norm l*
then have *0 < e − norm l* **by** *simp*

with *assms*(2) **have** *eventually* (λx. *dist* (*f x*) *l* < *e* − *norm l*) *net*
by (*rule tendstoD*)
with *assms*(3) **have** *eventually* (λx. *e* ≤ *norm* (*f x*) ∧ *dist* (*f x*) *l* < *e* − *norm l*) *net*
by (*rule eventually-conjI*)
then obtain *w* **where** *e* ≤ *norm* (*f w*) *dist* (*f w*) *l* < *e* − *norm l*
using *assms*(1) *eventually-happens* **by** *auto*
hence *norm* (*f w* − *l*) + *norm l* < *e e* ≤ *norm* (*f w*) **by** (*simp-all add*: *dist-norm*)
hence *norm* (*f w* − *l*) + *norm l* < *norm* (*f w*) **by** (*rule less-le-trans*)
hence *norm* (*f w* − *l* + *l*) < *norm* (*f w*) **by** (*rule le-less-trans* [*OF norm-triangle-ineq*])
thus *False* **by** *simp*
qed

Uniqueness of the limit, when nontrivial.

lemma *Lim-unique*:
fixes *f* :: *'a* ⇒ *'b*::*t2-space*
assumes ¬ *trivial-limit net* (*f* −−−> *l*) *net* (*f* −−−> *l'*) *net*
shows *l* = *l'*
proof (*rule ccontr*)
assume *l* ≠ *l'*
obtain *U V* **where** *open U open V l* ∈ *U l'* ∈ *V U* ∩ *V* = {}
using *hausdorff* [*OF* ‹*l* ≠ *l'*›] **by** *fast*
have *eventually* (λx. *f x* ∈ *U*) *net*
using ‹(*f* −−−> *l*) *net*› ‹*open U*› ‹*l* ∈ *U*› **by** (*rule topological-tendstoD*)
moreover
have *eventually* (λx. *f x* ∈ *V*) *net*
using ‹(*f* −−−> *l'*) *net*› ‹*open V*› ‹*l'* ∈ *V*› **by** (*rule topological-tendstoD*)
ultimately
have *eventually* (λx. *False*) *net*
proof (*rule eventually-elim2*)
fix *x*
assume *f x* ∈ *U f x* ∈ *V*
hence *f x* ∈ *U* ∩ *V* **by** *simp*
with ‹*U* ∩ *V* = {}› **show** *False* **by** *simp*
qed
with ‹¬ *trivial-limit net*› **show** *False*
by (*simp add*: *eventually-False*)
qed

lemma *tendsto-Lim*:
fixes *f* :: *'a* ⇒ *'b*::*t2-space*
shows ~(*trivial-limit net*) ⟹ (*f* −−−> *l*) *net* ==> *Lim net f* = *l*
unfolding *Lim-def* **using** *Lim-unique*[*of net f*] **by** *auto*

Limit under bilinear function

lemma *Lim-bilinear*:
assumes (*f* −−−> *l*) *net* **and** (*g* −−−> *m*) *net* **and** *bounded-bilinear h*
shows ((λx. *h* (*f x*) (*g x*)) −−−> (*h l m*)) *net*
using ‹*bounded-bilinear h*› ‹(*f* −−−> *l*) *net*› ‹(*g* −−−> *m*) *net*›

by (*rule bounded-bilinear.tendsto*)

These are special for limits out of the same vector space.

lemma *Lim-within-id*: (*id* −−−> *a*) (*at a within s*)
unfolding *tendsto-def Limits.eventually-within eventually-at-topological*
by *auto*

lemmas *Lim-intros* = *Lim-add Lim-const Lim-sub Lim-cmul Lim-vmul Lim-within-id*

lemma *Lim-at-id*: (*id* −−−> *a*) (*at a*)
apply (*subst within-UNIV*[*symmetric*]) **by** (*simp add*: *Lim-within-id*)

lemma *Lim-at-zero*:
fixes *a* :: *'a*::*real-normed-vector*
fixes *l* :: *'b*::*topological-space*
shows (*f* −−−> *l*) (*at a*) ⟷ ((λ*x*. *f*(*a* + *x*)) −−−> *l*) (*at 0*) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
{ fix *S* **assume** *open S l* ∈ *S*
with ‹*?lhs*› **have** *eventually* (λ*x*. *f x* ∈ *S*) (*at a*)
by (*rule topological-tendstoD*)
then obtain *d* **where** *d*: *d*>*0* ∀*x*. *x* ≠ *a* ∧ *dist x a* < *d* ⟶ *f x* ∈ *S*
unfolding *Limits.eventually-at* **by** *fast*
{ fix *x*::*'a* **assume** *x* ≠ *0* ∧ *dist x 0* < *d*
hence *f* (*a* + *x*) ∈ *S* **using** *d*
apply(*erule-tac x*=*x*+*a* **in** *allE*)
by (*auto simp add*: *add-commute dist-norm dist-commute*)
}
hence ∃*d*>*0*. ∀*x*. *x* ≠ *0* ∧ *dist x 0* < *d* ⟶ *f* (*a* + *x*) ∈ *S*
using *d*(*1*) **by** *auto*
hence *eventually* (λ*x*. *f* (*a* + *x*) ∈ *S*) (*at 0*)
unfolding *Limits.eventually-at* **.**
}
thus *?rhs* **by** (*rule topological-tendstoI*)
next
assume *?rhs*
{ fix *S* **assume** *open S l* ∈ *S*
with ‹*?rhs*› **have** *eventually* (λ*x*. *f* (*a* + *x*) ∈ *S*) (*at 0*)
by (*rule topological-tendstoD*)
then obtain *d* **where** *d*: *d*>*0* ∀*x*. *x* ≠ *0* ∧ *dist x 0* < *d* ⟶ *f* (*a* + *x*) ∈ *S*
unfolding *Limits.eventually-at* **by** *fast*
{ fix *x*::*'a* **assume** *x* ≠ *a* ∧ *dist x a* < *d*
hence *f x* ∈ *S* **using** *d* **apply**(*erule-tac x*=*x*−*a* **in** *allE*)
by(*auto simp add*: *add-commute dist-norm dist-commute*)
}
hence ∃*d*>*0*. ∀*x*. *x* ≠ *a* ∧ *dist x a* < *d* ⟶ *f x* ∈ *S* **using** *d*(*1*) **by** *auto*
hence *eventually* (λ*x*. *f x* ∈ *S*) (*at a*) **unfolding** *Limits.eventually-at* **.**
}
thus *?lhs* **by** (*rule topological-tendstoI*)

qed

It's also sometimes useful to extract the limit point from the net.

definition
netlimit :: *'a::t2-space net* ⇒ *'a* **where**
netlimit net = (*SOME a*. ((λ*x*. *x*) −−−> *a*) *net*)

lemma *netlimit-within*:
assumes ¬ *trivial-limit* (*at a within S*)
shows *netlimit* (*at a within S*) = *a*
unfolding *netlimit-def*
apply (*rule some-equality*)
apply (*rule Lim-at-within*)
apply (*rule Lim-ident-at*)
apply (*erule Lim-unique* [*OF assms*])
apply (*rule Lim-at-within*)
apply (*rule Lim-ident-at*)
done

lemma *netlimit-at*:
fixes *a* :: *'a::perfect-space*
shows *netlimit* (*at a*) = *a*
apply (*subst within-UNIV*[*symmetric*])
using *netlimit-within*[*of a UNIV*]
by (*simp add*: *trivial-limit-at within-UNIV*)

Transformation of limit.

lemma *Lim-transform*:
fixes *f g* :: *'a::type* ⇒ *'b::real-normed-vector*
assumes ((λ*x*. *f x* − *g x*) −−−> *0*) *net* (*f* −−−> *l*) *net*
shows (*g* −−−> *l*) *net*
proof−
from *assms* **have** ((λ*x*. *f x* − *g x* − *f x*) −−−> *0* − *l*) *net* **using** *Lim-sub*[*of* λ*x*. *f x* − *g x 0 net f l*] **by** *auto*
thus *?thesis* **using** *Lim-neg* [*of* λ *x*. − *g x* −*l net*] **by** *auto*
qed

lemma *Lim-transform-eventually*:
eventually (λ*x*. *f x* = *g x*) *net* ⟹ (*f* −−−> *l*) *net* ⟹ (*g* −−−> *l*) *net*
apply (*rule topological-tendstoI*)
apply (*drule* (*2*) *topological-tendstoD*)
apply (*erule* (*1*) *eventually-elim2*, *simp*)
done

lemma *Lim-transform-within*:
assumes *0* < *d* **and** ∀ *x'*∈*S*. *0* < *dist x' x* ∧ *dist x' x* < *d* ⟶ *f x'* = *g x'*
and (*f* −−−> *l*) (*at x within S*)
shows (*g* −−−> *l*) (*at x within S*)
proof (*rule Lim-transform-eventually*)

show *eventually* $(\lambda x.\ f\ x = g\ x)$ $(at\ x\ within\ S)$
unfolding *eventually-within*
using *assms*(*1*,*2*) **by** *auto*
show $(f \longrightarrow l)$ $(at\ x\ within\ S)$ **by** *fact*
qed

lemma *Lim-transform-at*:
assumes $0 < d$ **and** $\forall x'.\ 0 < dist\ x'\ x \wedge dist\ x'\ x < d \longrightarrow f\ x' = g\ x'$
and $(f \longrightarrow l)$ $(at\ x)$
shows $(g \longrightarrow l)$ $(at\ x)$
proof (*rule Lim-transform-eventually*)
show *eventually* $(\lambda x.\ f\ x = g\ x)$ $(at\ x)$
unfolding *eventually-at*
using *assms*(*1*,*2*) **by** *auto*
show $(f \longrightarrow l)$ $(at\ x)$ **by** *fact*
qed

Common case assuming being away from some crucial point like 0.

lemma *Lim-transform-away-within*:
fixes *a b* :: *'a*::*t1-space*
assumes $a \neq b$ **and** $\forall x{\in}S.\ x \neq a \wedge x \neq b \longrightarrow f\ x = g\ x$
and $(f \longrightarrow l)$ $(at\ a\ within\ S)$
shows $(g \longrightarrow l)$ $(at\ a\ within\ S)$
proof (*rule Lim-transform-eventually*)
show $(f \longrightarrow l)$ $(at\ a\ within\ S)$ **by** *fact*
show *eventually* $(\lambda x.\ f\ x = g\ x)$ $(at\ a\ within\ S)$
unfolding *Limits.eventually-within eventually-at-topological*
by (*rule exI* [**where** *x*=− {*b*}], *simp add*: *open-Compl assms*)
qed

lemma *Lim-transform-away-at*:
fixes *a b* :: *'a*::*t1-space*
assumes *ab*: $a{\neq}b$ **and** *fg*: $\forall x.\ x \neq a \wedge x \neq b \longrightarrow f\ x = g\ x$
and *fl*: $(f \longrightarrow l)$ $(at\ a)$
shows $(g \longrightarrow l)$ $(at\ a)$
using *Lim-transform-away-within*[*OF ab*, *of UNIV f g l*] *fg fl*
by (*auto simp add*: *within-UNIV*)

Alternatively, within an open set.

lemma *Lim-transform-within-open*:
assumes *open S* **and** $a \in S$ **and** $\forall x{\in}S.\ x \neq a \longrightarrow f\ x = g\ x$
and $(f \longrightarrow l)$ $(at\ a)$
shows $(g \longrightarrow l)$ $(at\ a)$
proof (*rule Lim-transform-eventually*)
show *eventually* $(\lambda x.\ f\ x = g\ x)$ $(at\ a)$
unfolding *eventually-at-topological*
using *assms*(*1*,*2*,*3*) **by** *auto*
show $(f \longrightarrow l)$ $(at\ a)$ **by** *fact*
qed

A congruence rule allowing us to transform limits assuming not at point.

lemma *Lim-cong-within*:
assumes $\bigwedge x.\ x \neq a \Longrightarrow f\ x = g\ x$
shows $((\lambda x.\ f\ x) \longrightarrow l)\ (at\ a\ within\ S) \longleftrightarrow ((g \longrightarrow l)\ (at\ a\ within\ S))$
unfolding *tendsto-def Limits.eventually-within eventually-at-topological*
using *assms* **by** *simp*

lemma *Lim-cong-at*:
assumes $\bigwedge x.\ x \neq a \Longrightarrow f\ x = g\ x$
shows $((\lambda x.\ f\ x) \longrightarrow l)\ (at\ a) \longleftrightarrow ((g \longrightarrow l)\ (at\ a))$
unfolding *tendsto-def eventually-at-topological*
using *assms* **by** *simp*

Useful lemmas on closure and set of possible sequential limits.

lemma *closure-sequential*:
fixes $l :: 'a$::*metric-space*
shows $l \in closure\ S \longleftrightarrow (\exists x.\ (\forall n.\ x\ n \in S) \wedge (x \longrightarrow l)\ sequentially)$ (**is** *?lhs* $=$ *?rhs*)
proof
assume *?lhs* **moreover**
{ assume $l \in S$
hence *?rhs* **using** *Lim-const*[*of l sequentially*] **by** *auto*
} moreover
{ assume *l islimpt S*
hence *?rhs* **unfolding** *islimpt-sequential* **by** *auto*
} ultimately
show *?rhs* **unfolding** *closure-def* **by** *auto*
next
assume *?rhs*
thus *?lhs* **unfolding** *closure-def* **unfolding** *islimpt-sequential* **by** *auto*
qed

lemma *closed-sequential-limits*:
fixes $S :: 'a$::*metric-space set*
shows *closed S* $\longleftrightarrow (\forall x\ l.\ (\forall n.\ x\ n \in S) \wedge (x \longrightarrow l)\ sequentially \longrightarrow l \in S)$
unfolding *closed-limpt*
using *closure-sequential* [**where** $'a='a$] *closure-closed* [**where** $'a='a$] *closed-limpt* [**where** $'a='a$] *islimpt-sequential* [**where** $'a='a$] *mem-delete* [**where** $'a='a$]
by *metis*

lemma *closure-approachable*:
fixes $S :: 'a$::*metric-space set*
shows $x \in closure\ S \longleftrightarrow (\forall e{>}0.\ \exists y{\in}S.\ dist\ y\ x < e)$
apply (*auto simp add*: *closure-def islimpt-approachable*)
by (*metis dist-self*)

lemma *closed-approachable*:
fixes $S :: 'a$::*metric-space set*

shows *closed S ==> (∀ e>0. ∃ y∈S. dist y x < e) ⟷ x ∈ S*
by (*metis closure-closed closure-approachable*)

Some other lemmas about sequences.

lemma *sequentially-offset*:
assumes *eventually* (*λi. P i*) *sequentially*
shows *eventually* (*λi. P* (*i + k*)) *sequentially*
using *assms* **unfolding** *eventually-sequentially* **by** (*metis trans-le-add1*)

lemma *seq-offset*:
assumes (*f ---> l*) *sequentially*
shows ((*λi. f* (*i + k*)) *---> l*) *sequentially*
using *assms* **unfolding** *tendsto-def*
by *clarify* (*rule sequentially-offset*, *simp*)

lemma *seq-offset-neg*:
(*f ---> l*) *sequentially ==>* ((*λi. f*(*i − k*)) *---> l*) *sequentially*
apply (*rule topological-tendstoI*)
apply (*drule* (*2*) *topological-tendstoD*)
apply (*simp only*: *eventually-sequentially*)
apply (*subgoal-tac ⋀N k* (*n*::*nat*). *N + k <= n ==> N <= n − k*)
apply *metis*
by *arith*

lemma *seq-offset-rev*:
((*λi. f*(*i + k*)) *---> l*) *sequentially ==>* (*f ---> l*) *sequentially*
apply (*rule topological-tendstoI*)
apply (*drule* (*2*) *topological-tendstoD*)
apply (*simp only*: *eventually-sequentially*)
apply (*subgoal-tac ⋀N k* (*n*::*nat*). *N + k <= n ==> N <= n − k ∧* (*n − k*) *+ k = n*)
by *metis arith*

lemma *seq-harmonic*: ((*λn. inverse* (*real n*)) *---> 0*) *sequentially*
proof−
{ fix *e*::*real* **assume** *e>0*
hence *∃ N*::*nat*. *∀ n*::*nat*≥*N*. *inverse* (*real n*) < *e*
using *real-arch-inv*[*of e*] **apply** *auto* **apply**(*rule-tac x*=*n* **in** *exI*)
by (*metis le-imp-inverse-le not-less real-of-nat-gt-zero-cancel-iff real-of-nat-less-iff xt1*(*7*))
}
thus *?thesis* **unfolding** *Lim-sequentially dist-norm* **by** *simp*
qed

16.16 More properties of closed balls.

lemma *closed-cball*: *closed* (*cball x e*)
unfolding *cball-def closed-def*
unfolding *Collect-neg-eq* [*symmetric*] *not-le*

apply (*clarsimp simp add*: *open-dist*, *rename-tac y*)
apply (*rule-tac x*=*dist x y* − *e* **in** *exI*, *clarsimp*)
apply (*rename-tac x'*)
apply (*cut-tac x*=*x* **and** *y*=*x'* **and** *z*=*y* **in** *dist-triangle*)
apply *simp*
done

lemma *open-contains-cball*: *open S* ⟷ (∀ *x*∈*S*. ∃ *e*>*0*. *cball x e* ⊆ *S*)
proof−
{ fix *x* **and** *e*::*real* **assume** *x*∈*S e*>*0 ball x e* ⊆ *S*
hence ∃ *d*>*0*. *cball x d* ⊆ *S* **unfolding** *subset-eq* **by** (*rule-tac x*=*e*/*2* **in** *exI*, *auto*)
} moreover
{ fix *x* **and** *e*::*real* **assume** *x*∈*S e*>*0 cball x e* ⊆ *S*
hence ∃ *d*>*0*. *ball x d* ⊆ *S* **unfolding** *subset-eq* **apply**(*rule-tac x*=*e*/*2* **in** *exI*) **by** *auto*
} ultimately
show *?thesis* **unfolding** *open-contains-ball* **by** *auto*
qed

lemma *open-contains-cball-eq*: *open S* ==> (∀ *x*. *x* ∈ *S* ⟷ (∃ *e*>*0*. *cball x e* ⊆ *S*))
by (*metis open-contains-cball subset-eq order-less-imp-le centre-in-cball mem-def*)

lemma *mem-interior-cball*: *x* ∈ *interior S* ⟷ (∃ *e*>*0*. *cball x e* ⊆ *S*)
apply (*simp add*: *interior-def*, *safe*)
apply (*force simp add*: *open-contains-cball*)
apply (*rule-tac x*=*ball x e* **in** *exI*)
apply (*simp add*: *subset-trans* [*OF ball-subset-cball*])
done

lemma *islimpt-ball*:
fixes *x y* :: *'a*::{*real-normed-vector*,*perfect-space*}
shows *y islimpt ball x e* ⟷ *0* < *e* ∧ *y* ∈ *cball x e* (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
{ assume *e* ≤ *0*
hence ∗:*ball x e* = {} **using** *ball-eq-empty*[*of x e*] **by** *auto*
have *False* **using** ‹*?lhs*› **unfolding** ∗ **using** *islimpt-EMPTY*[*of y*] **by** *auto*
}
hence *e* > *0* **by** (*metis not-less*)
moreover
have *y* ∈ *cball x e* **using** *closed-cball*[*of x e*] *islimpt-subset*[*of y ball x e cball x e*] *ball-subset-cball*[*of x e*] ‹*?lhs*› **unfolding** *closed-limpt* **by** *auto*
ultimately show *?rhs* **by** *auto*
next
assume *?rhs* **hence** *e*>*0* **by** *auto*
{ fix *d*::*real* **assume** *d*>*0*
have ∃ *x'*∈*ball x e*. *x'* ≠ *y* ∧ *dist x' y* < *d*

proof(*cases* $d \leq$ *dist* x y)
case *True* **thus** $\exists x'\in$*ball* x e. $x' \neq y \wedge$ *dist* x' $y < d$
proof(*cases* x=y)
case *True* **hence** *False* **using** ‹$d \leq$ *dist* x y› ‹d>0› **by** *auto*
thus $\exists x'\in$*ball* x e. $x' \neq y \wedge$ *dist* x' $y < d$ **by** *auto*
next
case *False*

have *dist* x (y − (d / (2 ∗ *dist* y x)) ∗$_R$ (y − x))
= *norm* (x − y + (d / (2 ∗ *norm* (y − x))) ∗$_R$ (y − x))
unfolding *mem-cball mem-ball dist-norm diff-diff-eq2 diff-add-eq*[*THEN sym*] **by** *auto*
also have ... = |− 1 + d / (2 ∗ *norm* (x − y))| ∗ *norm* (x − y)
using *scaleR-left-distrib*[*of* − 1 d / (2 ∗ *norm* (y − x)), *THEN sym*, *of* y − x]
unfolding *scaleR-minus-left scaleR-one*
by (*auto simp add*: *norm-minus-commute*)
also have ... = |− *norm* (x − y) + d / 2|
unfolding *abs-mult-pos*[*of norm* (x − y), *OF norm-ge-zero*[*of* x − y]]
unfolding *left-distrib* **using** ‹x≠y›[*unfolded dist-nz*, *unfolded dist-norm*] **by** *auto*
also have ... ≤ e − d/2 **using** ‹$d \leq$ *dist* x y› **and** ‹d>0› **and** ‹*?rhs*› **by**(*auto simp add*: *dist-norm*)
finally have y − (d / (2 ∗ *dist* y x)) ∗$_R$ (y − x) ∈ *ball* x e **using** ‹d>0› **by** *auto*

moreover

have (d / (2∗*dist* y x)) ∗$_R$ (y − x) ≠ 0
using ‹x≠y›[*unfolded dist-nz*] ‹d>0› **unfolding** *scaleR-eq-0-iff* **by** (*auto simp add*: *dist-commute*)
moreover
have *dist* (y − (d / (2 ∗ *dist* y x)) ∗$_R$ (y − x)) $y < d$ **unfolding** *dist-norm* **apply** *simp* **unfolding** *norm-minus-cancel*
using ‹d>0› ‹x≠y›[*unfolded dist-nz*] *dist-commute*[*of* x y]
unfolding *dist-norm* **by** *auto*
ultimately show $\exists x'\in$*ball* x e. $x' \neq y \wedge$ *dist* x' $y < d$ **by** (*rule-tac* x=y − (d / (2∗*dist* y x)) ∗$_R$ (y − x) **in** *bexI*) *auto*
qed
next
case *False* **hence** $d >$ *dist* x y **by** *auto*
show $\exists x'\in$*ball* x e. $x' \neq y \wedge$ *dist* x' $y < d$
proof(*cases* x=y)
case *True*
obtain z **where** **: $z \neq y$ *dist* z $y <$ *min* e d
using *perfect-choose-dist*[*of min* e d y]
using ‹$d > 0$› ‹e>0› **by** *auto*
show $\exists x'\in$*ball* x e. $x' \neq y \wedge$ *dist* x' $y < d$
unfolding ‹$x = y$›

```
      using ‹z ≠ y› **
      by (rule-tac x=z in bexI, auto simp add: dist-commute)
    next
     case False thus ∃x'∈ball x e. x' ≠ y ∧ dist x' y < d
      using ‹d>0› ‹d > dist x y› ‹?rhs› by(rule-tac x=x in bexI, auto)
    qed
  qed }
 thus ?lhs unfolding mem-cball islimpt-approachable mem-ball by auto
qed

lemma closure-ball-lemma:
 fixes x y :: 'a::real-normed-vector
 assumes x ≠ y shows y islimpt ball x (dist x y)
proof (rule islimptI)
 fix T assume y ∈ T open T
 then obtain r where 0 < r ∀z. dist z y < r ⟶ z ∈ T
  unfolding open-dist by fast

 def k ≡ min 1 (r / (2 * dist x y))
 def z ≡ y + scaleR k (x - y)
 have z-def2: z = x + scaleR (1 - k) (y - x)
  unfolding z-def by (simp add: algebra-simps)
 have dist z y < r
  unfolding z-def k-def using ‹0 < r›
  by (simp add: dist-norm min-def)
 hence z ∈ T using ‹∀z. dist z y < r ⟶ z ∈ T› by simp
 have dist x z < dist x y
  unfolding z-def2 dist-norm
  apply (simp add: norm-minus-commute)
  apply (simp only: dist-norm [symmetric])
  apply (subgoal-tac |1 - k| * dist x y < 1 * dist x y, simp)
  apply (rule mult-strict-right-mono)
  apply (simp add: k-def divide-pos-pos zero-less-dist-iff ‹0 < r› ‹x ≠ y›)
  apply (simp add: zero-less-dist-iff ‹x ≠ y›)
  done
 hence z ∈ ball x (dist x y) by simp
 have z ≠ y
  unfolding z-def k-def using ‹x ≠ y› ‹0 < r›
  by (simp add: min-def)
 show ∃z∈ball x (dist x y). z ∈ T ∧ z ≠ y
  using ‹z ∈ ball x (dist x y)› ‹z ∈ T› ‹z ≠ y›
  by fast
qed

lemma closure-ball:
 fixes x :: 'a::real-normed-vector
 shows 0 < e ⟹ closure (ball x e) = cball x e
apply (rule equalityI)
apply (rule closure-minimal)
```

apply (*rule ball-subset-cball*)
apply (*rule closed-cball*)
apply (*rule subsetI*, *rename-tac y*)
apply (*simp add*: *le-less* [**where** *'a=real*])
apply (*erule disjE*)
apply (*rule subsetD* [*OF closure-subset*], *simp*)
apply (*simp add*: *closure-def*)
apply *clarify*
apply (*rule closure-ball-lemma*)
apply (*simp add*: *zero-less-dist-iff*)
done

lemma *interior-cball*:
fixes $x :: 'a::\{real\text{-}normed\text{-}vector, perfect\text{-}space\}$
shows *interior* (*cball x e*) = *ball x e*
proof(*cases* $e \geq 0$)
case *False* **note** *cs* = *this*
from *cs* **have** *ball x e* = {} **using** *ball-empty*[*of e x*] **by** *auto* **moreover**
{ fix *y* **assume** $y \in$ *cball x e*
hence *False* **unfolding** *mem-cball* **using** *dist-nz*[*of x y*] *cs* **by** *auto* **}**
hence *cball x e* = {} **by** *auto*
hence *interior* (*cball x e*) = {} **using** *interior-empty* **by** *auto*
ultimately show *?thesis* **by** *blast*
next
case *True* **note** *cs* = *this*
have *ball x e* $\subseteq$ *cball x e* **using** *ball-subset-cball* **by** *auto* **moreover**
{ fix *S y* **assume** *as*: $S \subseteq$ *cball x e open S* $y \in S$
then obtain *d* **where** $d>0$ **and** $d{:}\forall x'.\ dist\ x'\ y < d \longrightarrow x' \in S$ **unfolding** *open-dist* **by** *blast*

then obtain *xa* **where** *xa-y*: $xa \neq y$ **and** *xa*: *dist xa y* $< d$
using *perfect-choose-dist* [*of d*] **by** *auto*
have $xa \in S$ **using** *d*[*THEN spec*[**where** *x=xa*]] **using** *xa* **by**(*auto simp add*: *dist-commute*)
hence *xa-cball*:$xa \in$ *cball x e* **using** *as*(*1*) **by** *auto*

hence $y \in$ *ball x e* **proof**(*cases* $x = y$)
case *True*
hence $e>0$ **using** *xa-y*[*unfolded dist-nz*] *xa-cball*[*unfolded mem-cball*] **by** (*auto simp add*: *dist-commute*)
thus $y \in$ *ball x e* **using** ‹$x = y$› **by** *simp*
next
case *False*
have *dist* (y + (d / 2 / *dist y x*) $*_R$ ($y - x$)) $y < d$ **unfolding** *dist-norm*
using ‹$d>0$› *norm-ge-zero*[*of* $y - x$] ‹$x \neq y$› **by** *auto*
hence $*$:y + (d / 2 / *dist y x*) $*_R$ ($y - x$) $\in$ *cball x e* **using** *d as*(*1*)[*unfolded subset-eq*] **by** *blast*
have $y - x \neq 0$ **using** ‹$x \neq y$› **by** *auto*

hence $**$:d / $(2 * norm\ (y - x)) > 0$ **unfolding** *zero-less-norm-iff*[*THEN sym*]
using ‹$d>0$› *divide-pos-pos*[*of d 2*norm* $(y - x)$] **by** *auto*

have *dist* $(y + (d / 2 / dist\ y\ x) *_R (y - x))\ x = norm\ (y + (d / (2 * norm\ (y - x))) *_R y - (d / (2 * norm\ (y - x))) *_R x - x)$
by (*auto simp add*: *dist-norm algebra-simps*)
also have $\ldots = norm\ ((1 + d / (2 * norm\ (y - x))) *_R (y - x))$
by (*auto simp add*: *algebra-simps*)
also have $\ldots = |1 + d / (2 * norm\ (y - x))| * norm\ (y - x)$
using $**$ **by** *auto*
also have $\ldots = (dist\ y\ x) + d/2$**using** $**$ **by** (*auto simp add*: *left-distrib dist-norm*)
finally have $e \geq dist\ x\ y + d/2$ **using** $*$[*unfolded mem-cball*] **by** (*auto simp add*: *dist-commute*)
thus $y \in ball\ x\ e$ **unfolding** *mem-ball* **using** ‹$d>0$› **by** *auto*
qed }
hence $\forall S \subseteq cball\ x\ e.\ open\ S \longrightarrow S \subseteq ball\ x\ e$ **by** *auto*
ultimately show *?thesis* **using** *interior-unique*[*of ball x e cball x e*] **using** *open-ball*[*of x e*] **by** *auto*
qed

lemma *frontier-ball*:
fixes a :: $'a$::*real-normed-vector*
shows $0 < e$ ==> $frontier(ball\ a\ e) = \{x.\ dist\ a\ x = e\}$
apply (*simp add*: *frontier-def closure-ball interior-open order-less-imp-le*)
apply (*simp add*: *expand-set-eq*)
by *arith*

lemma *frontier-cball*:
fixes a :: $'a$::{*real-normed-vector*, *perfect-space*}
shows $frontier(cball\ a\ e) = \{x.\ dist\ a\ x = e\}$
apply (*simp add*: *frontier-def interior-cball closed-cball order-less-imp-le*)
apply (*simp add*: *expand-set-eq*)
by *arith*

lemma *cball-eq-empty*: $(cball\ x\ e = \{\}) \longleftrightarrow e < 0$
apply (*simp add*: *expand-set-eq not-le*)
by (*metis zero-le-dist dist-self order-less-le-trans*)
lemma *cball-empty*: $e < 0$ ==> $cball\ x\ e = \{\}$ **by** (*simp add*: *cball-eq-empty*)

lemma *cball-eq-sing*:
fixes x :: $'a$::*perfect-space*
shows $(cball\ x\ e = \{x\}) \longleftrightarrow e = 0$
proof (*rule linorder-cases*)
assume e: $0 < e$
obtain a **where** $a \neq x\ dist\ a\ x < e$
using *perfect-choose-dist* [*OF e*] **by** *auto*
hence $a \neq x\ dist\ x\ a \leq e$ **by** (*auto simp add*: *dist-commute*)

with *e* **show** *?thesis* **by** (*auto simp add*: *expand-set-eq*)
qed *auto*

lemma *cball-sing*:
fixes *x* :: *'a*::*metric-space*
shows *e* = *0* ==> *cball x e* = {*x*}
by (*auto simp add*: *expand-set-eq*)

For points in the interior, localization of limits makes no difference.

lemma *eventually-within-interior*:
assumes *x* ∈ *interior S*
shows *eventually P* (*at x within S*) ⟷ *eventually P* (*at x*) (**is** *?lhs* = *?rhs*)
proof−
from *assms* **obtain** *T* **where** *T*: *open T x* ∈ *T T* ⊆ *S*
unfolding *interior-def* **by** *fast*
{ assume *?lhs*
then obtain *A* **where** *open A x* ∈ *A* ∀ *y*∈*A*. *y* ≠ *x* ⟶ *y* ∈ *S* ⟶ *P y*
unfolding *Limits.eventually-within Limits.eventually-at-topological*
by *auto*
with *T* **have** *open* (*A* ∩ *T*) *x* ∈ *A* ∩ *T* ∀ *y*∈(*A* ∩ *T*). *y* ≠ *x* ⟶ *P y*
by *auto*
then have *?rhs*
unfolding *Limits.eventually-at-topological* **by** *auto*
} moreover
{ assume *?rhs* **hence** *?lhs*
unfolding *Limits.eventually-within*
by (*auto elim*: *eventually-elim1*)
} ultimately
show *?thesis* **..**
qed

lemma *lim-within-interior*:
x ∈ *interior S* ⟹ (*f* −−−> *l*) (*at x within S*) ⟷ (*f* −−−> *l*) (*at x*)
unfolding *tendsto-def* **by** (*simp add*: *eventually-within-interior*)

lemma *netlimit-within-interior*:
fixes *x* :: *'a*::{*perfect-space*, *real-normed-vector*}

assumes *x* ∈ *interior S*
shows *netlimit*(*at x within S*) = *x* (**is** *?lhs* = *?rhs*)
proof−
from *assms* **obtain** *e*::*real* **where** *e*:*e*>*0 ball x e* ⊆ *S* **using** *open-interior*[*of S*]
unfolding *open-contains-ball* **using** *interior-subset*[*of S*] **by** *auto*
hence ¬ *trivial-limit* (*at x within S*) **using** *islimpt-subset*[*of x ball x e S*] **unfolding** *trivial-limit-within islimpt-ball centre-in-cball* **by** *auto*
thus *?thesis* **using** *netlimit-within* **by** *auto*
qed

16.17 Boundedness.

definition
bounded :: *'a*::*metric-space set* ⇒ *bool* **where**
bounded S ⟷ (∃ *x e*. ∀ *y*∈*S*. *dist x y* ≤ *e*)

lemma *bounded-any-center*: *bounded S* ⟷ (∃ *e*. ∀ *y*∈*S*. *dist a y* ≤ *e*)
unfolding *bounded-def*
apply *safe*
apply (*rule-tac x*=*dist a x* + *e* **in** *exI*, *clarify*)
apply (*drule* (*1*) *bspec*)
apply (*erule order-trans* [*OF dist-triangle add-left-mono*])
apply *auto*
done

lemma *bounded-iff*: *bounded S* ⟷ (∃ *a*. ∀ *x*∈*S*. *norm x* ≤ *a*)
unfolding *bounded-any-center* [**where** *a*=*0*]
by (*simp add*: *dist-norm*)

lemma *bounded-empty*[*simp*]: *bounded* {} **by** (*simp add*: *bounded-def*)
lemma *bounded-subset*: *bounded T* ⟹ *S* ⊆ *T* ==> *bounded S*
by (*metis bounded-def subset-eq*)

lemma *bounded-interior*[*intro*]: *bounded S* ==> *bounded*(*interior S*)
by (*metis bounded-subset interior-subset*)

lemma *bounded-closure*[*intro*]: **assumes** *bounded S* **shows** *bounded*(*closure S*)
proof−
from *assms* **obtain** *x* **and** *a* **where** *a*: ∀ *y*∈*S*. *dist x y* ≤ *a* **unfolding** *bounded-def*
by *auto*
{ fix *y* **assume** *y* ∈ *closure S*
then obtain *f* **where** *f*: ∀ *n*. *f n* ∈ *S* (*f* −−−> *y*) *sequentially*
unfolding *closure-sequential* **by** *auto*
have ∀ *n*. *f n* ∈ *S* ⟶ *dist x* (*f n*) ≤ *a* **using** *a* **by** *simp*
hence *eventually* (λ*n*. *dist x* (*f n*) ≤ *a*) *sequentially*
by (*rule eventually-mono*, *simp add*: *f*(*1*))
have *dist x y* ≤ *a*
apply (*rule Lim-dist-ubound* [*of sequentially f*])
apply (*rule trivial-limit-sequentially*)
apply (*rule f*(*2*))
apply *fact*
done
}
thus *?thesis* **unfolding** *bounded-def* **by** *auto*
qed

lemma *bounded-cball*[*simp*,*intro*]: *bounded* (*cball x e*)
apply (*simp add*: *bounded-def*)
apply (*rule-tac x*=*x* **in** *exI*)
apply (*rule-tac x*=*e* **in** *exI*)

apply *auto*
done

lemma *bounded-ball*[*simp*,*intro*]: *bounded*(*ball x e*)
by (*metis ball-subset-cball bounded-cball bounded-subset*)

lemma *finite-imp-bounded*[*intro*]:
fixes *S* :: *'a*::*metric-space set* **assumes** *finite S* **shows** *bounded S*
proof−
{ fix *a* **and** *F* :: *'a set* **assume** *as*:*bounded F*
then obtain *x e* **where** ∀ *y*∈*F*. *dist x y* ≤ *e* **unfolding** *bounded-def* **by** *auto*
hence ∀ *y*∈(*insert a F*). *dist x y* ≤ *max e* (*dist x a*) **by** *auto*
hence *bounded* (*insert a F*) **unfolding** *bounded-def* **by** (*intro exI*)
}
thus *?thesis* **using** *finite-induct*[*of S bounded*] **using** *bounded-empty assms* **by** *auto*
qed

lemma *bounded-Un*[*simp*]: *bounded* (*S* ∪ *T*) ⟷ *bounded S* ∧ *bounded T*
apply (*auto simp add*: *bounded-def*)
apply (*rename-tac x y r s*)
apply (*rule-tac x*=*x* **in** *exI*)
apply (*rule-tac x*=*max r* (*dist x y* + *s*) **in** *exI*)
apply (*rule ballI*, *rename-tac z*, *safe*)
apply (*drule* (*1*) *bspec*, *simp*)
apply (*drule* (*1*) *bspec*)
apply (*rule min-max.le-supI2*)
apply (*erule order-trans* [*OF dist-triangle add-left-mono*])
done

lemma *bounded-Union*[*intro*]: *finite F* ⟹ (∀ *S*∈*F*. *bounded S*) ⟹ *bounded*(⋃ *F*)
by (*induct rule*: *finite-induct*[*of F*], *auto*)

lemma *bounded-pos*: *bounded S* ⟷ (∃ *b*>*0*. ∀ *x*∈ *S*. *norm x* <= *b*)
apply (*simp add*: *bounded-iff*)
apply (*subgoal-tac* ⋀*x* (*y*::*real*). *0* < *1* + *abs y* ∧ (*x* <= *y* ⟶ *x* <= *1* + *abs y*))
by *metis arith*

lemma *bounded-Int*[*intro*]: *bounded S* ∨ *bounded T* ⟹ *bounded* (*S* ∩ *T*)
by (*metis Int-lower1 Int-lower2 bounded-subset*)

lemma *bounded-diff*[*intro*]: *bounded S* ==> *bounded* (*S* − *T*)
apply (*metis Diff-subset bounded-subset*)
done

lemma *bounded-insert*[*intro*]:*bounded*(*insert x S*) ⟷ *bounded S*
by (*metis Diff-cancel Un-empty-right Un-insert-right bounded-Un bounded-subset finite.emptyI finite-imp-bounded infinite-remove subset-insertI*)

lemma *not-bounded-UNIV*[*simp*, *intro*]:
 ¬ *bounded* (*UNIV* :: *'a*::{*real-normed-vector*, *perfect-space*} *set*)
proof(*auto simp add*: *bounded-pos not-le*)
 obtain *x* :: *'a* **where** *x* ≠ *0*
  using *perfect-choose-dist* [*OF zero-less-one*] **by** *fast*
 fix *b*::*real* **assume** *b*: *b* >*0*
 have *b1*: *b* +*1* ≥ *0* **using** *b* **by** *simp*
 with ‹*x* ≠ *0*› **have** *b* < *norm* (*scaleR* (*b* + *1*) (*sgn x*))
  by (*simp add*: *norm-sgn*)
 then show ∃ *x*::*'a*. *b* < *norm x* **..**
qed

lemma *bounded-linear-image*:
 assumes *bounded S bounded-linear f*
 shows *bounded*(*f* ‘ *S*)
proof−
 from *assms*(*1*) **obtain** *b* **where** *b*:*b*>*0* ∀ *x*∈*S*. *norm x* ≤ *b* **unfolding** *bounded-pos*
by *auto*
 from *assms*(*2*) **obtain** *B* **where** *B*:*B*>*0* ∀ *x*. *norm* (*f x*) ≤ *B* ∗ *norm x* **using**
bounded-linear.pos-bounded **by** (*auto simp add*: *mult-ac*)
 { fix *x* **assume** *x*∈*S*
  hence *norm x* ≤ *b* **using** *b* **by** *auto*
  hence *norm* (*f x*) ≤ *B* ∗ *b* **using** *B*(*2*) **apply**(*erule-tac x*=*x* **in** *allE*)
   by (*metis B*(*1*) *B*(*2*) *order-trans mult-le-cancel-left-pos*)
 }
 thus *?thesis* **unfolding** *bounded-pos* **apply**(*rule-tac x*=*b*∗*B* **in** *exI*)
  using *b B mult-pos-pos* [*of b B*] **by** (*auto simp add*: *mult-commute*)
qed

lemma *bounded-scaling*:
 fixes *S* :: *'a*::*real-normed-vector set*
 shows *bounded S* ⟹ *bounded* ((λ*x*. *c* ∗*R x*) ‘ *S*)
 apply (*rule bounded-linear-image*, *assumption*)
 apply (*rule scaleR.bounded-linear-right*)
 done

lemma *bounded-translation*:
 fixes *S* :: *'a*::*real-normed-vector set*
 assumes *bounded S* **shows** *bounded* ((λ*x*. *a* + *x*) ‘ *S*)
proof−
 from *assms* **obtain** *b* **where** *b*:*b*>*0* ∀ *x*∈*S*. *norm x* ≤ *b* **unfolding** *bounded-pos*
by *auto*
 { fix *x* **assume** *x*∈*S*
  hence *norm* (*a* + *x*) ≤ *b* + *norm a* **using** *norm-triangle-ineq*[*of a x*] *b* **by**
auto
 }
 thus *?thesis* **unfolding** *bounded-pos* **using** *norm-ge-zero*[*of a*] *b*(*1*) **using** *add-strict-increasing*[*of*
b 0 norm a]

by (*auto intro*!: *add exI*[*of* - *b* + *norm a*])
qed

Some theorems on sups and infs using the notion "bounded".

lemma *bounded-real*:
fixes *S* :: *real set*
shows *bounded S* ⟷ (∃ *a*. ∀ *x*∈*S*. *abs x* <= *a*)
by (*simp add*: *bounded-iff*)

lemma *bounded-has-Sup*:
fixes *S* :: *real set*
assumes *bounded S S* ≠ {}
shows ∀ *x*∈*S*. *x* <= *Sup S* **and** ∀ *b*. (∀ *x*∈*S*. *x* <= *b*) ⟶ *Sup S* <= *b*
proof
fix *x* **assume** *x*∈*S*
thus *x* ≤ *Sup S*
by (*metis SupInf*.*Sup-upper abs-le-D1 assms*(*1*) *bounded-real*)
next
show ∀ *b*. (∀ *x*∈*S*. *x* ≤ *b*) ⟶ *Sup S* ≤ *b* **using** *assms*
by (*metis SupInf*.*Sup-least*)
qed

lemma *Sup-insert*:
fixes *S* :: *real set*
shows *bounded S* ==> *Sup*(*insert x S*) = (*if S* = {} *then x else max x* (*Sup S*))

by *auto* (*metis Int-absorb Sup-insert-nonempty assms bounded-has-Sup*(*1*) *disjoint-iff-not-equal*)

lemma *Sup-insert-finite*:
fixes *S* :: *real set*
shows *finite S* ⟹ *Sup*(*insert x S*) = (*if S* = {} *then x else max x* (*Sup S*))
apply (*rule Sup-insert*)
apply (*rule finite-imp-bounded*)
by *simp*

lemma *bounded-has-Inf*:
fixes *S* :: *real set*
assumes *bounded S S* ≠ {}
shows ∀ *x*∈*S*. *x* >= *Inf S* **and** ∀ *b*. (∀ *x*∈*S*. *x* >= *b*) ⟶ *Inf S* >= *b*
proof
fix *x* **assume** *x*∈*S*
from *assms*(*1*) **obtain** *a* **where** *a*:∀ *x*∈*S*. |*x*| ≤ *a* **unfolding** *bounded-real* **by**
auto
thus *x* ≥ *Inf S* **using** ‹*x*∈*S*›
by (*metis Inf-lower-EX abs-le-D2 minus-le-iff*)
next
show ∀ *b*. (∀ *x*∈*S*. *x* >= *b*) ⟶ *Inf S* ≥ *b* **using** *assms*
by (*metis SupInf*.*Inf-greatest*)

qed

lemma *Inf-insert*:
fixes S :: *real set*
shows *bounded* S ==> *Inf*(*insert* x S) = (*if* $S = \{\}$ *then* x *else min* x (*Inf* S))
by *auto* (*metis Int-absorb Inf-insert-nonempty bounded-has-Inf*(*1*) *disjoint-iff-not-equal*)

lemma *Inf-insert-finite*:
fixes S :: *real set*
shows *finite* S ==> *Inf*(*insert* x S) = (*if* $S = \{\}$ *then* x *else min* x (*Inf* S))
by (*rule Inf-insert*, *rule finite-imp-bounded*, *simp*)

lemma *real-isGlb-unique*: [| *isGlb* R S x; *isGlb* R S y |] ==> x = (y::*real*)
apply (*frule isGlb-isLb*)
apply (*frule-tac* x = y **in** *isGlb-isLb*)
apply (*blast intro*!: *order-antisym dest*!: *isGlb-le-isLb*)
done

16.18 Equivalent versions of compactness

16.18.1 Sequential compactness

definition
compact :: 'a::*metric-space set* ⇒ *bool* **where**
compact S ⟷
(∀f. (∀n. f n ∈ S) ⟶
(∃l∈S. ∃r. *subseq* r ∧ ((f *o* r) ---> l) *sequentially*))

A metric space (or topological vector space) is said to have the Heine-Borel property if every closed and bounded subset is compact.

class *heine-borel* =
assumes *bounded-imp-convergent-subsequence*:
bounded s ⟹ ∀n. f n ∈ s
⟹ ∃l r. *subseq* r ∧ ((f ∘ r) ---> l) *sequentially*

lemma *bounded-closed-imp-compact*:
fixes s::'a::*heine-borel set*
assumes *bounded* s **and** *closed* s **shows** *compact* s
proof (*unfold compact-def*, *clarify*)
fix f :: *nat* ⇒ 'a **assume** f: ∀n. f n ∈ s
obtain l r **where** r: *subseq* r **and** l: ((f ∘ r) ---> l) *sequentially*
using *bounded-imp-convergent-subsequence* [*OF* ‹*bounded* s› ‹∀n. f n ∈ s›] **by** *auto*
from f **have** fr: ∀n. (f ∘ r) n ∈ s **by** *simp*
have l ∈ s **using** ‹*closed* s› fr l
unfolding *closed-sequential-limits* **by** *blast*
show ∃l∈s. ∃r. *subseq* r ∧ ((f ∘ r) ---> l) *sequentially*
using ‹l ∈ s› r l **by** *blast*

qed

lemma *subseq-bigger*: **assumes** *subseq r* **shows** $n \leq r\ n$
proof(*induct n*)
  show $0 \leq r\ 0$ **by** *auto*
next
  fix *n* **assume** $n \leq r\ n$
  moreover have $r\ n < r\ (Suc\ n)$
    using *assms* [*unfolded subseq-def*] **by** *auto*
  ultimately show $Suc\ n \leq r\ (Suc\ n)$ **by** *auto*
qed

lemma *eventually-subseq*:
  assumes *r*: *subseq r*
  shows *eventually P sequentially* $\Longrightarrow$ *eventually* $(\lambda n.\ P\ (r\ n))$ *sequentially*
unfolding *eventually-sequentially*
by (*metis subseq-bigger* [*OF r*] *le-trans*)

lemma *lim-subseq*:
  subseq r $\Longrightarrow$ $(s \longrightarrow l)$ *sequentially* $\Longrightarrow$ $((s\ o\ r) \longrightarrow l)$ *sequentially*
unfolding *tendsto-def eventually-sequentially o-def*
by (*metis subseq-bigger le-trans*)

lemma *num-Axiom*: $EX!\ g.\ g\ 0 = e \wedge (\forall n.\ g\ (Suc\ n) = f\ n\ (g\ n))$
  unfolding *Ex1-def*
  apply (*rule-tac x*=*nat-rec e f* **in** *exI*)
  apply (*rule conjI*)+
apply (*rule def-nat-rec-0*, *simp*)
apply (*rule allI*, *rule def-nat-rec-Suc*, *simp*)
apply (*rule allI*, *rule impI*, *rule ext*)
apply (*erule conjE*)
apply (*induct-tac x*)
apply *simp*
apply (*erule-tac x*=*n* **in** *allE*)
apply (*simp*)
done

lemma *convergent-bounded-increasing*: **fixes** *s* ::*nat*⇒*real*
  assumes *incseq s* **and** $\forall n.\ abs(s\ n) \leq b$
  shows $\exists\ l.\ \forall e{::}real{>}0.\ \exists\ N.\ \forall n \geq N.\ \ abs(s\ n - l) < e$
proof−
  have *isUb UNIV* (*range s*) *b* **using** *assms*(*2*) **and** *abs-le-D1* **unfolding** *isUb-def* **and** *setle-def* **by** *auto*
  then obtain *t* **where** *t*:*isLub UNIV* (*range s*) *t* **using** *reals-complete*[*of range s*] **by** *auto*
  { fix *e*::*real* **assume** *e*>*0* **and** *as*:$\forall N.\ \exists n{\geq}N.\ \neg\ |s\ n - t| < e$
    { fix *n*::*nat*
      obtain *N* **where** *N*≥*n* **and** *n*:$|s\ N - t| \geq e$ **using** *as*[*THEN spec*[**where** *x*=*n*]] **by** *auto*

have $t \geq s\ N$ **using** *isLub-isUb*[*OF* t, *unfolded isUb-def setle-def*] **by** *auto*
with n **have** $s\ N \leq t - e$ **using** ‹$e>0$› **by** *auto*
hence $s\ n \leq t - e$ **using** *assms*(*1*)[*unfolded incseq-def*, *THEN spec*[**where** $x=n$], *THEN spec*[**where** $x=N$]] **using** ‹$n \leq N$› **by** *auto* **}**
hence *isUb UNIV* (*range s*) $(t - e)$ **unfolding** *isUb-def* **and** *setle-def* **by** *auto*
hence *False* **using** *isLub-le-isUb*[*OF* t, *of* $t - e$] **and** ‹$e>0$› **by** *auto* **}**
thus *?thesis* **by** *blast*
qed

lemma *convergent-bounded-monotone*: **fixes** s::*nat* $\Rightarrow$ *real*
assumes $\forall\, n.\ abs(s\ n) \leq b$ **and** *monoseq s*
shows $\exists\, l.\ \forall\, e$::*real*$>0.\ \exists\, N.\ \forall\, n \geq N.\ abs(s\ n - l) < e$
using *convergent-bounded-increasing*[*of s b*] *assms* **using** *convergent-bounded-increasing*[*of* $\lambda n.\ -s\ n$ *b*]
unfolding *monoseq-def incseq-def*
apply *auto* **unfolding** *minus-add-distrib*[*THEN sym*, *unfolded diff-minus*[*THEN sym*]]
unfolding *abs-minus-cancel* **by**(*rule-tac* $x=-l$ **in** *exI*)*auto*

lemma *compact-real-lemma*:
assumes $\forall\, n$::*nat*. $abs(s\ n) \leq b$
shows $\exists\,(l$::*real*$)\ r.$ *subseq* $r \wedge ((s \circ r) \longrightarrow l)$ *sequentially*
proof−
obtain r **where** r:*subseq r monoseq* $(\lambda n.\ s\ (r\ n))$
using *seq-monosub*[*of s*] **by** *auto*
thus *?thesis* **using** *convergent-bounded-monotone*[*of* $\lambda n.\ s\ (r\ n)$ *b*] **and** *assms*
unfolding *tendsto-iff dist-norm eventually-sequentially* **by** *auto*
qed

instance *real* :: *heine-borel*
proof
fix s :: *real set* **and** f :: *nat* $\Rightarrow$ *real*
assume s: *bounded s* **and** f: $\forall\, n.\ f\ n \in s$
then obtain b **where** b: $\forall\, n.\ abs\ (f\ n) \leq b$
unfolding *bounded-iff* **by** *auto*
obtain l :: *real* **and** r :: *nat* $\Rightarrow$ *nat* **where**
r: *subseq r* **and** l: $((f \circ r) \longrightarrow l)$ *sequentially*
using *compact-real-lemma* [*OF b*] **by** *auto*
thus $\exists\, l\ r.$ *subseq* $r \wedge ((f \circ r) \longrightarrow l)$ *sequentially*
by *auto*
qed

lemma *bounded-component*: *bounded s* $\Longrightarrow$ *bounded* $((\lambda x.\ x\ \$\ i)\ `\ s)$
unfolding *bounded-def*
apply *clarify*
apply (*rule-tac* $x=x\ \$\ i$ **in** *exI*)
apply (*rule-tac* $x=e$ **in** *exI*)
apply *clarify*

apply (*rule order-trans* [*OF dist-nth-le*], *simp*)
done

lemma *compact-lemma*:
fixes $f :: nat \Rightarrow 'a::heine\text{-}borel\ \hat{}\ 'n$
assumes *bounded s* **and** $\forall n.\ f\ n \in s$
shows $\forall d.$
$\exists l\ r.\ subseq\ r \wedge$
$(\forall e{>}0.\ eventually\ (\lambda n.\ \forall i{\in}d.\ dist\ (f\ (r\ n)\ \$\ i)\ (l\ \$\ i) < e)\ sequentially)$
proof
fix $d::'n\ set$ **have** *finite d* **by** *simp*
thus $\exists l::'a\ \hat{}\ 'n.\ \exists r.\ subseq\ r \wedge$
$(\forall e{>}0.\ eventually\ (\lambda n.\ \forall i{\in}d.\ dist\ (f\ (r\ n)\ \$\ i)\ (l\ \$\ i) < e)\ sequentially)$
proof(*induct d*) **case** *empty* **thus** *?case* **unfolding** *subseq-def* **by** *auto*
next case (*insert k d*)
have *s'*: *bounded* $((\lambda x.\ x\ \$\ k)\ `\ s)$ **using** ‹*bounded s*› **by** (*rule bounded-component*)
obtain *l1*::$'a\hat{}'n$ **and** *r1* **where** *r1*:*subseq r1* **and** *lr1*:$\forall e{>}0.\ eventually\ (\lambda n.\ \forall i{\in}d.\ dist\ (f\ (r1\ n)\ \$\ i)\ (l1\ \$\ i) < e)\ sequentially$
using *insert*(*3*) **by** *auto*
have *f'*: $\forall n.\ f\ (r1\ n)\ \$\ k \in (\lambda x.\ x\ \$\ k)\ `\ s$ **using** ‹$\forall n.\ f\ n \in s$› **by** *simp*
obtain *l2 r2* **where** *r2*:*subseq r2* **and** *lr2*:$((\lambda i.\ f\ (r1\ (r2\ i))\ \$\ k) \longrightarrow l2)\ sequentially$
using *bounded-imp-convergent-subsequence*[*OF s' f'*] **unfolding** *o-def* **by** *auto*
def $r \equiv r1 \circ r2$ **have** *r*:*subseq r*
using *r1* **and** *r2* **unfolding** *r-def o-def subseq-def* **by** *auto*
moreover
def $l \equiv (\chi\ i.\ if\ i = k\ then\ l2\ else\ l1\$i)::'a\hat{}'n$
{ **fix** *e*::*real* **assume** *e*>*0*
from *lr1* ‹*e*>*0*› **have** *N1*:$eventually\ (\lambda n.\ \forall i{\in}d.\ dist\ (f\ (r1\ n)\ \$\ i)\ (l1\ \$\ i) < e)\ sequentially$ **by** *blast*
from *lr2* ‹*e*>*0*› **have** *N2*:$eventually\ (\lambda n.\ dist\ (f\ (r1\ (r2\ n))\ \$\ k)\ l2 < e)\ sequentially$ **by** (*rule tendstoD*)
from *r2 N1* **have** *N1'*: $eventually\ (\lambda n.\ \forall i{\in}d.\ dist\ (f\ (r1\ (r2\ n))\ \$\ i)\ (l1\ \$\ i) < e)\ sequentially$
by (*rule eventually-subseq*)
have $eventually\ (\lambda n.\ \forall i{\in}(insert\ k\ d).\ dist\ (f\ (r\ n)\ \$\ i)\ (l\ \$\ i) < e)\ sequentially$
using *N1' N2* **by** (*rule eventually-elim2*, *simp add*: *l-def r-def*)
}
ultimately show *?case* **by** *auto*
qed
qed

instance *cart* :: (*heine-borel*, *finite*) *heine-borel*
proof
fix $s :: ('a\ \hat{}\ 'b)\ set$ **and** $f :: nat \Rightarrow 'a\ \hat{}\ 'b$
assume *s*: *bounded s* **and** *f*: $\forall n.\ f\ n \in s$
then obtain *l r* **where** *r*: *subseq r*
and *l*: $\forall e{>}0.\ eventually\ (\lambda n.\ \forall i{\in}UNIV.\ dist\ (f\ (r\ n)\ \$\ i)\ (l\ \$\ i) < e)\ sequentially$

```
    using compact-lemma [OF s f] by blast
  let ?d = UNIV::'b set
  { fix e::real assume e>0
    hence 0 < e / (real-of-nat (card ?d))
      using zero-less-card-finite using divide-pos-pos[of e, of real-of-nat (card ?d)]
by auto
    with l have eventually (λn. ∀ i. dist (f (r n) $ i) (l $ i) < e / (real-of-nat
(card ?d))) sequentially
      by simp
    moreover
    { fix n assume n: ∀ i. dist (f (r n) $ i) (l $ i) < e / (real-of-nat (card ?d))
      have dist (f (r n)) l ≤ (∑ i∈?d. dist (f (r n) $ i) (l $ i))
        unfolding dist-vector-def using zero-le-dist by (rule setL2-le-setsum)
      also have ... < (∑ i∈?d. e / (real-of-nat (card ?d)))
        by (rule setsum-strict-mono) (simp-all add: n)
      finally have dist (f (r n)) l < e by simp
    }
    ultimately have eventually (λn. dist (f (r n)) l < e) sequentially
      by (rule eventually-elim1)
  }
  hence *:((f ∘ r) ---> l) sequentially unfolding o-def tendsto-iff by simp
  with r show ∃ l r. subseq r ∧ ((f ∘ r) ---> l) sequentially by auto
qed

lemma bounded-fst: bounded s ⟹ bounded (fst ' s)
unfolding bounded-def
apply clarify
apply (rule-tac x=a in exI)
apply (rule-tac x=e in exI)
apply clarsimp
apply (drule (1) bspec)
apply (simp add: dist-Pair-Pair)
apply (erule order-trans [OF real-sqrt-sum-squares-ge1])
done

lemma bounded-snd: bounded s ⟹ bounded (snd ' s)
unfolding bounded-def
apply clarify
apply (rule-tac x=b in exI)
apply (rule-tac x=e in exI)
apply clarsimp
apply (drule (1) bspec)
apply (simp add: dist-Pair-Pair)
apply (erule order-trans [OF real-sqrt-sum-squares-ge2])
done

instance * :: (heine-borel, heine-borel) heine-borel
proof
  fix s :: ('a * 'b) set and f :: nat ⇒ 'a * 'b
```

assume *s*: *bounded s* **and** *f*: $\forall\, n.\ f\ n \in s$
from *s* **have** *s1*: *bounded* (*fst* ' *s*) **by** (*rule bounded-fst*)
from *f* **have** *f1*: $\forall\, n.$ *fst* (*f n*) $\in$ *fst* ' *s* **by** *simp*
obtain *l1 r1* **where** *r1*: *subseq r1*
and *l1*: ((λ*n*. *fst* (*f* (*r1 n*))) ---> *l1*) *sequentially*
using *bounded-imp-convergent-subsequence* [*OF s1 f1*]
unfolding *o-def* **by** *fast*
from *s* **have** *s2*: *bounded* (*snd* ' *s*) **by** (*rule bounded-snd*)
from *f* **have** *f2*: $\forall\, n.$ *snd* (*f* (*r1 n*)) $\in$ *snd* ' *s* **by** *simp*
obtain *l2 r2* **where** *r2*: *subseq r2*
and *l2*: ((λ*n*. *snd* (*f* (*r1* (*r2 n*)))) ---> *l2*) *sequentially*
using *bounded-imp-convergent-subsequence* [*OF s2 f2*]
unfolding *o-def* **by** *fast*
have *l1'*: ((λ*n*. *fst* (*f* (*r1* (*r2 n*)))) ---> *l1*) *sequentially*
using *lim-subseq* [*OF r2 l1*] **unfolding** *o-def* **.**
have *l*: ((*f* ∘ (*r1* ∘ *r2*)) ---> (*l1*, *l2*)) *sequentially*
using *tendsto-Pair* [*OF l1' l2*] **unfolding** *o-def* **by** *simp*
have *r*: *subseq* (*r1* ∘ *r2*)
using *r1 r2* **unfolding** *subseq-def* **by** *simp*
show $\exists\, l\ r.$ *subseq r* $\wedge$ ((*f* ∘ *r*) ---> *l*) *sequentially*
using *l r* **by** *fast*
qed

16.18.2 Completeness

lemma *cauchy-def*:
Cauchy s $\longleftrightarrow$ $(\forall\, e{>}0.\ \exists\, N.\ \forall\, m\ n.\ m \geq N \wedge n \geq N$ --> *dist*(*s m*)(*s n*) $< e)$
unfolding *Cauchy-def* **by** *blast*

definition
complete :: 'a::*metric-space set* ⇒ *bool* **where**
complete s $\longleftrightarrow$ $(\forall\, f.\ (\forall\, n.\ f\ n \in s) \wedge$ *Cauchy f*
--> $(\exists\, l \in s.\ (f$ ---> *l*) *sequentially*))

lemma *cauchy*: *Cauchy s* $\longleftrightarrow$ $(\forall\, e{>}0.\exists\ N$::*nat*. $\forall\, n{\geq}N.$ *dist*(*s n*)(*s N*) $< e)$ (**is** *?lhs* = *?rhs*)
proof−
{ assume *?rhs*
{ fix *e*::*real*
assume *e>0*
with ‹*?rhs*› **obtain** *N* **where** *N*:$\forall\, n{\geq}N.$ *dist* (*s n*) (*s N*) $< e/2$
by (*erule-tac x*=*e*/*2* **in** *allE*) *auto*
{ fix *n m*
assume *nm*:$N \leq m \wedge N \leq n$
hence *dist* (*s m*) (*s n*) $< e$ **using** *N*
using *dist-triangle-half-l*[*of s m s N e s n*]
by *blast*
}
hence $\exists\, N.\ \forall\, m\ n.\ N \leq m \wedge N \leq n \longrightarrow$ *dist* (*s m*) (*s n*) $< e$

```
      by blast
    }
    hence ?lhs
      unfolding cauchy-def
      by blast
  }
  thus ?thesis
    unfolding cauchy-def
    using dist-triangle-half-l
    by blast
qed

lemma convergent-imp-cauchy:
 (s ---> l) sequentially ==> Cauchy s
proof(simp only: cauchy-def, rule, rule)
  fix e::real assume e>0 (s ---> l) sequentially
  then obtain N::nat where N:∀ n≥N. dist (s n) l < e/2 unfolding Lim-sequentially
by(erule-tac x=e/2 in allE) auto
  thus ∃ N. ∀ m n. N ≤ m ∧ N ≤ n ⟶ dist (s m) (s n) < e using dist-triangle-half-l[of
- l e -] by (rule-tac x=N in exI) auto
qed

lemma cauchy-imp-bounded: assumes Cauchy s shows bounded (range s)
proof−
  from assms obtain N::nat where ∀ m n. N ≤ m ∧ N ≤ n ⟶ dist (s m) (s
n) < 1 unfolding cauchy-def apply(erule-tac x= 1 in allE) by auto
  hence N:∀ n. N ≤ n ⟶ dist (s N) (s n) < 1 by auto
  moreover
  have bounded (s ' {0..N}) using finite-imp-bounded[of s ' {1..N}] by auto
  then obtain a where a:∀ x∈s ' {0..N}. dist (s N) x ≤ a
    unfolding bounded-any-center [where a=s N] by auto
  ultimately show ?thesis
    unfolding bounded-any-center [where a=s N]
    apply(rule-tac x=max a 1 in exI) apply auto
    apply(erule-tac x=y in allE) apply(erule-tac x=y in ballE) by auto
qed

lemma compact-imp-complete: assumes compact s shows complete s
proof−
  { fix f assume as: (∀ n::nat. f n ∈ s) Cauchy f
    from as(1) obtain l r where lr: l∈s subseq r ((f ∘ r) ---> l) sequentially
using assms unfolding compact-def by blast

    note lr' = subseq-bigger [OF lr(2)]

    { fix e::real assume e>0
      from as(2) obtain N where N:∀ m n. N ≤ m ∧ N ≤ n ⟶ dist (f m) (f
n) < e/2 unfolding cauchy-def using ‹e>0› apply (erule-tac x=e/2 in allE)
by auto
```

from *lr*(*3*)[*unfolded Lim-sequentially*, *THEN spec*[**where** *x*=*e*/*2*]] **obtain** *M* **where** *M*:∀ *n*≥*M*. *dist* ((*f* ∘ *r*) *n*) *l* < *e*/*2* **using** ‹*e*>*0*› **by** *auto*
{ **fix** *n*::*nat* **assume** *n*:*n* ≥ *max N M*
have *dist* ((*f* ∘ *r*) *n*) *l* < *e*/*2* **using** *n M* **by** *auto*
moreover have *r n* ≥ *N* **using** *lr*'[*of n*] *n* **by** *auto*
hence *dist* (*f n*) ((*f* ∘ *r*) *n*) < *e* / *2* **using** *N* **using** *n* **by** *auto*
ultimately have *dist* (*f n*) *l* < *e* **using** *dist-triangle-half-r*[*of f* (*r n*) *f n e l*] **by** (*auto simp add*: *dist-commute*) }
hence ∃ *N*. ∀ *n*≥*N*. *dist* (*f n*) *l* < *e* **by** *blast* }
hence ∃ *l*∈*s*. (*f* ---> *l*) *sequentially* **using** ‹*l*∈*s*› **unfolding** *Lim-sequentially* **by** *auto* }
thus *?thesis* **unfolding** *complete-def* **by** *auto*
qed

instance *heine-borel* < *complete-space*
proof
fix *f* :: *nat* ⇒ *'a* **assume** *Cauchy f*
hence *bounded* (*range f*)
by (*rule cauchy-imp-bounded*)
hence *compact* (*closure* (*range f*))
using *bounded-closed-imp-compact* [*of closure* (*range f*)] **by** *auto*
hence *complete* (*closure* (*range f*))
by (*rule compact-imp-complete*)
moreover have ∀ *n*. *f n* ∈ *closure* (*range f*)
using *closure-subset* [*of range f*] **by** *auto*
ultimately have ∃ *l*∈*closure* (*range f*). (*f* ---> *l*) *sequentially*
using ‹*Cauchy f*› **unfolding** *complete-def* **by** *auto*
then show *convergent f*
unfolding *convergent-def* **by** *auto*
qed

lemma *complete-univ*: *complete* (*UNIV* :: *'a*::*complete-space set*)
proof(*simp add*: *complete-def*, *rule*, *rule*)
fix *f* :: *nat* ⇒ *'a* **assume** *Cauchy f*
hence *convergent f* **by** (*rule Cauchy-convergent*)
thus ∃ *l*. *f* ----> *l* **unfolding** *convergent-def* **.**
qed

lemma *complete-imp-closed*: **assumes** *complete s* **shows** *closed s*
proof −
{ **fix** *x* **assume** *x islimpt s*
then obtain *f* **where** *f*: ∀ *n*. *f n* ∈ *s* − {*x*} (*f* ---> *x*) *sequentially*
unfolding *islimpt-sequential* **by** *auto*
then obtain *l* **where** *l*: *l*∈*s* (*f* ---> *l*) *sequentially*
using ‹*complete s*›[*unfolded complete-def*] **using** *convergent-imp-cauchy*[*of f x*] **by** *auto*
hence *x* ∈ *s* **using** *Lim-unique*[*of sequentially f l x*] *trivial-limit-sequentially f*(*2*) **by** *auto*
}

thus *closed s* **unfolding** *closed-limpt* **by** *auto*
qed

lemma *complete-eq-closed*:
fixes *s* :: *'a*::*complete-space set*
shows *complete s* ⟷ *closed s* (**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **thus** *?rhs* **by** (*rule complete-imp-closed*)
next
assume *?rhs*
{ fix *f* **assume** *as*:∀ *n*::*nat*. *f n* ∈ *s Cauchy f*
then obtain *l* **where** (*f* −−−> *l*) *sequentially* **using** *complete-univ*[*unfolded complete-def*, *THEN spec*[**where** *x*=*f*]] **by** *auto*
hence ∃ *l*∈*s*. (*f* −−−> *l*) *sequentially* **using** ‹*?rhs*›[*unfolded closed-sequential-limits*, *THEN spec*[**where** *x*=*f*], *THEN spec*[**where** *x*=*l*]] **using** *as*(*1*) **by** *auto* **}**
thus *?lhs* **unfolding** *complete-def* **by** *auto*
qed

lemma *convergent-eq-cauchy*:
fixes *s* :: *nat* ⇒ *'a*::*complete-space*
shows (∃ *l*. (*s* −−−> *l*) *sequentially*) ⟷ *Cauchy s* (**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then obtain** *l* **where** (*s* −−−> *l*) *sequentially* **by** *auto*
thus *?rhs* **using** *convergent-imp-cauchy* **by** *auto*
next
assume *?rhs* **thus** *?lhs* **using** *complete-univ*[*unfolded complete-def*, *THEN spec*[**where** *x*=*s*]] **by** *auto*
qed

lemma *convergent-imp-bounded*:
fixes *s* :: *nat* ⇒ *'a*::*metric-space*
shows (*s* −−−> *l*) *sequentially* ==> *bounded* (*s* ‘ (*UNIV*::(*nat set*)))
using *convergent-imp-cauchy*[*of s*]
using *cauchy-imp-bounded*[*of s*]
unfolding *image-def*
by *auto*

16.18.3 Total boundedness

fun *helper-1*::(*'a*::*metric-space set*) ⇒ *real* ⇒ *nat* ⇒ *'a* **where**
helper-1 s e n = (*SOME y*::*'a*. *y* ∈ *s* ∧ (∀ *m*<*n*. ¬ (*dist* (*helper-1 s e m*) *y* < *e*)))
declare *helper-1.simps*[*simp del*]

lemma *compact-imp-totally-bounded*:
assumes *compact s*
shows ∀ *e*>*0*. ∃ *k*. *finite k* ∧ *k* ⊆ *s* ∧ *s* ⊆ (⋃ ((λ*x*. *ball x e*) ‘ *k*))
proof(*rule*, *rule*, *rule ccontr*)
fix *e*::*real* **assume** *e*>*0* **and** *assm*:¬ (∃ *k*. *finite k* ∧ *k* ⊆ *s* ∧ *s* ⊆ ⋃(λ*x*. *ball x*

e) ‘ *k*)
def *x* ≡ *helper-1 s e*
{ fix *n*
have *x n* ∈ *s* ∧ (∀ *m*<*n*. ¬ *dist* (*x m*) (*x n*) < *e*)
proof(*induct-tac rule*:*nat-less-induct*)
fix *n* **def** *Q* ≡ (λ*y*. *y* ∈ *s* ∧ (∀ *m*<*n*. ¬ *dist* (*x m*) *y* < *e*))
assume *as*:∀ *m*<*n*. *x m* ∈ *s* ∧ (∀ *ma*<*m*. ¬ *dist* (*x ma*) (*x m*) < *e*)
have ¬ *s* ⊆ (⋃ *x*∈*x* ‘ {*0*..<*n*}. *ball x e*) **using** *assm* **apply** *simp* **apply**(*erule-tac x*=*x* ‘ {*0* ..< *n*} **in** *allE*) **using** *as* **by** *auto*
then obtain *z* **where** *z*:*z*∈*s z* ∉ (⋃ *x*∈*x* ‘ {*0*..<*n*}. *ball x e*) **unfolding** *subset-eq* **by** *auto*
have *Q* (*x n*) **unfolding** *x-def* **and** *helper-1.simps*[*of s e n*]
apply(*rule someI2*[**where** *a*=*z*]) **unfolding** *x-def*[*symmetric*] **and** *Q-def* **using** *z* **by** *auto*
thus *x n* ∈ *s* ∧ (∀ *m*<*n*. ¬ *dist* (*x m*) (*x n*) < *e*) **unfolding** *Q-def* **by** *auto*
qed }
hence ∀ *n*::*nat*. *x n* ∈ *s* **and** *x*:∀ *n*. ∀ *m* < *n*. ¬ (*dist* (*x m*) (*x n*) < *e*) **by** *blast*+
then obtain *l r* **where** *l*∈*s* **and** *r*:*subseq r* **and** ((*x* ∘ *r*) −−−> *l*) *sequentially* **using** *assms*(*1*)[*unfolded compact-def*, *THEN spec*[**where** *x*=*x*]] **by** *auto*
from *this*(*3*) **have** *Cauchy* (*x* ∘ *r*) **using** *convergent-imp-cauchy* **by** *auto*
then obtain *N*::*nat* **where** *N*:∀ *m n*. *N* ≤ *m* ∧ *N* ≤ *n* ⟶ *dist* ((*x* ∘ *r*) *m*) ((*x* ∘ *r*) *n*) < *e* **unfolding** *cauchy-def* **using** ‹*e*>*0*› **by** *auto*
show *False*
using *N*[*THEN spec*[**where** *x*=*N*], *THEN spec*[**where** *x*=*N*+*1*]]
using *r*[*unfolded subseq-def*, *THEN spec*[**where** *x*=*N*], *THEN spec*[**where** *x*=*N*+*1*]]
using *x*[*THEN spec*[**where** *x*=*r* (*N*+*1*)], *THEN spec*[**where** *x*=*r* (*N*)]] **by** *auto*
qed

16.18.4 Heine-Borel theorem

Following Burkill & Burkill vol. 2.

lemma *heine-borel-lemma*: **fixes** *s*::′*a*::*metric-space set*
assumes *compact s* *s* ⊆ (⋃ *t*) ∀ *b* ∈ *t*. *open b*
shows ∃ *e*>*0*. ∀ *x* ∈ *s*. ∃ *b* ∈ *t*. *ball x e* ⊆ *b*
proof(*rule ccontr*)
assume ¬ (∃ *e*>*0*. ∀ *x*∈*s*. ∃ *b*∈*t*. *ball x e* ⊆ *b*)
hence *cont*:∀ *e*>*0*. ∃ *x*∈*s*. ∀ *xa*∈*t*. ¬ (*ball x e* ⊆ *xa*) **by** *auto*
{ fix *n*::*nat*
have *1* / *real* (*n* + *1*) > *0* **by** *auto*
hence ∃ *x*. *x*∈*s* ∧ (∀ *xa*∈*t*. ¬ (*ball x* (*inverse* (*real* (*n*+*1*))) ⊆ *xa*)) **using** *cont* **unfolding** *Bex-def* **by** *auto* **}**
hence ∀ *n*::*nat*. ∃ *x*. *x* ∈ *s* ∧ (∀ *xa*∈*t*. ¬ *ball x* (*inverse* (*real* (*n* + *1*))) ⊆ *xa*) **by** *auto*
then obtain *f* **where** *f*:∀ *n*::*nat*. *f n* ∈ *s* ∧ (∀ *xa*∈*t*. ¬ *ball* (*f n*) (*inverse* (*real* (*n* + *1*))) ⊆ *xa*)
using *choice*[*of* λ*n*::*nat*. λ*x*. *x*∈*s* ∧ (∀ *xa*∈*t*. ¬ *ball x* (*inverse* (*real* (*n* + *1*))) ⊆ *xa*)] **by** *auto*

then obtain $l\ r$ **where** l:$l \in s$ **and** r:*subseq* r **and** lr:$((f \circ r) \longrightarrow l)$ *sequentially*
using *assms*(*1*)[*unfolded compact-def*, *THEN spec*[**where** x=f]] **by** *auto*

obtain b **where** $l \in b$ $b \in t$ **using** *assms*(*2*) **and** l **by** *auto*
then obtain e **where** $e>0$ **and** e:$\forall z.\ dist\ z\ l < e \longrightarrow z \in b$
using *assms*(*3*)[*THEN bspec*[**where** x=b]] **unfolding** *open-dist* **by** *auto*

then obtain *N1* **where** *N1*:$\forall n \geq N1.\ dist\ ((f \circ r)\ n)\ l < e\ /\ 2$
using *lr*[*unfolded Lim-sequentially*, *THEN spec*[**where** x=$e/2$]] **by** *auto*

obtain *N2*::*nat* **where** *N2*:$N2>0$ *inverse* (*real N2*) $<$ *e /2* **using** *real-arch-inv*[*of e/2*] **and** ‹$e>0$› **by** *auto*
have *N2'*:*inverse* (*real* (*r* (*N1* + *N2*) +*1*)) $<$ *e*/*2*
apply(*rule order-less-trans*) **apply**(*rule less-imp-inverse-less*) **using** *N2*
using *subseq-bigger*[*OF r*, *of N1* + *N2*] **by** *auto*

def $x \equiv (f\ (r\ (N1 + N2)))$
have x:$\neg$ *ball x* (*inverse* (*real* (*r* (*N1* + *N2*) + *1*))) $\subseteq$ *b* **unfolding** *x-def*
using *f*[*THEN spec*[**where** x=$r\ (N1 + N2)$]] **using** ‹$b \in t$› **by** *auto*
have $\exists y \in ball\ x\ (inverse\ (real\ (r\ (N1 + N2) + 1)))$. $y \notin b$ **apply**(*rule ccontr*) **using** *x* **by** *auto*
then obtain *y* **where** *y*:$y \in ball\ x\ (inverse\ (real\ (r\ (N1 + N2) + 1)))\ y \notin b$ **by** *auto*

have *dist x l* $<$ *e*/*2* **using** *N1* **unfolding** *x-def o-def* **by** *auto*
hence *dist y l* $<$ *e* **using** *y N2'* **using** *dist-triangle*[*of y l x*]**by** (*auto simp add*:*dist-commute*)

thus *False* **using** *e* **and** ‹$y \notin b$› **by** *auto*
qed

lemma *compact-imp-heine-borel*: *compact s* ==> $(\forall f.\ (\forall t \in f.\ open\ t) \wedge s \subseteq (\bigcup f)$
$\longrightarrow (\exists f'.\ f' \subseteq f \wedge finite\ f' \wedge s \subseteq (\bigcup f')))$
proof *clarify*
fix *f* **assume** *compact s* $\forall t \in f.\ open\ t\ s \subseteq \bigcup f$
then obtain *e*::*real* **where** $e>0$ **and** $\forall x \in s.\ \exists b \in f.\ ball\ x\ e \subseteq b$ **using** *heine-borel-lemma*[*of s f*] **by** *auto*
hence $\forall x \in s.\ \exists b.\ b \in f \wedge ball\ x\ e \subseteq b$ **by** *auto*
hence $\exists bb.\ \forall x \in s.\ bb\ x \in f \wedge ball\ x\ e \subseteq bb\ x$ **using** *bchoice*[*of s* $\lambda x\ b.\ b \in f \wedge ball\ x\ e \subseteq b$] **by** *auto*
then obtain *bb* **where** *bb*:$\forall x \in s.\ (bb\ x) \in f \wedge ball\ x\ e \subseteq (bb\ x)$ **by** *blast*

from ‹*compact s*› **have** $\exists\ k.\ finite\ k \wedge k \subseteq s \wedge s \subseteq \bigcup(\lambda x.\ ball\ x\ e)\ `\ k$ **using** *compact-imp-totally-bounded*[*of s*] ‹$e>0$› **by** *auto*
then obtain *k* **where** *k*:*finite k* $k \subseteq s$ $s \subseteq \bigcup(\lambda x.\ ball\ x\ e)\ `\ k$ **by** *auto*

have *finite* (*bb* ` *k*) **using** *k*(*1*) **by** *auto*

moreover
{ fix x **assume** $x{\in}s$
hence $x{\in}\bigcup(\lambda x.\ ball\ x\ e)$ ‘ k **using** $k(3)$ **unfolding** *subset-eq* **by** *auto*
hence $\exists X{\in}bb$ ‘ $k.\ x \in X$ **using** $bb\ k(2)$ **by** *blast*
hence $x \in \bigcup(bb$ ‘ $k)$ **using** *Union-iff*[*of x bb* ‘ k] **by** *auto*
}
ultimately show $\exists f'{\subseteq}f.\ finite\ f' \wedge s \subseteq \bigcup f'$ **using** $bb\ k(2)$ **by** (*rule-tac* $x{=}bb$ ‘ k **in** *exI*) *auto*
qed

16.18.5 Bolzano-Weierstrass property

lemma *heine-borel-imp-bolzano-weierstrass*:
assumes $\forall f.\ (\forall t \in f.\ open\ t) \wedge s \subseteq (\bigcup f) \longrightarrow (\exists f'.\ f' \subseteq f \wedge finite\ f' \wedge s \subseteq (\bigcup f'))$
infinite t $t \subseteq s$
shows $\exists x \in s.\ x\ islimpt\ t$
proof(*rule ccontr*)
assume $\neg\ (\exists x \in s.\ x\ islimpt\ t)$
then obtain f **where** $f{:}\forall x{\in}s.\ x \in f\ x \wedge open\ (f\ x) \wedge (\forall y{\in}t.\ y \in f\ x \longrightarrow y = x)$ **unfolding** *islimpt-def*
using *bchoice*[*of s* $\lambda\ x\ T.\ x \in T \wedge open\ T \wedge (\forall y{\in}t.\ y \in T \longrightarrow y = x)$] **by** *auto*
obtain g **where** $g{:}g{\subseteq}\{t.\ \exists x.\ x \in s \wedge t = f\ x\}\ finite\ g\ s \subseteq \bigcup g$
using *assms*(*1*)[*THEN spec*[**where** $x{=}\{t.\ \exists x.\ x{\in}s \wedge t = f\ x\}$]] **using** f **by** *auto*
from $g(1,3)$ **have** $g'{:}\forall x{\in}g.\ \exists xa \in s.\ x = f\ xa$ **by** *auto*
{ fix $x\ y$ **assume** $x{\in}t\ y{\in}t\ f\ x = f\ y$
hence $x \in f\ x\ \ y \in f\ x \longrightarrow y = x$ **using** f[*THEN bspec*[**where** $x{=}x$]] **and** ‹$t{\subseteq}s$› **by** *auto*
hence $x = y$ **using** ‹$f\ x = f\ y$› **and** f[*THEN bspec*[**where** $x{=}y$]] **and** ‹$y{\in}t$› **and** ‹$t{\subseteq}s$› **by** *auto* **}**
hence *inj-on f t* **unfolding** *inj-on-def* **by** *simp*
hence *infinite* $(f$ ‘ $t)$ **using** *assms*(*2*) **using** *finite-imageD* **by** *auto*
moreover
{ fix x **assume** $x{\in}t\ f\ x \notin g$
from $g(3)$ *assms*(*3*) ‹$x{\in}t$› **obtain** h **where** $h{\in}g$ **and** $x{\in}h$ **by** *auto*
then obtain y **where** $y{\in}s\ h = f\ y$ **using** g'[*THEN bspec*[**where** $x{=}h$]] **by** *auto*
hence $y = x$ **using** f[*THEN bspec*[**where** $x{=}y$]] **and** ‹$x{\in}t$› **and** ‹$x{\in}h$›[*unfolded* ‹$h = f\ y$›] **by** *auto*
hence *False* **using** ‹$f\ x \notin g$› ‹$h{\in}g$› **unfolding** ‹$h = f\ y$› **by** *auto* **}**
hence f ‘ $t \subseteq g$ **by** *auto*
ultimately show *False* **using** $g(2)$ **using** *finite-subset* **by** *auto*
qed

16.18.6 Complete the chain of compactness variants

primrec *helper-2*::(*real* ⇒ *'a*::*metric-space*) ⇒ *nat* ⇒ *'a* **where**
helper-2 beyond 0 = beyond 0 |

helper-2 beyond (Suc n) = beyond (dist undefined (helper-2 beyond n) + 1)

lemma *bolzano-weierstrass-imp-bounded*: **fixes** *s*::*'a*::*metric-space set*
assumes $\forall t.$ *infinite* $t \wedge t \subseteq s$ $--> (\exists x \in s.$ *x islimpt t*)
shows *bounded s*
proof(*rule ccontr*)
assume ¬ *bounded s*
then obtain *beyond* **where** $\forall a.$ *beyond a* $\in s \wedge$ ¬ *dist undefined* (*beyond a*) $\leq a$
unfolding *bounded-any-center* [**where** *a*=*undefined*]
apply *simp* **using** *choice*[*of* $\lambda a\ x.$ *x*$\in$*s* $\wedge$ ¬ *dist undefined x* $\leq a$] **by** *auto*
hence *beyond*:$\bigwedge a.$ *beyond a* $\in s \bigwedge a.$ *dist undefined* (*beyond a*) $> a$
unfolding *linorder-not-le* **by** *auto*
def $x \equiv$ *helper-2 beyond*

{ fix *m n* ::*nat* **assume** *m*<*n*
hence *dist undefined* (*x m*) + *1* < *dist undefined* (*x n*)
proof(*induct n*)
case *0* **thus** *?case* **by** *auto*
next
case (*Suc n*)
have ∗:*dist undefined* (*x n*) + *1* < *dist undefined* (*x* (*Suc n*))
unfolding *x-def* **and** *helper-2.simps*
using *beyond*(*2*)[*of dist undefined* (*helper-2 beyond n*) + *1*] **by** *auto*
thus *?case* **proof**(*cases m* < *n*)
case *True* **thus** *?thesis* **using** *Suc* **and** ∗ **by** *auto*
next
case *False* **hence** *m* = *n* **using** *Suc*(*2*) **by** *auto*
thus *?thesis* **using** ∗ **by** *auto*
qed
qed **} note** ∗ = *this*
{ fix *m n* ::*nat* **assume** *m*≠*n*
have *1* < *dist* (*x m*) (*x n*)
proof(*cases m*<*n*)
case *True*
hence *1* < *dist undefined* (*x n*) − *dist undefined* (*x m*) **using** ∗[*of m n*] **by** *auto*
thus *?thesis* **using** *dist-triangle* [*of undefined x n x m*] **by** *arith*
next
case *False* **hence** *n*<*m* **using** ‹*m*≠*n*› **by** *auto*
hence *1* < *dist undefined* (*x m*) − *dist undefined* (*x n*) **using** ∗[*of n m*] **by** *auto*
thus *?thesis* **using** *dist-triangle2* [*of undefined x m x n*] **by** *arith*
qed **} note** ∗∗ = *this*
{ fix *a b* **assume** *x a* = *x b a* ≠ *b*
hence *False* **using** ∗∗[*of a b*] **by** *auto* **}**
hence *inj x* **unfolding** *inj-on-def* **by** *auto*
moreover
{ fix *n*::*nat*
have *x n* ∈ *s*

proof(*cases n = 0*)
case *True* **thus** *?thesis* **unfolding** *x-def* **using** *beyond* **by** *auto*
next
case *False* **then obtain** *z* **where** *n = Suc z* **using** *not0-implies-Suc* **by** *auto*
thus *?thesis* **unfolding** *x-def* **using** *beyond* **by** *auto*
qed **}**
ultimately have *infinite (range x) ∧ range x ⊆ s* **unfolding** *x-def* **using** *range-inj-infinite*[*of helper-2 beyond*] **using** *beyond*(*1*) **by** *auto*

then obtain *l* **where** *l∈s* **and** *l*:*l islimpt range x* **using** *assms*[*THEN spec*[**where** *x=range x*]] **by** *auto*
then obtain *y* **where** *x y ≠ l* **and** *y*:*dist (x y) l < 1/2* **unfolding** *islimpt-approachable* **apply**(*erule-tac x=1/2* **in** *allE*) **by** *auto*
then obtain *z* **where** *x z ≠ l* **and** *z*:*dist (x z) l < dist (x y) l* **using** *l*[*unfolded islimpt-approachable*, *THEN spec*[**where** *x=dist (x y) l*]]
unfolding *dist-nz* **by** *auto*
show *False* **using** *y* **and** *z* **and** *dist-triangle-half-l*[*of x y l 1 x z*] **and** *∗∗*[*of y z*] **by** *auto*
qed

lemma *sequence-infinite-lemma*:
fixes *l* :: *'a*::*metric-space*
assumes *∀ n*::*nat*. (*f n ≠ l*) (*f −−−> l*) *sequentially*
shows *infinite* (*range f*)
proof
let *?A* = (*λx*. *dist x l*) ‘ *range f*
assume *finite* (*range f*)
hence *∗∗*:*finite ?A ?A ≠ {}* **by** *auto*
obtain *k* **where** *k*:*dist* (*f k*) *l* = *Min ?A* **using** *Min-in*[*OF ∗∗*] **by** *auto*
have *0 < Min ?A* **using** *assms*(*1*) **unfolding** *dist-nz* **unfolding** *Min-gr-iff*[*OF ∗∗*] **by** *auto*
then obtain *N* **where** *dist* (*f N*) *l < Min ?A* **using** *assms*(*2*)[*unfolded Lim-sequentially*, *THEN spec*[**where** *x=Min ?A*]] **by** *auto*
moreover have *dist* (*f N*) *l* ∈ *?A* **by** *auto*
ultimately show *False* **using** *Min-le*[*OF ∗∗*(*1*), *of dist* (*f N*) *l*] **by** *auto*
qed

lemma *sequence-unique-limpt*:
fixes *l* :: *'a*::*metric-space*
assumes *∀ n*::*nat*. (*f n ≠ l*) (*f −−−> l*) *sequentially* *l′ islimpt* (*range f*)
shows *l′ = l*
proof(*rule ccontr*)
def *e ≡ dist l′ l*
assume *l′ ≠ l* **hence** *e>0* **unfolding** *dist-nz e-def* **by** *auto*
then obtain *N*::*nat* **where** *N*:*∀ n≥N*. *dist* (*f n*) *l < e / 2*
using *assms*(*2*)[*unfolded Lim-sequentially*, *THEN spec*[**where** *x=e/2*]] **by** *auto*
def *d ≡ Min* (*insert* (*e/2*) ((*λn*. *if dist* (*f n*) *l′ = 0 then e/2 else dist* (*f n*) *l′*) ‘ {*0* .. *N*}))
have *d>0* **using** ‹*e>0*› **unfolding** *d-def e-def* **using** *zero-le-dist*[*of - l′*, *unfolded*

order-le-less] **by** *auto*
obtain *k* **where** *k*:*f k* ≠ *l'* *dist* (*f k*) *l'* < *d* **using** ‹*d*>*0*› **and** *assms*(*3*)[*unfolded islimpt-approachable*, *THEN spec*[**where** *x*=*d*]] **by** *auto*
have *k*≥*N* **using** *k*(*1*)[*unfolded dist-nz*] **using** *k*(*2*)[*unfolded d-def*]
by (*force simp del*: *Min.insert-idem*)
hence *dist l' l* < *e* **using** *N*[*THEN spec*[**where** *x*=*k*]] **using** *k*(*2*)[*unfolded d-def*] **and** *dist-triangle-half-r*[*of f k l' e l*] **by** (*auto simp del*: *Min.insert-idem*)
thus *False* **unfolding** *e-def* **by** *auto*
qed

lemma *bolzano-weierstrass-imp-closed*:
fixes *s* :: *'a*::*metric-space set*
assumes ∀ *t*. *infinite t* ∧ *t* ⊆ *s* −−> (∃ *x* ∈ *s*. *x islimpt t*)
shows *closed s*
proof−
{ fix *x l* **assume** *as*: ∀ *n*::*nat*. *x n* ∈ *s* (*x* −−−> *l*) *sequentially*
hence *l* ∈ *s*
proof(*cases* ∀ *n*. *x n* ≠ *l*)
case *False* **thus** *l*∈*s* **using** *as*(*1*) **by** *auto*
next
case *True* **note** *cas* = *this*
with *as*(*2*) **have** *infinite* (*range x*) **using** *sequence-infinite-lemma*[*of x l*] **by** *auto*
then obtain *l'* **where** *l'*∈*s* *l' islimpt* (*range x*) **using** *assms*[*THEN spec*[**where** *x*=*range x*]] *as*(*1*) **by** *auto*
thus *l*∈*s* **using** *sequence-unique-limpt*[*of x l l'*] **using** *as cas* **by** *auto*
qed **}**
thus *?thesis* **unfolding** *closed-sequential-limits* **by** *fast*
qed

Hence express everything as an equivalence.

lemma *compact-eq-heine-borel*:
fixes *s* :: *'a*::*heine-borel set*
shows *compact s* ⟷
(∀ *f*. (∀ *t* ∈ *f*. *open t*) ∧ *s* ⊆ (⋃ *f*)
−−> (∃ *f'*. *f'* ⊆ *f* ∧ *finite f'* ∧ *s* ⊆ (⋃ *f'*))) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **thus** *?rhs* **using** *compact-imp-heine-borel*[*of s*] **by** *blast*
next
assume *?rhs*
hence ∀ *t*. *infinite t* ∧ *t* ⊆ *s* ⟶ (∃ *x*∈*s*. *x islimpt t*)
by (*blast intro*: *heine-borel-imp-bolzano-weierstrass*[*of s*])
thus *?lhs* **using** *bolzano-weierstrass-imp-bounded*[*of s*] *bolzano-weierstrass-imp-closed*[*of s*] *bounded-closed-imp-compact*[*of s*] **by** *blast*
qed

lemma *compact-eq-bolzano-weierstrass*:
fixes *s* :: *'a*::*heine-borel set*
shows *compact s* ⟷ (∀ *t*. *infinite t* ∧ *t* ⊆ *s* −−> (∃ *x* ∈ *s*. *x islimpt t*)) (**is**

?lhs = ?rhs)
proof
assume *?lhs* **thus** *?rhs* **unfolding** *compact-eq-heine-borel* **using** *heine-borel-imp-bolzano-weierstrass*[*of s*] **by** *auto*
next
assume *?rhs* **thus** *?lhs* **using** *bolzano-weierstrass-imp-bounded bolzano-weierstrass-imp-closed bounded-closed-imp-compact* **by** *auto*
qed

lemma *compact-eq-bounded-closed*:
fixes *s* :: *'a*::*heine-borel set*
shows *compact s* ⟷ *bounded s* ∧ *closed s* (**is** *?lhs = ?rhs*)
proof
assume *?lhs* **thus** *?rhs* **unfolding** *compact-eq-bolzano-weierstrass* **using** *bolzano-weierstrass-imp-bounded bolzano-weierstrass-imp-closed* **by** *auto*
next
assume *?rhs* **thus** *?lhs* **using** *bounded-closed-imp-compact* **by** *auto*
qed

lemma *compact-imp-bounded*:
fixes *s* :: *'a*::*metric-space set*
shows *compact s* ==> *bounded s*
proof −
assume *compact s*
hence ∀*f*. (∀ *t*∈*f*. *open t*) ∧ *s* ⊆ ⋃*f* ⟶ (∃*f'*⊆*f*. *finite f'* ∧ *s* ⊆ ⋃*f'*)
by (*rule compact-imp-heine-borel*)
hence ∀ *t*. *infinite t* ∧ *t* ⊆ *s* ⟶ (∃ *x* ∈ *s*. *x islimpt t*)
using *heine-borel-imp-bolzano-weierstrass*[*of s*] **by** *auto*
thus *bounded s*
by (*rule bolzano-weierstrass-imp-bounded*)
qed

lemma *compact-imp-closed*:
fixes *s* :: *'a*::*metric-space set*
shows *compact s* ==> *closed s*
proof −
assume *compact s*
hence ∀*f*. (∀ *t*∈*f*. *open t*) ∧ *s* ⊆ ⋃*f* ⟶ (∃*f'*⊆*f*. *finite f'* ∧ *s* ⊆ ⋃*f'*)
by (*rule compact-imp-heine-borel*)
hence ∀ *t*. *infinite t* ∧ *t* ⊆ *s* ⟶ (∃ *x* ∈ *s*. *x islimpt t*)
using *heine-borel-imp-bolzano-weierstrass*[*of s*] **by** *auto*
thus *closed s*
by (*rule bolzano-weierstrass-imp-closed*)
qed

In particular, some common special cases.

lemma *compact-empty*[*simp*]:
compact {}
unfolding *compact-def*

by *simp*

lemma *compact-union*[*intro*]:
fixes *s t* :: *'a*::*heine-borel set*
shows *compact s* ⟹ *compact t* ==> *compact* (*s* ∪ *t*)
unfolding *compact-eq-bounded-closed*
using *bounded-Un*[*of s t*]
using *closed-Un*[*of s t*]
by *simp*

lemma *compact-inter*[*intro*]:
fixes *s t* :: *'a*::*heine-borel set*
shows *compact s* ⟹ *compact t* ==> *compact* (*s* ∩ *t*)
unfolding *compact-eq-bounded-closed*
using *bounded-Int*[*of s t*]
using *closed-Int*[*of s t*]
by *simp*

lemma *compact-inter-closed*[*intro*]:
fixes *s t* :: *'a*::*heine-borel set*
shows *compact s* ⟹ *closed t* ==> *compact* (*s* ∩ *t*)
unfolding *compact-eq-bounded-closed*
using *closed-Int*[*of s t*]
using *bounded-subset*[*of s* ∩ *t s*]
by *blast*

lemma *closed-inter-compact*[*intro*]:
fixes *s t* :: *'a*::*heine-borel set*
shows *closed s* ⟹ *compact t* ==> *compact* (*s* ∩ *t*)
proof−
assume *closed s compact t*
moreover
have *s* ∩ *t* = *t* ∩ *s* **by** *auto* **ultimately**
show *?thesis*
using *compact-inter-closed*[*of t s*]
by *auto*
qed

lemma *finite-imp-compact*:
fixes *s* :: *'a*::*heine-borel set*
shows *finite s* ==> *compact s*
unfolding *compact-eq-bounded-closed*
using *finite-imp-closed finite-imp-bounded*
by *blast*

lemma *compact-sing* [*simp*]: *compact* {*a*}

unfolding *compact-def o-def subseq-def*
by (*auto simp add*: *tendsto-const*)

lemma *compact-cball*[*simp*]:
fixes x :: *'a*::*heine-borel*
shows *compact*(*cball x e*)
using *compact-eq-bounded-closed bounded-cball closed-cball*
by *blast*

lemma *compact-frontier-bounded*[*intro*]:
fixes s :: *'a*::*heine-borel set*
shows *bounded s* ==> *compact*(*frontier s*)
unfolding *frontier-def*
using *compact-eq-bounded-closed*
by *blast*

lemma *compact-frontier*[*intro*]:
fixes s :: *'a*::*heine-borel set*
shows *compact s* ==> *compact* (*frontier s*)
using *compact-eq-bounded-closed compact-frontier-bounded*
by *blast*

lemma *frontier-subset-compact*:
fixes s :: *'a*::*heine-borel set*
shows *compact s* ==> *frontier s* $\subseteq$ *s*
using *frontier-subset-closed compact-eq-bounded-closed*
by *blast*

lemma *open-delete*:
fixes s :: *'a*::*t1-space set*
shows *open s* $\Longrightarrow$ *open* ($s - \{x\}$)
by (*simp add*: *open-Diff*)

Finite intersection property. I could make it an equivalence in fact.

lemma *compact-imp-fip*:
fixes s :: *'a*::*heine-borel set*
assumes *compact s* $\forall t \in f$. *closed t*
$\forall f'$. *finite* $f' \wedge f' \subseteq f$ --> $(s \cap (\bigcap f') \neq \{\})$
shows $s \cap (\bigcap f) \neq \{\}$
proof
assume *as*:$s \cap (\bigcap f) = \{\}$
hence $s \subseteq \bigcup$ *uminus ' f* **by** *auto*
moreover have *Ball* (*uminus ' f*) *open* **using** *open-Diff closed-Diff* **using** *assms*(*2*) **by** *auto*
ultimately obtain f' **where** f':$f' \subseteq$ *uminus ' f finite* f' $s \subseteq \bigcup f'$ **using** *assms*(*1*)[*unfolded compact-eq-heine-borel*, *THEN spec*[**where** x=($\lambda t.\ -\ t$) *' f*]]
by *auto*
hence *finite* (*uminus ' f'*) $\wedge$ *uminus ' f'* $\subseteq f$ **by**(*auto simp add*: *Diff-Diff-Int*)
hence $s \cap \bigcap$ *uminus ' f'* $\neq \{\}$ **using** *assms*(*3*)[*THEN spec*[**where** x=*uminus '*

f']] **by** *auto*
thus *False* **using** *f'*(*3*) **unfolding** *subset-eq* **and** *Union-iff* **by** *blast*
qed

16.19 Bounded closed nest property (proof does not use Heine-Borel).

lemma *bounded-closed-nest*:
assumes ∀ *n*. *closed*(*s n*) ∀ *n*. (*s n* ≠ {})
(∀ *m n*. *m* ≤ *n* −−> *s n* ⊆ *s m*) *bounded*(*s* 0)
shows ∃ *a*::'*a*::*heine-borel*. ∀ *n*::*nat*. *a* ∈ *s*(*n*)
proof−
from *assms*(*2*) **obtain** *x* **where** *x*:∀ *n*::*nat*. *x n* ∈ *s n* **using** *choice*[*of* λ*n x*. *x*∈ *s n*] **by** *auto*
from *assms*(*4*,*1*) **have** ∗:*compact* (*s* 0) **using** *bounded-closed-imp-compact*[*of s* 0] **by** *auto*

then obtain *l r* **where** *lr*:*l*∈*s* 0 *subseq r* ((*x* ∘ *r*) −−−> *l*) *sequentially*
unfolding *compact-def* **apply**(*erule-tac x*=*x* **in** *allE*) **using** *x* **using** *assms*(*3*)
by *blast*

{ fix *n*::*nat*
{ fix *e*::*real* **assume** *e*>*0*
with *lr*(*3*) **obtain** *N* **where** *N*:∀ *m*≥*N*. *dist* ((*x* ∘ *r*) *m*) *l* < *e* **unfolding** *Lim-sequentially* **by** *auto*
hence *dist* ((*x* ∘ *r*) (*max N n*)) *l* < *e* **by** *auto*
moreover
have *r* (*max N n*) ≥ *n* **using** *lr*(*2*) **using** *subseq-bigger*[*of r max N n*] **by** *auto*
hence (*x* ∘ *r*) (*max N n*) ∈ *s n*
using *x* **apply**(*erule-tac x*=*n* **in** *allE*)
using *x* **apply**(*erule-tac x*=*r* (*max N n*) **in** *allE*)
using *assms*(*3*) **apply**(*erule-tac x*=*n* **in** *allE*)**apply**(*erule-tac x*=*r* (*max N n*) **in** *allE*) **by** *auto*
ultimately have ∃ *y*∈*s n*. *dist y l* < *e* **by** *auto*
}
hence *l* ∈ *s n* **using** *closed-approachable*[*of s n l*] *assms*(*1*) **by** *blast*
}
thus *?thesis* **by** *auto*
qed

Decreasing case does not even need compactness, just completeness.

lemma *decreasing-closed-nest*:
assumes ∀ *n*. *closed*(*s n*)
∀ *n*. (*s n* ≠ {})
∀ *m n*. *m* ≤ *n* −−> *s n* ⊆ *s m*
∀ *e*>*0*. ∃ *n*. ∀ *x* ∈ (*s n*). ∀ *y* ∈ (*s n*). *dist x y* < *e*
shows ∃ *a*::'*a*::*heine-borel*. ∀ *n*::*nat*. *a* ∈ *s n*
proof−

have $\forall n.\ \exists\ x.\ x{\in}s\ n$ **using** *assms(2)* **by** *auto*
hence $\exists t.\ \forall n.\ t\ n \in s\ n$ **using** *choice*[*of* $\lambda\ n\ x.\ x \in s\ n$] **by** *auto*
then obtain t **where** t: $\forall n.\ t\ n \in s\ n$ **by** *auto*
{ fix e::*real* **assume** $e{>}0$
then obtain N **where** N:$\forall x{\in}s\ N.\ \forall y{\in}s\ N.\ dist\ x\ y < e$ **using** *assms(4)* **by** *auto*
{ fix $m\ n$ ::*nat* **assume** $N \leq m \wedge N \leq n$
hence $t\ m \in s\ N\ t\ n \in s\ N$ **using** *assms(3)* t **unfolding** *subset-eq* t **by** *blast+*
hence $dist\ (t\ m)\ (t\ n) < e$ **using** N **by** *auto*
}
hence $\exists N.\ \forall m\ n.\ N \leq m \wedge N \leq n \longrightarrow dist\ (t\ m)\ (t\ n) < e$ **by** *auto*
}
hence *Cauchy* t **unfolding** *cauchy-def* **by** *auto*
then obtain l **where** l:$(t \dashrightarrow l)$ *sequentially* **using** *complete-univ* **unfolding** *complete-def* **by** *auto*
{ fix n::*nat*
{ fix e::*real* **assume** $e{>}0$
then obtain N::*nat* **where** N:$\forall n{\geq}N.\ dist\ (t\ n)\ l < e$ **using** l[*unfolded Lim-sequentially*] **by** *auto*
have $t\ (max\ n\ N) \in s\ n$ **using** *assms(3)* **unfolding** *subset-eq* **apply**(*erule-tac* $x{=}n$ **in** *allE*) **apply** (*erule-tac* $x{=}max\ n\ N$ **in** *allE*) **using** t **by** *auto*
hence $\exists y{\in}s\ n.\ dist\ y\ l < e$ **apply**(*rule-tac* $x{=}t\ (max\ n\ N)$ **in** *bexI*) **using** N **by** *auto*
}
hence $l \in s\ n$ **using** *closed-approachable*[*of* $s\ n\ l$] *assms(1)* **by** *auto*
}
then show *?thesis* **by** *auto*
qed

Strengthen it to the intersection actually being a singleton.

lemma *decreasing-closed-nest-sing*:
fixes s :: $nat \Rightarrow {'a}$::*heine-borel set*
assumes $\forall n.\ closed(s\ n)$
$\forall n.\ s\ n \neq \{\}$
$\forall m\ n.\ m \leq n \dashrightarrow s\ n \subseteq s\ m$
$\forall e{>}0.\ \exists n.\ \forall x \in (s\ n).\ \forall\ y{\in}(s\ n).\ dist\ x\ y < e$
shows $\exists a.\ \bigcap(range\ s) = \{a\}$
proof−
obtain a **where** a:$\forall n.\ a \in s\ n$ **using** *decreasing-closed-nest*[*of* s] **using** *assms* **by** *auto*
{ fix b **assume** b:$b \in \bigcap(range\ s)$
{ fix e::*real* **assume** $e{>}0$
hence $dist\ a\ b < e$ **using** *assms(4*)**using** b **using** a **by** *blast*
}
hence $dist\ a\ b = 0$ **by** (*metis dist-eq-0-iff dist-nz less-le*)
}
with a **have** $\bigcap(range\ s) = \{a\}$ **unfolding** *image-def* **by** *auto*
thus *?thesis* **..**

qed

Cauchy-type criteria for uniform convergence.

lemma *uniformly-convergent-eq-cauchy*: **fixes** *s*::*nat* ⇒ *'b* ⇒ *'a*::*heine-borel* **shows**
(∃ *l*. ∀ *e*>*0*. ∃ *N*. ∀ *n x*. *N* ≤ *n* ∧ *P x* −−> *dist*(*s n x*)(*l x*) < *e*) ⟷
(∀ *e*>*0*. ∃ *N*. ∀ *m n x*. *N* ≤ *m* ∧ *N* ≤ *n* ∧ *P x* −−> *dist* (*s m x*) (*s n x*) < *e*)
(**is** *?lhs* = *?rhs*)
proof(*rule*)
assume *?lhs*
then obtain *l* **where** *l*:∀ *e*>*0*. ∃ *N*. ∀ *n x*. *N* ≤ *n* ∧ *P x* ⟶ *dist* (*s n x*) (*l x*) < *e* **by** *auto*
{ fix *e*::*real* **assume** *e*>*0*
then obtain *N*::*nat* **where** *N*:∀ *n x*. *N* ≤ *n* ∧ *P x* ⟶ *dist* (*s n x*) (*l x*) < *e* / *2* **using** *l*[*THEN spec*[**where** *x*=*e*/*2*]] **by** *auto*
{ fix *n m*::*nat* **and** *x*::*'b* **assume** *N* ≤ *m* ∧ *N* ≤ *n* ∧ *P x*
hence *dist* (*s m x*) (*s n x*) < *e*
using *N*[*THEN spec*[**where** *x*=*m*], *THEN spec*[**where** *x*=*x*]]
using *N*[*THEN spec*[**where** *x*=*n*], *THEN spec*[**where** *x*=*x*]]
using *dist-triangle-half-l*[*of s m x l x e s n x*] **by** *auto* **}**
hence ∃ *N*. ∀ *m n x*. *N* ≤ *m* ∧ *N* ≤ *n* ∧ *P x* −−> *dist* (*s m x*) (*s n x*) < *e* **by** *auto* **}**
thus *?rhs* **by** *auto*
next
assume *?rhs*
hence ∀ *x*. *P x* ⟶ *Cauchy* (λ*n*. *s n x*) **unfolding** *cauchy-def* **apply** *auto* **by** (*erule-tac x*=*e* **in** *allE*)*auto*
then obtain *l* **where** *l*:∀ *x*. *P x* ⟶ ((λ*n*. *s n x*) −−−> *l x*) *sequentially* **unfolding** *convergent-eq-cauchy*[*THEN sym*]
using *choice*[*of* λ*x l*. *P x* ⟶ ((λ*n*. *s n x*) −−−> *l*) *sequentially*] **by** *auto*
{ fix *e*::*real* **assume** *e*>*0*
then obtain *N* **where** *N*:∀ *m n x*. *N* ≤ *m* ∧ *N* ≤ *n* ∧ *P x* ⟶ *dist* (*s m x*) (*s n x*) < *e*/*2*
using ‹*?rhs*›[*THEN spec*[**where** *x*=*e*/*2*]] **by** *auto*
{ fix *x* **assume** *P x*
then obtain *M* **where** *M*:∀ *n*≥*M*. *dist* (*s n x*) (*l x*) < *e*/*2*
using *l*[*THEN spec*[**where** *x*=*x*], *unfolded Lim-sequentially*] **using** ‹*e*>*0*› **by**(*auto elim*!: *allE*[**where** *x*=*e*/*2*])
fix *n*::*nat* **assume** *n*≥*N*
hence *dist*(*s n x*)(*l x*) < *e* **using** ‹*P x*›**and** *N*[*THEN spec*[**where** *x*=*n*], *THEN spec*[**where** *x*=*N*+*M*], *THEN spec*[**where** *x*=*x*]]
using *M*[*THEN spec*[**where** *x*=*N*+*M*]] **and** *dist-triangle-half-l*[*of s n x s* (*N*+*M*) *x e l x*] **by** (*auto simp add*: *dist-commute*) **}**
hence ∃ *N*. ∀ *n x*. *N* ≤ *n* ∧ *P x* ⟶ *dist*(*s n x*)(*l x*) < *e* **by** *auto* **}**
thus *?lhs* **by** *auto*
qed

lemma *uniformly-cauchy-imp-uniformly-convergent*:
fixes *s* :: *nat* ⇒ *'a* ⇒ *'b*::*heine-borel*
assumes ∀ *e*>*0*.∃ *N*. ∀ *m* (*n*::*nat*) *x*. *N* ≤ *m* ∧ *N* ≤ *n* ∧ *P x* −−> *dist*(*s m x*)(*s*

$n\ x) < e$
$\forall x.\ P\ x \longrightarrow (\forall e{>}0.\ \exists N.\ \forall n.\ N \leq n \longrightarrow dist(s\ n\ x)(l\ x) < e)$
shows $\forall e{>}0.\ \exists N.\ \forall n\ x.\ N \leq n \wedge P\ x \longrightarrow dist(s\ n\ x)(l\ x) < e$
proof−
obtain l' **where** $l{:}\forall e{>}0.\ \exists N.\ \forall n\ x.\ N \leq n \wedge P\ x \longrightarrow dist\ (s\ n\ x)\ (l'\ x) < e$
using *assms*(*1*) **unfolding** *uniformly-convergent-eq-cauchy*[*THEN sym*] **by** *auto*
moreover
{ fix x **assume** $P\ x$
hence $l\ x = l'\ x$ **using** *Lim-unique*[*OF trivial-limit-sequentially*, *of* $\lambda n.\ s\ n\ x$ $l\ x\ l'\ x$]
using l **and** *assms*(*2*) **unfolding** *Lim-sequentially* **by** *blast* **}**
ultimately show *?thesis* **by** *auto*
qed

16.20 Continuity

Define continuity over a net to take in restrictions of the set.

definition
continuous :: $'a{::}$*t2-space net* $\Rightarrow$ ($'a \Rightarrow 'b{::}$*topological-space*) $\Rightarrow$ *bool* **where**
continuous net f $\longleftrightarrow$ (f −−−> f(*netlimit net*)) *net*

lemma *continuous-trivial-limit*:
trivial-limit net ==> *continuous net f*
unfolding *continuous-def tendsto-def trivial-limit-eq* **by** *auto*

lemma *continuous-within*: *continuous* (*at x within s*) f $\longleftrightarrow$ (f −−−> $f(x)$) (*at x within s*)
unfolding *continuous-def*
unfolding *tendsto-def*
using *netlimit-within*[*of x s*]
by (*cases trivial-limit* (*at x within s*)) (*auto simp add*: *trivial-limit-eventually*)

lemma *continuous-at*: *continuous* (*at x*) f $\longleftrightarrow$ (f −−−> $f(x)$) (*at x*)
using *continuous-within* [*of x UNIV f*] **by** (*simp add*: *within-UNIV*)

lemma *continuous-at-within*:
assumes *continuous* (*at x*) f **shows** *continuous* (*at x within s*) f
using *assms* **unfolding** *continuous-at continuous-within*
by (*rule Lim-at-within*)

Derive the epsilon-delta forms, which we often use as "definitions"

lemma *continuous-within-eps-delta*:
continuous (*at x within s*) f $\longleftrightarrow$ ($\forall e{>}0.\ \exists d{>}0.\ \forall x'{\in}\ s.$ *dist* $x'\ x < d \longrightarrow$ *dist* ($f\ x'$) ($f\ x$) $< e$)
unfolding *continuous-within* **and** *Lim-within*
apply *auto* **unfolding** *dist-nz*[*THEN sym*] **apply**(*auto elim*!:*allE*) **apply**(*rule-tac x*=*d* **in** *exI*) **by** *auto*

lemma *continuous-at-eps-delta*: *continuous* (*at x*) *f* ⟷ (∀ *e*>*0*. ∃ *d*>*0*. ∀ *x'*. *dist x' x* < *d* −−> *dist*(*f x'*)(*f x*) < *e*)
using *continuous-within-eps-delta*[*of x UNIV f*]
unfolding *within-UNIV* **by** *blast*

Versions in terms of open balls.

lemma *continuous-within-ball*:
continuous (*at x within s*) *f* ⟷ (∀ *e*>*0*. ∃ *d*>*0*. *f* ‘ (*ball x d* ∩ *s*) ⊆ *ball* (*f x*) *e*) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
{ fix *e*::*real* **assume** *e*>*0*
then obtain *d* **where** *d*: *d*>*0* ∀ *xa*∈*s*. *0* < *dist xa x* ∧ *dist xa x* < *d* ⟶ *dist* (*f xa*) (*f x*) < *e*
using ‹*?lhs*›[*unfolded continuous-within Lim-within*] **by** *auto*
{ fix *y* **assume** *y*∈*f* ‘ (*ball x d* ∩ *s*)
hence *y* ∈ *ball* (*f x*) *e* **using** *d*(*2*) **unfolding** *dist-nz*[*THEN sym*]
apply (*auto simp add*: *dist-commute*) **apply**(*erule-tac x*=*xa* **in** *ballE*) **apply** *auto* **using** ‹*e*>*0*› **by** *auto*
}
hence ∃ *d*>*0*. *f* ‘ (*ball x d* ∩ *s*) ⊆ *ball* (*f x*) *e* **using** ‹*d*>*0*› **unfolding** *subset-eq ball-def* **by** (*auto simp add*: *dist-commute*) **}**
thus *?rhs* **by** *auto*
next
assume *?rhs* **thus** *?lhs* **unfolding** *continuous-within Lim-within ball-def subset-eq*
apply (*auto simp add*: *dist-commute*) **apply**(*erule-tac x*=*e* **in** *allE*) **by** *auto*
qed

lemma *continuous-at-ball*:
continuous (*at x*) *f* ⟷ (∀ *e*>*0*. ∃ *d*>*0*. *f* ‘ (*ball x d*) ⊆ *ball* (*f x*) *e*) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **thus** *?rhs* **unfolding** *continuous-at Lim-at subset-eq Ball-def Bex-def image-iff mem-ball*
apply *auto* **apply**(*erule-tac x*=*e* **in** *allE*) **apply** *auto* **apply**(*rule-tac x*=*d* **in** *exI*) **apply** *auto* **apply**(*erule-tac x*=*xa* **in** *allE*) **apply** (*auto simp add*: *dist-commute dist-nz*)
unfolding *dist-nz*[*THEN sym*] **by** *auto*
next
assume *?rhs* **thus** *?lhs* **unfolding** *continuous-at Lim-at subset-eq Ball-def Bex-def image-iff mem-ball*
apply *auto* **apply**(*erule-tac x*=*e* **in** *allE*) **apply** *auto* **apply**(*rule-tac x*=*d* **in** *exI*) **apply** *auto* **apply**(*erule-tac x*=*f xa* **in** *allE*) **by** (*auto simp add*: *dist-commute dist-nz*)
qed

Define setwise continuity in terms of limits within the set.

definition
continuous-on ::

'a set ⇒ (*'a*::*topological-space* ⇒ *'b*::*topological-space*) ⇒ *bool*
where
continuous-on s f ⟷ (∀ *x*∈*s*. (*f* −−−> *f x*) (*at x within s*))

lemma *continuous-on-topological*:
continuous-on s f ⟷
(∀ *x*∈*s*. ∀ *B*. *open B* ⟶ *f x* ∈ *B* ⟶
(∃ *A*. *open A* ∧ *x* ∈ *A* ∧ (∀ *y*∈*s*. *y* ∈ *A* ⟶ *f y* ∈ *B*)))
unfolding *continuous-on-def tendsto-def*
unfolding *Limits.eventually-within eventually-at-topological*
by (*intro ball-cong* [*OF refl*] *all-cong imp-cong ex-cong conj-cong refl*) *auto*

lemma *continuous-on-iff*:
continuous-on s f ⟷
(∀ *x*∈*s*. ∀ *e*>*0*. ∃ *d*>*0*. ∀ *x'*∈*s*. *dist x' x* < *d* ⟶ *dist* (*f x'*) (*f x*) < *e*)
unfolding *continuous-on-def Lim-within*
apply (*intro ball-cong* [*OF refl*] *all-cong ex-cong*)
apply (*rename-tac y*, *case-tac y* = *x*, *simp*)
apply (*simp add*: *dist-nz*)
done

definition
uniformly-continuous-on ::
'a set ⇒ (*'a*::*metric-space* ⇒ *'b*::*metric-space*) ⇒ *bool*
where
uniformly-continuous-on s f ⟷
(∀ *e*>*0*. ∃ *d*>*0*. ∀ *x*∈*s*. ∀ *x'*∈*s*. *dist x' x* < *d* ⟶ *dist* (*f x'*) (*f x*) < *e*)

Some simple consequential lemmas.

lemma *uniformly-continuous-imp-continuous*:
uniformly-continuous-on s f ==> *continuous-on s f*
unfolding *uniformly-continuous-on-def continuous-on-iff* **by** *blast*

lemma *continuous-at-imp-continuous-within*:
continuous (*at x*) *f* ==> *continuous* (*at x within s*) *f*
unfolding *continuous-within continuous-at* **using** *Lim-at-within* **by** *auto*

lemma *Lim-trivial-limit*: *trivial-limit net* ⟹ (*f* −−−> *l*) *net*
unfolding *tendsto-def* **by** (*simp add*: *trivial-limit-eq*)

lemma *continuous-at-imp-continuous-on*:
assumes ∀ *x*∈*s*. *continuous* (*at x*) *f*
shows *continuous-on s f*
unfolding *continuous-on-def*
proof
fix *x* **assume** *x* ∈ *s*
with *assms* **have** ∗: (*f* −−−> *f* (*netlimit* (*at x*))) (*at x*)
unfolding *continuous-def* **by** *simp*
have (*f* −−−> *f x*) (*at x*)

proof (*cases trivial-limit* (*at x*))
case *True* **thus** *?thesis*
by (*rule Lim-trivial-limit*)
next
case *False*
hence *1*: *netlimit* (*at x*) = *x*
using *netlimit-within* [*of x UNIV*]
by (*simp add*: *within-UNIV*)
with ∗ **show** *?thesis* **by** *simp*
qed
thus (*f* −−−> *f x*) (*at x within s*)
by (*rule Lim-at-within*)
qed

lemma *continuous-on-eq-continuous-within*:
continuous-on s f ⟷ (∀ *x* ∈ *s*. *continuous* (*at x within s*) *f*)
unfolding *continuous-on-def continuous-def*
apply (*rule ball-cong* [*OF refl*])
apply (*case-tac trivial-limit* (*at x within s*))
apply (*simp add*: *Lim-trivial-limit*)
apply (*simp add*: *netlimit-within*)
done

lemmas *continuous-on* = *continuous-on-def* — legacy theorem name

lemma *continuous-on-eq-continuous-at*:
shows *open s* ==> (*continuous-on s f* ⟷ (∀ *x* ∈ *s*. *continuous* (*at x*) *f*))
by (*auto simp add*: *continuous-on continuous-at Lim-within-open*)

lemma *continuous-within-subset*:
continuous (*at x within s*) *f* ⟹ *t* ⊆ *s*
==> *continuous* (*at x within t*) *f*
unfolding *continuous-within* **by**(*metis Lim-within-subset*)

lemma *continuous-on-subset*:
shows *continuous-on s f* ⟹ *t* ⊆ *s* ==> *continuous-on t f*
unfolding *continuous-on* **by** (*metis subset-eq Lim-within-subset*)

lemma *continuous-on-interior*:
shows *continuous-on s f* ⟹ *x* ∈ *interior s* ==> *continuous* (*at x*) *f*
unfolding *interior-def*
apply *simp*
by (*meson continuous-on-eq-continuous-at continuous-on-subset*)

lemma *continuous-on-eq*:
(∀ *x* ∈ *s*. *f x* = *g x*) ⟹ *continuous-on s f* ⟹ *continuous-on s g*
unfolding *continuous-on-def tendsto-def Limits.eventually-within*
by *simp*

Characterization of various kinds of continuity in terms of sequences.

lemma *continuous-within-sequentially*:
fixes f :: *'a::metric-space* ⇒ *'b::metric-space*
shows *continuous* (*at a within s*) f ⟷
(∀ *x*. (∀ *n::nat*. *x n* ∈ *s*) ∧ (*x* −−−> *a*) *sequentially*
−−> ((*f o x*) −−−> *f a*) *sequentially*) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
{ **fix** *x::nat* ⇒ *'a* **assume** *x*:∀ *n*. *x n* ∈ *s* ∀ *e*>0. ∃ *N*. ∀ *n*≥*N*. *dist* (*x n*) *a* < *e*
fix *e::real* **assume** *e*>0
from ‹*?lhs*› **obtain** *d* **where** *d*>0 **and** *d*:∀ *x*∈*s*. 0 < *dist x a* ∧ *dist x a* < *d*
⟶ *dist* (*f x*) (*f a*) < *e* **unfolding** *continuous-within Lim-within* **using** ‹*e*>0› **by**
auto
from *x*(*2*) ‹*d*>0› **obtain** *N* **where** *N*:∀ *n*≥*N*. *dist* (*x n*) *a* < *d* **by** *auto*
hence ∃ *N*. ∀ *n*≥*N*. *dist* ((*f* ∘ *x*) *n*) (*f a*) < *e*
apply(*rule-tac* *x*=*N* **in** *exI*) **using** *N d* **apply** *auto* **using** *x*(*1*)
apply(*erule-tac x*=*n* **in** *allE*) **apply**(*erule-tac x*=*n* **in** *allE*)
apply(*erule-tac x*=*x n* **in** *ballE*) **apply** *auto* **unfolding** *dist-nz*[*THEN sym*]
apply *auto* **using** ‹*e*>0› **by** *auto*
}
thus *?rhs* **unfolding** *continuous-within* **unfolding** *Lim-sequentially* **by** *simp*
next
assume *?rhs*
{ **fix** *e::real* **assume** *e*>0
assume ¬ (∃ *d*>0. ∀ *x*∈*s*. 0 < *dist x a* ∧ *dist x a* < *d* ⟶ *dist* (*f x*) (*f a*) < *e*)
hence ∀ *d*. ∃ *x*. *d*>0 ⟶ *x*∈*s* ∧ (0 < *dist x a* ∧ *dist x a* < *d* ∧ ¬ *dist* (*f x*) (*f*
a) < *e*) **by** *blast*
then obtain *x* **where** *x*:∀ *d*>0. *x d* ∈ *s* ∧ (0 < *dist* (*x d*) *a* ∧ *dist* (*x d*) *a* <
d ∧ ¬ *dist* (*f* (*x d*)) (*f a*) < *e*)
using *choice*[*of* λ*d x*.0<*d* ⟶ *x*∈*s* ∧ (0 < *dist x a* ∧ *dist x a* < *d* ∧ ¬ *dist*
(*f x*) (*f a*) < *e*)] **by** *auto*
{ **fix** *d::real* **assume** *d*>0
hence ∃ *N::nat*. *inverse* (*real* (*N* + *1*)) < *d* **using** *real-arch-inv*[*of d*] **by**
(*auto*, *rule-tac x*=*n* − *1* **in** *exI*)*auto*
then obtain *N::nat* **where** *N*:*inverse* (*real* (*N* + *1*)) < *d* **by** *auto*
{ **fix** *n::nat* **assume** *n*:*n*≥*N*
hence *dist* (*x* (*inverse* (*real* (*n* + *1*)))) *a* < *inverse* (*real* (*n* + *1*)) **using**
x[*THEN spec*[**where** *x*=*inverse* (*real* (*n* + *1*))]] **by** *auto*
moreover have *inverse* (*real* (*n* + *1*)) < *d* **using** *N n* **by** (*auto*, *metis*
Suc-le-mono le-SucE less-imp-inverse-less nat-le-real-less order-less-trans real-of-nat-Suc
real-of-nat-Suc-gt-zero)
ultimately have *dist* (*x* (*inverse* (*real* (*n* + *1*)))) *a* < *d* **by** *auto*
}
hence ∃ *N::nat*. ∀ *n*≥*N*. *dist* (*x* (*inverse* (*real* (*n* + *1*)))) *a* < *d* **by** *auto*
}
hence (∀ *n::nat*. *x* (*inverse* (*real* (*n* + *1*))) ∈ *s*) ∧ (∀ *e*>0. ∃ *N::nat*. ∀ *n*≥*N*.
dist (*x* (*inverse* (*real* (*n* + *1*)))) *a* < *e*) **using** *x* **by** *auto*
hence ∀ *e*>0. ∃ *N::nat*. ∀ *n*≥*N*. *dist* (*f* (*x* (*inverse* (*real* (*n* + *1*))))) (*f a*) <
e **using** ‹*?rhs*›[*THEN spec*[**where** *x*=λ*n::nat*. *x* (*inverse* (*real* (*n*+*1*)))], *unfolded*
Lim-sequentially] **by** *auto*

hence *False* **apply**(*erule-tac x=e* **in** *allE*) **using** ‹*e>0*› **using** *x* **by** *auto*
}
thus *?lhs* **unfolding** *continuous-within* **unfolding** *Lim-within* **unfolding** *Lim-sequentially* **by** *blast*
qed

lemma *continuous-at-sequentially*:
fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*metric-space*
shows *continuous* (*at a*) *f* ⟷ (∀ *x*. (*x* −−−> *a*) *sequentially*
−−> ((*f o x*) −−−> *f a*) *sequentially*)
using *continuous-within-sequentially*[*of a UNIV f*] **unfolding** *within-UNIV* **by** *auto*

lemma *continuous-on-sequentially*:
fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*metric-space*
shows *continuous-on s f* ⟷
(∀ *x*. ∀ *a* ∈ *s*. (∀ *n*. *x*(*n*) ∈ *s*) ∧ (*x* −−−> *a*) *sequentially*
−−> ((*f o x*) −−−> *f*(*a*)) *sequentially*) (**is** *?lhs* = *?rhs*)
proof
assume *?rhs* **thus** *?lhs* **using** *continuous-within-sequentially*[*of - s f*] **unfolding** *continuous-on-eq-continuous-within* **by** *auto*
next
assume *?lhs* **thus** *?rhs* **unfolding** *continuous-on-eq-continuous-within* **using** *continuous-within-sequentially*[*of - s f*] **by** *auto*
qed

lemma *uniformly-continuous-on-sequentially'*:
uniformly-continuous-on s f ⟷ (∀ *x y*. (∀ *n*. *x n* ∈ *s*) ∧ (∀ *n*. *y n* ∈ *s*) ∧
((λ*n*. *dist* (*x n*) (*y n*)) −−−> *0*) *sequentially*
⟶ ((λ*n*. *dist* (*f*(*x n*)) (*f*(*y n*))) −−−> *0*) *sequentially*) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
{ fix *x y* **assume** *x*:∀ *n*. *x n* ∈ *s* **and** *y*:∀ *n*. *y n* ∈ *s* **and** *xy*:((λ*n*. *dist* (*x n*) (*y n*)) −−−> *0*) *sequentially*
{ fix *e*::*real* **assume** *e>0*
then obtain *d* **where** *d>0* **and** *d*:∀ *x*∈*s*. ∀ *x'*∈*s*. *dist x' x* < *d* ⟶ *dist* (*f x'*) (*f x*) < *e*
using ‹*?lhs*›[*unfolded uniformly-continuous-on-def*, *THEN spec*[**where** *x=e*]] **by** *auto*
obtain *N* **where** *N*:∀ *n*≥*N*. *dist* (*x n*) (*y n*) < *d* **using** *xy*[*unfolded Lim-sequentially dist-norm*] **and** ‹*d>0*› **by** *auto*
{ fix *n* **assume** *n*≥*N*
hence *dist* (*f* (*x n*)) (*f* (*y n*)) < *e*
using *N*[*THEN spec*[**where** *x=n*]] **using** *d*[*THEN bspec*[**where** *x=x n*], *THEN bspec*[**where** *x=y n*]] **using** *x* **and** *y*
unfolding *dist-commute* **by** *simp* **}**
hence ∃ *N*. ∀ *n*≥*N*. *dist* (*f* (*x n*)) (*f* (*y n*)) < *e* **by** *auto* **}**
hence ((λ*n*. *dist* (*f*(*x n*)) (*f*(*y n*))) −−−> *0*) *sequentially* **unfolding** *Lim-sequentially*

and *dist-real-def* **by** *auto* **}**
thus *?rhs* **by** *auto*
next
assume *?rhs*
{ assume ¬ *?lhs*
then obtain *e* **where** *e>0* ∀ *d>0*. ∃ *x∈s*. ∃ *x′∈s*. *dist x′ x* < *d* ∧ ¬ *dist* (*f x′*) (*f x*) < *e* **unfolding** *uniformly-continuous-on-def* **by** *auto*
then obtain *fa* **where** *fa*:∀ *x*. *0* < *x* ⟶ *fst* (*fa x*) ∈ *s* ∧ *snd* (*fa x*) ∈ *s* ∧ *dist* (*fst* (*fa x*)) (*snd* (*fa x*)) < *x* ∧ ¬ *dist* (*f* (*fst* (*fa x*))) (*f* (*snd* (*fa x*))) < *e*
using *choice*[*of* λ*d x*. *d>0* ⟶ *fst x* ∈ *s* ∧ *snd x* ∈ *s* ∧ *dist* (*snd x*) (*fst x*) < *d* ∧ ¬ *dist* (*f* (*snd x*)) (*f* (*fst x*)) < *e*] **unfolding** *Bex-def*
by (*auto simp add*: *dist-commute*)
def *x* ≡ λ*n*::*nat*. *fst* (*fa* (*inverse* (*real n* + *1*)))
def *y* ≡ λ*n*::*nat*. *snd* (*fa* (*inverse* (*real n* + *1*)))
have *xyn*:∀ *n*. *x n* ∈ *s* ∧ *y n* ∈ *s* **and** *xy0*:∀ *n*. *dist* (*x n*) (*y n*) < *inverse* (*real n* + *1*) **and** *fxy*:∀ *n*. ¬ *dist* (*f* (*x n*)) (*f* (*y n*)) < *e*
unfolding *x-def* **and** *y-def* **using** *fa* **by** *auto*
{ fix *e*::*real* **assume** *e>0*
then obtain *N*::*nat* **where** *N* ≠ *0* **and** *N*:*0* < *inverse* (*real N*) ∧ *inverse* (*real N*) < *e* **unfolding** *real-arch-inv*[*of e*] **by** *auto*
{ fix *n*::*nat* **assume** *n*≥*N*
hence *inverse* (*real n* + *1*) < *inverse* (*real N*) **using** *real-of-nat-ge-zero* **and** ‹*N*≠*0*› **by** *auto*
also have ... < *e* **using** *N* **by** *auto*
finally have *inverse* (*real n* + *1*) < *e* **by** *auto*
hence *dist* (*x n*) (*y n*) < *e* **using** *xy0*[*THEN spec*[**where** *x*=*n*]] **by** *auto* **}**
hence ∃ *N*. ∀ *n*≥*N*. *dist* (*x n*) (*y n*) < *e* **by** *auto* **}**
hence ∀ *e>0*. ∃ *N*. ∀ *n*≥*N*. *dist* (*f* (*x n*)) (*f* (*y n*)) < *e* **using** ‹*?rhs*›[*THEN spec*[**where** *x*=*x*], *THEN spec*[**where** *x*=*y*]] **and** *xyn* **unfolding** *Lim-sequentially dist-real-def* **by** *auto*
hence *False* **using** *fxy* **and** ‹*e>0*› **by** *auto* **}**
thus *?lhs* **unfolding** *uniformly-continuous-on-def* **by** *blast*
qed

lemma *uniformly-continuous-on-sequentially*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector*
shows *uniformly-continuous-on s f* ⟷ (∀ *x y*. (∀ *n*. *x n* ∈ *s*) ∧ (∀ *n*. *y n* ∈ *s*) ∧ ((λ*n*. *x n* − *y n*) −−−> *0*) *sequentially* ⟶ ((λ*n*. *f*(*x n*) − *f*(*y n*)) −−−> *0*) *sequentially*) (**is** *?lhs* = *?rhs*)

unfolding *uniformly-continuous-on-sequentially′*
unfolding *dist-norm Lim-null-norm* [*symmetric*] **..**

The usual transformation theorems.

lemma *continuous-transform-within*:
fixes *f g* :: *'a*::*metric-space* ⇒ *'b*::*topological-space*
assumes *0* < *d x* ∈ *s* ∀ *x′* ∈ *s*. *dist x′ x* < *d* −−> *f x′* = *g x′*
continuous (*at x within s*) *f*

shows *continuous* (*at x within s*) *g*
unfolding *continuous-within*
proof (*rule Lim-transform-within*)
show $0 < d$ **by** *fact*
show $\forall x' \in s.\ 0 < dist\ x'\ x \wedge dist\ x'\ x < d \longrightarrow f\ x' = g\ x'$
using *assms*(*3*) **by** *auto*
have $f\ x = g\ x$
using *assms*(*1*,*2*,*3*) **by** *auto*
thus (*f* ---> *g x*) (*at x within s*)
using *assms*(*4*) **unfolding** *continuous-within* **by** *simp*
qed

lemma *continuous-transform-at*:
fixes *f g* :: *'a*::*metric-space* ⇒ *'b*::*topological-space*
assumes $0 < d$ $\forall x'.$ *dist x' x* < *d* --> *f x'* = *g x'*
continuous (*at x*) *f*
shows *continuous* (*at x*) *g*
using *continuous-transform-within* [*of d x UNIV f g*] *assms*
by (*simp add*: *within-UNIV*)

Combination results for pointwise continuity.

lemma *continuous-const*: *continuous net* (λ*x*. *c*)
by (*auto simp add*: *continuous-def Lim-const*)

lemma *continuous-cmul*:
fixes *f* :: *'a*::*t2-space* ⇒ *'b*::*real-normed-vector*
shows *continuous net f* ==> *continuous net* (λ*x*. *c* *$_R$ *f x*)
by (*auto simp add*: *continuous-def Lim-cmul*)

lemma *continuous-neg*:
fixes *f* :: *'a*::*t2-space* ⇒ *'b*::*real-normed-vector*
shows *continuous net f* ==> *continuous net* (λ*x*. −(*f x*))
by (*auto simp add*: *continuous-def Lim-neg*)

lemma *continuous-add*:
fixes *f g* :: *'a*::*t2-space* ⇒ *'b*::*real-normed-vector*
shows *continuous net f* ⟹ *continuous net g* ⟹ *continuous net* (λ*x*. *f x* + *g x*)
by (*auto simp add*: *continuous-def Lim-add*)

lemma *continuous-sub*:
fixes *f g* :: *'a*::*t2-space* ⇒ *'b*::*real-normed-vector*
shows *continuous net f* ⟹ *continuous net g* ⟹ *continuous net* (λ*x*. *f x* − *g x*)
by (*auto simp add*: *continuous-def Lim-sub*)

Same thing for setwise continuity.

lemma *continuous-on-const*:
continuous-on s (λ*x*. *c*)

unfolding *continuous-on-def* **by** *auto*

lemma *continuous-on-cmul*:
fixes f :: *'a*::*topological-space* ⇒ *'b*::*real-normed-vector*
shows *continuous-on s f* ⟹ *continuous-on s* (λ*x*. *c* ∗$_R$ (*f x*))
unfolding *continuous-on-def* **by** (*auto intro*: *tendsto-intros*)

lemma *continuous-on-neg*:
fixes f :: *'a*::*topological-space* ⇒ *'b*::*real-normed-vector*
shows *continuous-on s f* ⟹ *continuous-on s* (λ*x*. − *f x*)
unfolding *continuous-on-def* **by** (*auto intro*: *tendsto-intros*)

lemma *continuous-on-add*:
fixes *f g* :: *'a*::*topological-space* ⇒ *'b*::*real-normed-vector*
shows *continuous-on s f* ⟹ *continuous-on s g*
⟹ *continuous-on s* (λ*x*. *f x* + *g x*)
unfolding *continuous-on-def* **by** (*auto intro*: *tendsto-intros*)

lemma *continuous-on-sub*:
fixes *f g* :: *'a*::*topological-space* ⇒ *'b*::*real-normed-vector*
shows *continuous-on s f* ⟹ *continuous-on s g*
⟹ *continuous-on s* (λ*x*. *f x* − *g x*)
unfolding *continuous-on-def* **by** (*auto intro*: *tendsto-intros*)

Same thing for uniform continuity, using sequential formulations.

lemma *uniformly-continuous-on-const*:
uniformly-continuous-on s (λ*x*. *c*)
unfolding *uniformly-continuous-on-def* **by** *simp*

lemma *uniformly-continuous-on-cmul*:
fixes f :: *'a*::*metric-space* ⇒ *'b*::*real-normed-vector*
assumes *uniformly-continuous-on s f*
shows *uniformly-continuous-on s* (λ*x*. *c* ∗$_R$ *f*(*x*))
proof−
{ fix *x y* **assume** ((λ*n*. *f* (*x n*) − *f* (*y n*)) −−−> *0*) *sequentially*
hence ((λ*n*. *c* ∗$_R$ *f* (*x n*) − *c* ∗$_R$ *f* (*y n*)) −−−> *0*) *sequentially*
using *Lim-cmul*[*of* (λ*n*. *f* (*x n*) − *f* (*y n*)) *0 sequentially c*]
unfolding *scaleR-zero-right scaleR-right-diff-distrib* **by** *auto*
}
thus *?thesis* **using** *assms* **unfolding** *uniformly-continuous-on-sequentially'*
unfolding *dist-norm Lim-null-norm* [*symmetric*] **by** *auto*
qed

lemma *dist-minus*:
fixes *x y* :: *'a*::*real-normed-vector*
shows *dist* (− *x*) (− *y*) = *dist x y*
unfolding *dist-norm minus-diff-minus norm-minus-cancel* **..**

lemma *uniformly-continuous-on-neg*:

fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*real-normed-vector*
shows *uniformly-continuous-on s f*
==> *uniformly-continuous-on s* (λ*x*. −(*f x*))
unfolding *uniformly-continuous-on-def dist-minus* **.**

lemma *uniformly-continuous-on-add*:
fixes *f g* :: *'a*::*metric-space* ⇒ *'b*::*real-normed-vector*
assumes *uniformly-continuous-on s f uniformly-continuous-on s g*
shows *uniformly-continuous-on s* (λ*x*. *f x* + *g x*)
proof−
{ fix *x y* **assume** ((λ*n*. *f* (*x n*) − *f* (*y n*)) −−−> *0*) *sequentially*
((λ*n*. *g* (*x n*) − *g* (*y n*)) −−−> *0*) *sequentially*
hence ((λ*xa*. *f* (*x xa*) − *f* (*y xa*) + (*g* (*x xa*) − *g* (*y xa*))) −−−> *0* + *0*) *sequentially*
using *Lim-add*[*of* λ *n*. *f* (*x n*) − *f* (*y n*) *0 sequentially* λ *n*. *g* (*x n*) − *g* (*y n*) *0*] **by** *auto*
hence ((λ*n*. *f* (*x n*) + *g* (*x n*) − (*f* (*y n*) + *g* (*y n*))) −−−> *0*) *sequentially*
unfolding *Lim-sequentially* **and** *add-diff-add* [*symmetric*] **by** *auto* **}**
thus *?thesis* **using** *assms* **unfolding** *uniformly-continuous-on-sequentially'*
unfolding *dist-norm Lim-null-norm* [*symmetric*] **by** *auto*
qed

lemma *uniformly-continuous-on-sub*:
fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*real-normed-vector*
shows *uniformly-continuous-on s f* ⟹ *uniformly-continuous-on s g*
==> *uniformly-continuous-on s* (λ*x*. *f x* − *g x*)
unfolding *ab-diff-minus*
using *uniformly-continuous-on-add*[*of s f* λ*x*. − *g x*]
using *uniformly-continuous-on-neg*[*of s g*] **by** *auto*

Identity function is continuous in every sense.

lemma *continuous-within-id*:
continuous (*at a within s*) (λ*x*. *x*)
unfolding *continuous-within* **by** (*rule Lim-at-within* [*OF Lim-ident-at*])

lemma *continuous-at-id*:
continuous (*at a*) (λ*x*. *x*)
unfolding *continuous-at* **by** (*rule Lim-ident-at*)

lemma *continuous-on-id*:
continuous-on s (λ*x*. *x*)
unfolding *continuous-on-def* **by** (*auto intro*: *tendsto-ident-at-within*)

lemma *uniformly-continuous-on-id*:
uniformly-continuous-on s (λ*x*. *x*)
unfolding *uniformly-continuous-on-def* **by** *auto*

Continuity of all kinds is preserved under composition.

lemma *continuous-within-topological*:

continuous (*at x within s*) *f* ⟷
(∀ *B*. *open B* ⟶ *f x* ∈ *B* ⟶
(∃ *A*. *open A* ∧ *x* ∈ *A* ∧ (∀ *y*∈*s*. *y* ∈ *A* ⟶ *f y* ∈ *B*)))
unfolding *continuous-within*
unfolding *tendsto-def Limits.eventually-within eventually-at-topological*
by (*intro ball-cong* [*OF refl*] *all-cong imp-cong ex-cong conj-cong refl*) *auto*

lemma *continuous-within-compose*:
assumes *continuous* (*at x within s*) *f*
assumes *continuous* (*at* (*f x*) *within f* ‘ *s*) *g*
shows *continuous* (*at x within s*) (*g o f*)
using *assms* **unfolding** *continuous-within-topological* **by** *simp metis*

lemma *continuous-at-compose*:
assumes *continuous* (*at x*) *f continuous* (*at* (*f x*)) *g*
shows *continuous* (*at x*) (*g o f*)
proof−
have *continuous* (*at* (*f x*) *within range f*) *g* **using** *assms*(*2*) **using** *continuous-within-subset*[*of f x UNIV g range f*, *unfolded within-UNIV*] **by** *auto*
thus *?thesis* **using** *assms*(*1*) **using** *continuous-within-compose*[*of x UNIV f g*, *unfolded within-UNIV*] **by** *auto*
qed

lemma *continuous-on-compose*:
continuous-on s f ⟹ *continuous-on* (*f* ‘ *s*) *g* ⟹ *continuous-on s* (*g o f*)
unfolding *continuous-on-topological* **by** *simp metis*

lemma *uniformly-continuous-on-compose*:
assumes *uniformly-continuous-on s f uniformly-continuous-on* (*f* ‘ *s*) *g*
shows *uniformly-continuous-on s* (*g o f*)
proof−
{ fix *e*::*real* **assume** *e*>*0*
then obtain *d* **where** *d*>*0* **and** *d*:∀ *x*∈*f* ‘ *s*. ∀ *x*′∈*f* ‘ *s*. *dist x*′ *x* < *d* ⟶ *dist* (*g x*′) (*g x*) < *e* **using** *assms*(*2*) **unfolding** *uniformly-continuous-on-def* **by** *auto*
obtain *d*′ **where** *d*′>*0* ∀ *x*∈*s*. ∀ *x*′∈*s*. *dist x*′ *x* < *d*′ ⟶ *dist* (*f x*′) (*f x*) < *d* **using** ‹*d*>*0*› **using** *assms*(*1*) **unfolding** *uniformly-continuous-on-def* **by** *auto*
hence ∃ *d*>*0*. ∀ *x*∈*s*. ∀ *x*′∈*s*. *dist x*′ *x* < *d* ⟶ *dist* ((*g* ∘ *f*) *x*′) ((*g* ∘ *f*) *x*) < *e* **using** ‹*d*>*0*› **using** *d* **by** *auto* **}**
thus *?thesis* **using** *assms* **unfolding** *uniformly-continuous-on-def* **by** *auto*
qed

Continuity in terms of open preimages.

lemma *continuous-at-open*:
shows *continuous* (*at x*) *f* ⟷ (∀ *t*. *open t* ∧ *f x* ∈ *t* −−> (∃ *s*. *open s* ∧ *x* ∈ *s* ∧ (∀ *x*′ ∈ *s*. (*f x*′) ∈ *t*)))
unfolding *continuous-within-topological* [*of x UNIV f*, *unfolded within-UNIV*]
unfolding *imp-conjL* **by** (*intro all-cong imp-cong ex-cong conj-cong refl*) *auto*

lemma *continuous-on-open*:

```
  shows continuous-on s f ⟷
      (∀ t. openin (subtopology euclidean (f ‘ s)) t
        −−> openin (subtopology euclidean s) {x ∈ s. f x ∈ t}) (is ?lhs = ?rhs)
proof (safe)
  fix t :: 'b set
  assume 1: continuous-on s f
  assume 2: openin (subtopology euclidean (f ‘ s)) t
  from 2 obtain B where B: open B and t: t = f ‘ s ∩ B
    unfolding openin-open by auto
  def U == ⋃{A. open A ∧ (∀ x∈s. x ∈ A ⟶ f x ∈ B)}
  have open U unfolding U-def by (simp add: open-Union)
  moreover have ∀ x∈s. x ∈ U ⟷ f x ∈ t
  proof (intro ballI iffI)
    fix x assume x ∈ s and x ∈ U thus f x ∈ t
      unfolding U-def t by auto
  next
    fix x assume x ∈ s and f x ∈ t
    hence x ∈ s and f x ∈ B
      unfolding t by auto
    with 1 B obtain A where open A x ∈ A ∀ y∈s. y ∈ A ⟶ f y ∈ B
      unfolding t continuous-on-topological by metis
    then show x ∈ U
      unfolding U-def by auto
  qed
  ultimately have open U ∧ {x ∈ s. f x ∈ t} = s ∩ U by auto
  then show openin (subtopology euclidean s) {x ∈ s. f x ∈ t}
    unfolding openin-open by fast
next
  assume ?rhs show continuous-on s f
  unfolding continuous-on-topological
  proof (clarify)
    fix x and B assume x ∈ s and open B and f x ∈ B
    have openin (subtopology euclidean (f ‘ s)) (f ‘ s ∩ B)
      unfolding openin-open using ‹open B› by auto
    then have openin (subtopology euclidean s) {x ∈ s. f x ∈ f ‘ s ∩ B}
      using ‹?rhs› by fast
    then show ∃ A. open A ∧ x ∈ A ∧ (∀ y∈s. y ∈ A ⟶ f y ∈ B)
      unfolding openin-open using ‹x ∈ s› and ‹f x ∈ B› by auto
  qed
qed
```

Similarly in terms of closed sets.

```
lemma continuous-on-closed:
  shows continuous-on s f ⟷  (∀ t. closedin (subtopology euclidean (f ‘ s)) t
−−> closedin (subtopology euclidean s) {x ∈ s. f x ∈ t}) (is ?lhs = ?rhs)
proof
  assume ?lhs
  { fix t
    have ∗:s − {x ∈ s. f x ∈ f ‘ s − t} = {x ∈ s. f x ∈ t} by auto
```

have $**$:f ‘ $s - (f$ ‘ $s - (f$ ‘ $s - t)) = f$ ‘ $s - t$ **by** *auto*
assume *as*:*closedin* (*subtopology euclidean* (f ‘ s)) t
hence *closedin* (*subtopology euclidean* (f ‘ s)) (f ‘ $s - (f$ ‘ $s - t)$) **unfolding** *closedin-def topspace-euclidean-subtopology* **unfolding** $**$ **by** *auto*
hence *closedin* (*subtopology euclidean s*) $\{x \in s.\ f\,x \in t\}$ **using** ‹*?lhs*›[*unfolded continuous-on-open*, *THEN spec*[**where** x=(f ‘ s) $- t$]]
unfolding *openin-closedin-eq topspace-euclidean-subtopology* **unfolding** $*$ **by** *auto* **}**
thus *?rhs* **by** *auto*
next
assume *?rhs*
{ fix t
have $*$:$s - \{x \in s.\ f\,x \in f$ ‘ $s - t\} = \{x \in s.\ f\,x \in t\}$ **by** *auto*
assume *as*:*openin* (*subtopology euclidean* (f ‘ s)) t
hence *openin* (*subtopology euclidean s*) $\{x \in s.\ f\,x \in t\}$ **using** ‹*?rhs*›[*THEN spec*[**where** x=(f ‘ s) $- t$]]
unfolding *openin-closedin-eq topspace-euclidean-subtopology* $*$[*THEN sym*] *closedin-subtopology* **by** *auto* **}**
thus *?lhs* **unfolding** *continuous-on-open* **by** *auto*
qed

Half-global and completely global cases.

lemma *continuous-open-in-preimage*:
assumes *continuous-on s f open t*
shows *openin* (*subtopology euclidean s*) $\{x \in s.\ f\,x \in t\}$
proof−
have $*$:$\forall x.\ x \in s \wedge f\,x \in t \longleftrightarrow x \in s \wedge f\,x \in (t \cap f$ ‘ $s)$ **by** *auto*
have *openin* (*subtopology euclidean* (f ‘ s)) ($t \cap f$ ‘ s)
using *openin-open-Int*[*of t f* ‘ *s*, *OF assms*(*2*)] **unfolding** *openin-open* **by** *auto*
thus *?thesis* **using** *assms*(*1*)[*unfolded continuous-on-open*, *THEN spec*[**where** x=$t \cap f$ ‘ s]] **using** $*$ **by** *auto*
qed

lemma *continuous-closed-in-preimage*:
assumes *continuous-on s f closed t*
shows *closedin* (*subtopology euclidean s*) $\{x \in s.\ f\,x \in t\}$
proof−
have $*$:$\forall x.\ x \in s \wedge f\,x \in t \longleftrightarrow x \in s \wedge f\,x \in (t \cap f$ ‘ $s)$ **by** *auto*
have *closedin* (*subtopology euclidean* (f ‘ s)) ($t \cap f$ ‘ s)
using *closedin-closed-Int*[*of t f* ‘ *s*, *OF assms*(*2*)] **unfolding** *Int-commute* **by** *auto*
thus *?thesis*
using *assms*(*1*)[*unfolded continuous-on-closed*, *THEN spec*[**where** x=$t \cap f$ ‘ s]] **using** $*$ **by** *auto*
qed

lemma *continuous-open-preimage*:
assumes *continuous-on s f open s open t*
shows *open* $\{x \in s.\ f\,x \in t\}$

proof−
obtain T **where** T: *open* T $\{x \in s.\ f\ x \in t\} = s \cap T$
using *continuous-open-in-preimage*[*OF assms*(*1*,*3*)] **unfolding** *openin-open* **by** *auto*
thus *?thesis* **using** *open-Int*[*of s T*, *OF assms*(*2*)] **by** *auto*
qed

lemma *continuous-closed-preimage*:
assumes *continuous-on s f closed s closed t*
shows *closed* $\{x \in s.\ f\ x \in t\}$
proof−
obtain T **where** T: *closed* T $\{x \in s.\ f\ x \in t\} = s \cap T$
using *continuous-closed-in-preimage*[*OF assms*(*1*,*3*)] **unfolding** *closedin-closed* **by** *auto*
thus *?thesis* **using** *closed-Int*[*of s T*, *OF assms*(*2*)] **by** *auto*
qed

lemma *continuous-open-preimage-univ*:
shows $\forall x.$ *continuous* (*at x*) $f \Longrightarrow$ *open s* $\Longrightarrow$ *open* $\{x.\ f\ x \in s\}$
using *continuous-open-preimage*[*of UNIV f s*] *open-UNIV continuous-at-imp-continuous-on* **by** *auto*

lemma *continuous-closed-preimage-univ*:
shows $(\forall x.$ *continuous* (*at x*) $f) \Longrightarrow$ *closed s* ==> *closed* $\{x.\ f\ x \in s\}$
using *continuous-closed-preimage*[*of UNIV f s*] *closed-UNIV continuous-at-imp-continuous-on* **by** *auto*

lemma *continuous-open-vimage*:
shows $\forall x.$ *continuous* (*at x*) $f \Longrightarrow$ *open s* $\Longrightarrow$ *open* (*f* −‘ *s*)
unfolding *vimage-def* **by** (*rule continuous-open-preimage-univ*)

lemma *continuous-closed-vimage*:
shows $\forall x.$ *continuous* (*at x*) $f \Longrightarrow$ *closed s* $\Longrightarrow$ *closed* (*f* −‘ *s*)
unfolding *vimage-def* **by** (*rule continuous-closed-preimage-univ*)

lemma *interior-image-subset*:
assumes $\forall x.$ *continuous* (*at x*) *f inj f*
shows *interior* (*f* ‘ *s*) $\subseteq$ *f* ‘ (*interior s*)
apply *rule* **unfolding** *interior-def mem-Collect-eq image-iff* **apply** *safe*
proof− **fix** *x T* **assume** *as*:*open T x* $\in$ *T T* $\subseteq$ *f* ‘ *s*
hence *x* $\in$ *f* ‘ *s* **by** *auto* **then guess** *y* **unfolding** *image-iff* **.. note** *y*=*this*
thus $\exists xa \in \{x.\ \exists T.$ *open T* $\wedge$ *x* $\in$ *T* $\wedge$ *T* $\subseteq s\}.$ *x* = *f xa* **apply**(*rule-tac x*=*y* **in** *bexI*) **using** *assms as*
apply *safe* **apply**(*rule-tac x*=$\{x.\ f\ x \in T\}$ **in** *exI*) **apply**(*safe*,*rule continuous-open-preimage-univ*)
proof− **fix** *x* **assume** *f x* $\in$ *T* **hence** *f x* $\in$ *f* ‘ *s* **using** *as* **by** *auto*
thus *x* $\in$ *s* **unfolding** *inj-image-mem-iff*[*OF assms*(*2*)] **. qed** *auto* **qed**

Equality of continuous functions on closure and related results.

lemma *continuous-closed-in-preimage-constant*:

fixes f :: - ⇒ $'b$::*t1-space*
shows *continuous-on s f ==> closedin* (*subtopology euclidean s*) $\{x \in s.\ f\ x = a\}$
using *continuous-closed-in-preimage*[*of s f* $\{a\}$] **by** *auto*

lemma *continuous-closed-preimage-constant*:
fixes f :: - ⇒ $'b$::*t1-space*
shows *continuous-on s f* ⟹ *closed s ==> closed* $\{x \in s.\ f\ x = a\}$
using *continuous-closed-preimage*[*of s f* $\{a\}$] **by** *auto*

lemma *continuous-constant-on-closure*:
fixes f :: - ⇒ $'b$::*t1-space*
assumes *continuous-on* (*closure s*) f
$\forall x \in s.\ f\ x = a$
shows $\forall x \in$ (*closure s*). $f\ x = a$
using *continuous-closed-preimage-constant*[*of closure s f a*]
assms closure-minimal[*of s* $\{x \in$ *closure s*. $f\ x = a\}$] *closure-subset* **unfolding** *subset-eq* **by** *auto*

lemma *image-closure-subset*:
assumes *continuous-on* (*closure s*) f *closed t* $(f$ ‘ $s) \subseteq t$
shows f ‘ (*closure s*) $\subseteq t$
proof−
have $s \subseteq \{x \in$ *closure s*. $f\ x \in t\}$ **using** *assms*(*3*) *closure-subset* **by** *auto*
moreover have *closed* $\{x \in$ *closure s*. $f\ x \in t\}$
using *continuous-closed-preimage*[*OF assms*(*1*)] **and** *assms*(*2*) **by** *auto*
ultimately have *closure s* = $\{x \in$ *closure s* . $f\ x \in t\}$
using *closure-minimal*[*of s* $\{x \in$ *closure s*. $f\ x \in t\}$] **by** *auto*
thus *?thesis* **by** *auto*
qed

lemma *continuous-on-closure-norm-le*:
fixes f :: $'a$::*metric-space* ⇒ $'b$::*real-normed-vector*
assumes *continuous-on* (*closure s*) f $\forall y \in s.$ *norm*$(f\ y) \leq b$ $x \in$ (*closure s*)
shows *norm*$(f\ x) \leq b$
proof−
have ∗:f ‘ $s \subseteq$ *cball 0 b* **using** *assms*(*2*)[*unfolded mem-cball-0*[*THEN sym*]] **by** *auto*
show *?thesis*
using *image-closure-subset*[*OF assms*(*1*) *closed-cball*[*of 0 b*] ∗] *assms*(*3*)
unfolding *subset-eq* **apply**(*erule-tac x*=$f\ x$ **in** *ballE*) **by** (*auto simp add*: *dist-norm*)
qed

Making a continuous function avoid some value in a neighbourhood.

lemma *continuous-within-avoid*:
fixes f :: $'a$::*metric-space* ⇒ $'b$::*metric-space*
assumes *continuous* (*at x within s*) f $x \in s$ $f\ x \neq a$
shows $\exists e{>}0.\ \forall y \in s.$ *dist x y* $<$ *e* −−> $f\ y \neq a$

proof−
obtain *d* **where** *d>0* **and** *d*:∀ *xa*∈*s*. *0* < *dist xa x* ∧ *dist xa x* < *d* ⟶ *dist* (*f xa*) (*f x*) < *dist* (*f x*) *a*
using *assms*(*1*)[*unfolded continuous-within Lim-within*, *THEN spec*[**where** *x*=*dist* (*f x*) *a*]] *assms*(*3*)[*unfolded dist-nz*] **by** *auto*
{ fix *y* **assume** *y*∈*s* *dist x y* < *d*
hence *f y* ≠ *a* **using** *d*[*THEN bspec*[**where** *x*=*y*]] *assms*(*3*)[*unfolded dist-nz*]
apply *auto* **unfolding** *dist-nz*[*THEN sym*] **by** (*auto simp add*: *dist-commute*)
}
thus *?thesis* **using** ‹*d>0*› **by** *auto*
qed

lemma *continuous-at-avoid*:
fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*metric-space*
assumes *continuous* (*at x*) *f* *f x* ≠ *a*
shows ∃ *e*>*0*. ∀ *y*. *dist x y* < *e* ⟶ *f y* ≠ *a*
using *assms* **using** *continuous-within-avoid*[*of x UNIV f a*, *unfolded within-UNIV*]
by *auto*

lemma *continuous-on-avoid*:
fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*metric-space*
assumes *continuous-on s f* *x* ∈ *s* *f x* ≠ *a*
shows ∃ *e*>*0*. ∀ *y* ∈ *s*. *dist x y* < *e* ⟶ *f y* ≠ *a*
using *assms*(*1*)[*unfolded continuous-on-eq-continuous-within*, *THEN bspec*[**where** *x*=*x*], *OF assms*(*2*)] *continuous-within-avoid*[*of x s f a*] *assms*(*2*,*3*) **by** *auto*

lemma *continuous-on-open-avoid*:
fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*metric-space*
assumes *continuous-on s f* *open s* *x* ∈ *s* *f x* ≠ *a*
shows ∃ *e*>*0*. ∀ *y*. *dist x y* < *e* ⟶ *f y* ≠ *a*
using *assms*(*1*)[*unfolded continuous-on-eq-continuous-at*[*OF assms*(*2*)], *THEN bspec*[**where** *x*=*x*], *OF assms*(*3*)] *continuous-at-avoid*[*of x f a*] *assms*(*3*,*4*) **by** *auto*

Proving a function is constant by proving open-ness of level set.

lemma *continuous-levelset-open-in-cases*:
fixes *f* :: - ⇒ *'b*::*t1-space*
shows *connected s* ⟹ *continuous-on s f* ⟹
openin (*subtopology euclidean s*) {*x* ∈ *s*. *f x* = *a*}
==> (∀ *x* ∈ *s*. *f x* ≠ *a*) ∨ (∀ *x* ∈ *s*. *f x* = *a*)
unfolding *connected-clopen* **using** *continuous-closed-in-preimage-constant* **by** *auto*

lemma *continuous-levelset-open-in*:
fixes *f* :: - ⇒ *'b*::*t1-space*
shows *connected s* ⟹ *continuous-on s f* ⟹
openin (*subtopology euclidean s*) {*x* ∈ *s*. *f x* = *a*} ⟹
(∃ *x* ∈ *s*. *f x* = *a*) ==> (∀ *x* ∈ *s*. *f x* = *a*)
using *continuous-levelset-open-in-cases*[*of s f*]
by *meson*

lemma *continuous-levelset-open*:
fixes f :: - ⇒ 'b::*t1-space*
assumes *connected* s *continuous-on* s f *open* $\{x \in s.\ f\ x = a\}$ $\exists x \in s.\ f\ x = a$
shows $\forall x \in s.\ f\ x = a$
using *continuous-levelset-open-in*[*OF assms*(*1*,*2*), *of a*, *unfolded openin-open*] **using** *assms* (*3*,*4*) **by** *fast*

Some arithmetical combinations (more to prove).

lemma *open-scaling*[*intro*]:
fixes s :: 'a::*real-normed-vector set*
assumes $c \neq 0$ *open* s
shows *open*$((\lambda x.\ c *_R x)\ `\ s)$
proof−
{ fix x **assume** $x \in s$
then obtain e **where** $e{>}0$ **and** $e{:}\forall x'.\ dist\ x'\ x < e \longrightarrow x' \in s$ **using** *assms*(*2*)[*unfolded open-dist*, *THEN bspec*[**where** $x{=}x$]] **by** *auto*
have $e * abs\ c > 0$ **using** *assms*(*1*)[*unfolded zero-less-abs-iff*[*THEN sym*]] **using** *mult-pos-pos*[*OF* ‹$e{>}0$›] **by** *auto*
moreover
{ fix y **assume** $dist\ y\ (c *_R x) < e * |c|$
hence $norm\ ((1\ /\ c) *_R y - x) < e$ **unfolding** *dist-norm*
using *norm-scaleR*[*of c* (*1* / *c*) *∗R y* − *x*, *unfolded scaleR-right-diff-distrib*, *unfolded scaleR-scaleR*] *assms*(*1*)
assms(*1*)[*unfolded zero-less-abs-iff*[*THEN sym*]] **by** (*simp del*:*zero-less-abs-iff*)
hence $y \in op *_R c\ `\ s$ **using** *rev-image-eqI*[*of* (*1* / *c*) *∗R y s y op ∗R c*] *e*[*THEN spec*[**where** $x{=}(1\ /\ c) *_R y$]] *assms*(*1*) **unfolding** *dist-norm scaleR-scaleR* **by** *auto* **}**
ultimately have $\exists e{>}0.\ \forall x'.\ dist\ x'\ (c *_R x) < e \longrightarrow x' \in op *_R c\ `\ s$ **apply**(*rule-tac x*=*e ∗ abs c* **in** *exI*) **by** *auto* **}**
thus *?thesis* **unfolding** *open-dist* **by** *auto*
qed

lemma *minus-image-eq-vimage*:
fixes A :: 'a::*ab-group-add set*
shows $(\lambda x.\ -x)\ `\ A = (\lambda x.\ -x) -`\ A$
by (*auto intro*!: *image-eqI* [**where** $f{=}\lambda x.\ -x$])

lemma *open-negations*:
fixes s :: 'a::*real-normed-vector set*
shows *open* s ==> *open* $((\lambda\ x.\ -x)\ `\ s)$
unfolding *scaleR-minus1-left* [*symmetric*]
by (*rule open-scaling*, *auto*)

lemma *open-translation*:
fixes s :: 'a::*real-normed-vector set*
assumes *open* s **shows** *open*$((\lambda x.\ a + x)\ `\ s)$
proof−
{ fix x **have** *continuous* (*at x*) $(\lambda x.\ x - a)$ **using** *continuous-sub*[*of at x* λ*x*. *x*

$\lambda x.\ a]$ *continuous-at-id*[*of x*] *continuous-const*[*of at x a*] **by** *auto* **}**
moreover have $\{x.\ x - a \in s\} = op + a\ `\ s$ **apply** *auto* **unfolding** *image-iff* **apply**(*rule-tac x=x − a* **in** *bexI*) **by** *auto*
ultimately show *?thesis* **using** *continuous-open-preimage-univ*[*of* $\lambda x.\ x - a$ *s*] **using** *assms* **by** *auto*
qed

lemma *open-affinity*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *open s* $c \neq 0$
shows *open* $((\lambda x.\ a + c *_R x)\ `\ s)$
proof−
have $*$:$(\lambda x.\ a + c *_R x) = (\lambda x.\ a + x) \circ (\lambda x.\ c *_R x)$ **unfolding** *o-def* **..**
have $op + a\ `\ op *_R c\ `\ s = (op + a \circ op *_R c)\ `\ s$ **by** *auto*
thus *?thesis* **using** *assms open-translation*[*of op* $*_R$ *c* ‘ *s a*] **unfolding** $*$ **by** *auto*
qed

lemma *interior-translation*:
fixes *s* :: *'a*::*real-normed-vector set*
shows *interior* $((\lambda x.\ a + x)\ `\ s) = (\lambda x.\ a + x)\ `\ (interior\ s)$
proof (*rule set-ext*, *rule*)
fix *x* **assume** $x \in interior\ (op + a\ `\ s)$
then obtain *e* **where** *e*>*0* **and** *e*:*ball x e* $\subseteq op + a\ `\ s$ **unfolding** *mem-interior* **by** *auto*
hence $ball\ (x - a)\ e \subseteq s$ **unfolding** *subset-eq Ball-def mem-ball dist-norm* **apply** *auto* **apply**(*erule-tac x=a + xa* **in** *allE*) **unfolding** *ab-group-add-class.diff-diff-eq*[*THEN sym*] **by** *auto*
thus $x \in op + a\ `\ interior\ s$ **unfolding** *image-iff* **apply**(*rule-tac x=x − a* **in** *bexI*) **unfolding** *mem-interior* **using** ‹$e > 0$› **by** *auto*
next
fix *x* **assume** $x \in op + a\ `\ interior\ s$
then obtain *y e* **where** *e*>*0* **and** *e*:*ball y e* $\subseteq s$ **and** *y*:$x = a + y$ **unfolding** *image-iff Bex-def mem-interior* **by** *auto*
{ fix *z* **have** $*$:$a + y - z = y + a - z$ **by** *auto*
assume *z*∈*ball x e*
hence $z - a \in s$ **using** *e*[*unfolded subset-eq*, *THEN bspec*[**where** *x=z − a*]] **unfolding** *mem-ball dist-norm y ab-group-add-class.diff-diff-eq2* $*$ **by** *auto*
hence $z \in op + a\ `\ s$ **unfolding** *image-iff* **by**(*auto intro*!: *bexI*[**where** *x=z − a*]) **}**
hence *ball x e* $\subseteq op + a\ `\ s$ **unfolding** *subset-eq* **by** *auto*
thus $x \in interior\ (op + a\ `\ s)$ **unfolding** *mem-interior* **using** ‹*e*>*0*› **by** *auto*
qed

We can now extend limit compositions to consider the scalar multiplier.

lemma *continuous-vmul*:
fixes *c* :: *'a*::*metric-space* ⇒ *real* **and** *v* :: *'b*::*real-normed-vector*
shows *continuous net c* ==> *continuous net* $(\lambda x.\ c(x) *_R v)$
unfolding *continuous-def* **using** *Lim-vmul*[*of c*] **by** *auto*

lemma *continuous-mul*:
fixes $c :: 'a$::*metric-space* $\Rightarrow$ *real*
fixes $f :: 'a$::*metric-space* $\Rightarrow$ $'b$::*real-normed-vector*
shows *continuous net c* $\Longrightarrow$ *continuous net f*
$\Longrightarrow$ *continuous net* $(\lambda x.\ c(x) *_R f\ x)$
unfolding *continuous-def* **by** (*intro tendsto-intros*)

lemmas *continuous-intros = continuous-add continuous-vmul continuous-cmul continuous-const continuous-sub continuous-at-id continuous-within-id continuous-mul*

lemma *continuous-on-vmul*:
fixes $c :: 'a$::*metric-space* $\Rightarrow$ *real* **and** $v :: 'b$::*real-normed-vector*
shows *continuous-on s c* $\Longrightarrow$ *continuous-on s* $(\lambda x.\ c(x) *_R v)$
unfolding *continuous-on-eq-continuous-within* **using** *continuous-vmul*[*of - c*] **by** *auto*

lemma *continuous-on-mul*:
fixes $c :: 'a$::*metric-space* $\Rightarrow$ *real*
fixes $f :: 'a$::*metric-space* $\Rightarrow$ $'b$::*real-normed-vector*
shows *continuous-on s c* $\Longrightarrow$ *continuous-on s f*
$\Longrightarrow$ *continuous-on s* $(\lambda x.\ c(x) *_R f\ x)$
unfolding *continuous-on-eq-continuous-within* **using** *continuous-mul*[*of - c*] **by** *auto*

lemmas *continuous-on-intros = continuous-on-add continuous-on-const continuous-on-id continuous-on-compose continuous-on-cmul continuous-on-neg continuous-on-sub uniformly-continuous-on-add uniformly-continuous-on-const uniformly-continuous-on-id uniformly-continuous-on-compose uniformly-continuous-on-cmul uniformly-continuous-on-neg uniformly-continuous-on-sub continuous-on-mul continuous-on-vmul*

And so we have continuity of inverse.

lemma *continuous-inv*:
fixes $f :: 'a$::*metric-space* $\Rightarrow$ *real*
shows *continuous net f* $\Longrightarrow$ f(*netlimit net*) $\neq$ *0*
$\Longrightarrow$ *continuous net* (*inverse o f*)
unfolding *continuous-def* **using** *Lim-inv* **by** *auto*

lemma *continuous-at-within-inv*:
fixes $f :: 'a$::*metric-space* $\Rightarrow$ $'b$::*real-normed-field*
assumes *continuous* (*at a within s*) *f f a* $\neq$ *0*
shows *continuous* (*at a within s*) (*inverse o f*)
using *assms* **unfolding** *continuous-within o-def*
by (*intro tendsto-intros*)

lemma *continuous-at-inv*:
fixes $f :: 'a$::*metric-space* $\Rightarrow$ $'b$::*real-normed-field*
shows *continuous* (*at a*) *f* $\Longrightarrow$ *f a* $\neq$ *0*
$\Longrightarrow$ *continuous* (*at a*) (*inverse o f*)

using *within-UNIV*[*THEN sym*, *of at a*] **using** *continuous-at-within-inv*[*of a UNIV*] **by** *auto*

Topological properties of linear functions.

lemma *linear-lim-0*:
assumes *bounded-linear f* **shows** (*f* ---> *0*) (*at* (*0*))
proof−
interpret *f*: *bounded-linear f* **by** *fact*
have (*f* ---> *f 0*) (*at 0*)
using *tendsto-ident-at* **by** (*rule f.tendsto*)
thus *?thesis* **unfolding** *f.zero* **.**
qed

lemma *linear-continuous-at*:
assumes *bounded-linear f* **shows** *continuous* (*at a*) *f*
unfolding *continuous-at* **using** *assms*
apply (*rule bounded-linear.tendsto*)
apply (*rule tendsto-ident-at*)
done

lemma *linear-continuous-within*:
shows *bounded-linear f* ==> *continuous* (*at x within s*) *f*
using *continuous-at-imp-continuous-within*[*of x f s*] **using** *linear-continuous-at*[*of f*] **by** *auto*

lemma *linear-continuous-on*:
shows *bounded-linear f* ==> *continuous-on s f*
using *continuous-at-imp-continuous-on*[*of s f*] **using** *linear-continuous-at*[*of f*] **by** *auto*

Also bilinear functions, in composition form.

lemma *bilinear-continuous-at-compose*:
shows *continuous* (*at x*) *f* ⟹ *continuous* (*at x*) *g* ⟹ *bounded-bilinear h*
==> *continuous* (*at x*) (λ*x*. *h* (*f x*) (*g x*))
unfolding *continuous-at* **using** *Lim-bilinear*[*of f f x* (*at x*) *g g x h*] **by** *auto*

lemma *bilinear-continuous-within-compose*:
shows *continuous* (*at x within s*) *f* ⟹ *continuous* (*at x within s*) *g* ⟹ *bounded-bilinear h*
==> *continuous* (*at x within s*) (λ*x*. *h* (*f x*) (*g x*))
unfolding *continuous-within* **using** *Lim-bilinear*[*of f f f x*] **by** *auto*

lemma *bilinear-continuous-on-compose*:
shows *continuous-on s f* ⟹ *continuous-on s g* ⟹ *bounded-bilinear h*
==> *continuous-on s* (λ*x*. *h* (*f x*) (*g x*))
unfolding *continuous-on-def*
by (*fast elim*: *bounded-bilinear.tendsto*)

Preservation of compactness and connectedness under continuous function.

lemma *compact-continuous-image*:
assumes *continuous-on s f* *compact s*
shows *compact*(*f ' s*)
proof−
{ fix *x* **assume** *x*:∀ *n*::*nat*. *x n* ∈ *f ' s*
then obtain *y* **where** *y*:∀ *n*. *y n* ∈ *s* ∧ *x n* = *f* (*y n*) **unfolding** *image-iff Bex-def* **using** *choice*[*of* λ*n xa*. *xa* ∈ *s* ∧ *x n* = *f xa*] **by** *auto*
then obtain *l r* **where** *l*∈*s* **and** *r*:*subseq r* **and** *lr*:((*y* ∘ *r*) −−−> *l*) *sequentially* **using** *assms*(*2*)[*unfolded compact-def*, *THEN spec*[**where** *x*=*y*]] **by** *auto*
{ fix *e*::*real* **assume** *e*>*0*
then obtain *d* **where** *d*>*0* **and** *d*:∀ *x'*∈*s*. *dist x' l* < *d* ⟶ *dist* (*f x'*) (*f l*) < *e* **using** *assms*(*1*)[*unfolded continuous-on-iff*, *THEN bspec*[**where** *x*=*l*], *OF* ‹*l*∈*s*›] **by** *auto*
then obtain *N*::*nat* **where** *N*:∀ *n*≥*N*. *dist* ((*y* ∘ *r*) *n*) *l* < *d* **using** *lr*[*unfolded Lim-sequentially*, *THEN spec*[**where** *x*=*d*]] **by** *auto*
{ fix *n*::*nat* **assume** *n*≥*N* **hence** *dist* ((*x* ∘ *r*) *n*) (*f l*) < *e* **using** *N*[*THEN spec*[**where** *x*=*n*]] *d*[*THEN bspec*[**where** *x*=*y* (*r n*)]] *y*[*THEN spec*[**where** *x*=*r n*]] **by** *auto* **}**
hence ∃ *N*. ∀ *n*≥*N*. *dist* ((*x* ∘ *r*) *n*) (*f l*) < *e* **by** *auto* **}**
hence ∃ *l*∈*f ' s*. ∃ *r*. *subseq r* ∧ ((*x* ∘ *r*) −−−> *l*) *sequentially* **unfolding** *Lim-sequentially* **using** *r lr* ‹*l*∈*s*› **by** *auto* **}**
thus *?thesis* **unfolding** *compact-def* **by** *auto*
qed

lemma *connected-continuous-image*:
assumes *continuous-on s f* *connected s*
shows *connected*(*f ' s*)
proof−
{ fix *T* **assume** *as*: *T* ≠ {} *T* ≠ *f ' s* *openin* (*subtopology euclidean* (*f ' s*)) *T* *closedin* (*subtopology euclidean* (*f ' s*)) *T*
have {*x* ∈ *s*. *f x* ∈ *T*} = {} ∨ {*x* ∈ *s*. *f x* ∈ *T*} = *s*
using *assms*(*1*)[*unfolded continuous-on-open*, *THEN spec*[**where** *x*=*T*]]
using *assms*(*1*)[*unfolded continuous-on-closed*, *THEN spec*[**where** *x*=*T*]]
using *assms*(*2*)[*unfolded connected-clopen*, *THEN spec*[**where** *x*={*x* ∈ *s*. *f x* ∈ *T*}]] *as*(*3*,*4*) **by** *auto*
hence *False* **using** *as*(*1*,*2*)
using *as*(*4*)[*unfolded closedin-def topspace-euclidean-subtopology*] **by** *auto* **}**
thus *?thesis* **unfolding** *connected-clopen* **by** *auto*
qed

Continuity implies uniform continuity on a compact domain.

lemma *compact-uniformly-continuous*:
assumes *continuous-on s f* *compact s*
shows *uniformly-continuous-on s f*
proof−
{ fix *x* **assume** *x*:*x*∈*s*
hence ∀ *xa*. ∃ *y*. *0* < *xa* ⟶ (*y* > *0* ∧ (∀ *x'*∈*s*. *dist x' x* < *y* ⟶ *dist* (*f x'*) (*f x*) < *xa*)) **using** *assms*(*1*)[*unfolded continuous-on-iff*, *THEN bspec*[**where** *x*=*x*]] **by** *auto*

hence $\exists fa.\ \forall xa{>}0.\ \forall x'{\in}s.\ fa\ xa > 0 \wedge (dist\ x'\ x < fa\ xa \longrightarrow dist\ (f\ x')\ (f\ x) < xa)$ **using** *choice*[*of* $\lambda e\ d.\ e{>}0 \longrightarrow d{>}0 \wedge(\forall x'{\in}s.\ (dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e))$] **by** *auto* **}**
then have $\forall x{\in}s.\ \exists y.\ \forall xa.\ 0 < xa \longrightarrow (\forall x'{\in}s.\ y\ xa > 0 \wedge (dist\ x'\ x < y\ xa \longrightarrow dist\ (f\ x')\ (f\ x) < xa))$ **by** *auto*
then obtain *d* **where** $d{:}\forall e{>}0.\ \forall x{\in}s.\ \forall x'{\in}s.\ d\ x\ e > 0 \wedge (dist\ x'\ x < d\ x\ e \longrightarrow dist\ (f\ x')\ (f\ x) < e)$
using *bchoice*[*of s* $\lambda x\ fa.\ \forall xa{>}0.\ \forall x'{\in}s.\ fa\ xa > 0 \wedge (dist\ x'\ x < fa\ xa \longrightarrow dist\ (f\ x')\ (f\ x) < xa)$] **by** *blast*

{ fix *e*::*real* **assume** *e*>*0*

{ fix *x* **assume** *x*∈*s* **hence** *x* ∈ *ball x* (*d x* (*e* / *2*)) **unfolding** *centre-in-ball* **using** *d*[*THEN spec*[**where** *x*=*e*/*2*]] **using** ‹*e*>*0*› **by** *auto* **}**
hence *s* ⊆ ⋃{*ball x* (*d x* (*e* / *2*)) |*x*. *x* ∈ *s*} **unfolding** *subset-eq* **by** *auto*
moreover
{ fix *b* **assume** *b*∈{*ball x* (*d x* (*e* / *2*)) |*x*. *x* ∈ *s*} **hence** *open b* **by** *auto* **}**
ultimately obtain *ea* **where** *ea*>*0* **and** *ea*:∀ *x*∈*s*. ∃ *b*∈{*ball x* (*d x* (*e* / *2*)) |*x*. *x* ∈ *s*}. *ball x ea* ⊆ *b* **using** *heine-borel-lemma*[*OF assms*(*2*), *of* {*ball x* (*d x* (*e* / *2*)) | *x*. *x*∈*s* }] **by** *auto*

{ fix *x y* **assume** *x*∈*s y*∈*s* **and** *as*:*dist y x* < *ea*
obtain *z* **where** *z*∈*s* **and** *z*:*ball x ea* ⊆ *ball z* (*d z* (*e* / *2*)) **using** *ea*[*THEN bspec*[**where** *x*=*x*]] **and** ‹*x*∈*s*› **by** *auto*
hence *x*∈*ball z* (*d z* (*e* / *2*)) **using** ‹*ea*>*0*› **unfolding** *subset-eq* **by** *auto*
hence *dist* (*f z*) (*f x*) < *e* / *2* **using** *d*[*THEN spec*[**where** *x*=*e*/*2*]] **and** ‹*e*>*0*› **and** ‹*x*∈*s*› **and** ‹*z*∈*s*›
by (*auto simp add*: *dist-commute*)
moreover have *y*∈*ball z* (*d z* (*e* / *2*)) **using** *as* **and** ‹*ea*>*0*› **and** *z*[*unfolded subset-eq*]
by (*auto simp add*: *dist-commute*)
hence *dist* (*f z*) (*f y*) < *e* / *2* **using** *d*[*THEN spec*[**where** *x*=*e*/*2*]] **and** ‹*e*>*0*› **and** ‹*y*∈*s*› **and** ‹*z*∈*s*›
by (*auto simp add*: *dist-commute*)
ultimately have *dist* (*f y*) (*f x*) < *e* **using** *dist-triangle-half-r*[*of f z f x e f y*]
by (*auto simp add*: *dist-commute*) **}**
then have ∃ *d*>*0*. ∀ *x*∈*s*. ∀ *x'*∈*s*. *dist x' x* < *d* ⟶ *dist* (*f x'*) (*f x*) < *e* **using** ‹*ea*>*0*› **by** *auto* **}**
thus *?thesis* **unfolding** *uniformly-continuous-on-def* **by** *auto*
qed

Continuity of inverse function on compact domain.

lemma *continuous-on-inverse*:
fixes *f* :: *'a*::*heine-borel* ⇒ *'b*::*heine-borel*

assumes *continuous-on s f* *compact s* ∀ *x* ∈ *s*. *g* (*f x*) = *x*
shows *continuous-on* (*f* ‘ *s*) *g*
proof−
have ∗:*g* ‘ *f* ‘ *s* = *s* **using** *assms*(*3*) **by** (*auto simp add*: *image-iff*)

{ fix t **assume** t:*closedin* (*subtopology euclidean* (g ‘ f ‘ s)) t
then obtain T **where** T: *closed* T t = $s \cap T$ **unfolding** *closedin-closed* **unfolding** $*$ **by** *auto*
have *continuous-on* ($s \cap T$) f **using** *continuous-on-subset*[*OF assms*(*1*), *of* $s \cap t$]
unfolding $T(2)$ **and** *Int-left-absorb* **by** *auto*
moreover have *compact* ($s \cap T$)
using *assms*(*2*) **unfolding** *compact-eq-bounded-closed*
using *bounded-subset*[*of* s $s \cap T$] **and** $T(1)$ **by** *auto*
ultimately have *closed* (f ‘ t) **using** $T(1)$ **unfolding** $T(2)$
using *compact-continuous-image* [*of* $s \cap T$ f] **unfolding** *compact-eq-bounded-closed* **by** *auto*
moreover have $\{x \in f$ ‘ $s.\ g\ x \in t\} = f$ ‘ $s \cap f$ ‘ t **using** *assms*(*3*) **unfolding** $T(2)$ **by** *auto*
ultimately have *closedin* (*subtopology euclidean* (f ‘ s)) $\{x \in f$ ‘ $s.\ g\ x \in t\}$
unfolding *closedin-closed* **by** *auto* **}**
thus *?thesis* **unfolding** *continuous-on-closed* **by** *auto*
qed

A uniformly convergent limit of continuous functions is continuous.

lemma *norm-triangle-lt*:
fixes $x\ y$ :: $'a$::*real-normed-vector*
shows *norm* x + *norm* $y < e \Longrightarrow$ *norm* ($x + y$) $< e$
by (*rule le-less-trans* [*OF norm-triangle-ineq*])

lemma *continuous-uniform-limit*:
fixes f :: $'a \Rightarrow 'b$::*metric-space* $\Rightarrow 'c$::*real-normed-vector*
assumes ¬ (*trivial-limit net*) *eventually* ($\lambda n.$ *continuous-on* s ($f\ n$)) *net*
$\forall\, e{>}0.$ *eventually* ($\lambda n.\ \forall\, x \in s.$ *norm*($f\ n\ x - g\ x$) $< e$) *net*
shows *continuous-on* s g
proof−
{ fix x **and** e::*real* **assume** $x{\in}s$ $e{>}0$
have *eventually* ($\lambda n.\ \forall\, x{\in}s.$ *norm* ($f\ n\ x - g\ x$) $<$ e / 3) *net* **using** ‹$e{>}0$› *assms*(*3*)[*THEN spec*[**where** $x{=}e/3$]] **by** *auto*
then obtain n **where** n:$\forall\, xa{\in}s.$ *norm* ($f\ n\ xa - g\ xa$) $<$ e / 3 *continuous-on* s ($f\ n$)
using *eventually-and*[*of* ($\lambda n.\ \forall\, x{\in}s.$ *norm* ($f\ n\ x - g\ x$) $<$ e / 3) ($\lambda n.$ *continuous-on* s ($f\ n$)) *net*] *assms*(*1*,*2*) *eventually-happens* **by** *blast*
have e / $3 > 0$ **using** ‹$e{>}0$› **by** *auto*
then obtain d **where** $d{>}0$ **and** d:$\forall\, x'{\in}s.$ *dist* x' $x < d \longrightarrow$ *dist* ($f\ n\ x'$) ($f\ n$ x) $<$ e / 3
using $n(2)$[*unfolded continuous-on-iff*, *THEN bspec*[**where** $x{=}x$], *OF* ‹$x{\in}s$›, *THEN spec*[**where** $x{=}e/3$]] **by** *blast*
{ fix y **assume** $y{\in}s$ *dist* y $x < d$
hence *dist* ($f\ n\ y$) ($f\ n\ x$) $<$ e / 3 **using** d[*THEN bspec*[**where** $x{=}y$]] **by** *auto*
hence *norm* ($f\ n\ y - g\ x$) $<$ $2 * e$ / 3 **using** *norm-triangle-lt*[*of* $f\ n\ y - f\ n\ x$ $f\ n\ x - g\ x$ $2{*}e/3$]
using $n(1)$[*THEN bspec*[**where** $x{=}x$], *OF* ‹$x{\in}s$›] **unfolding** *dist-norm*

unfolding *ab-group-add-class.ab-diff-minus* **by** *auto*
hence *dist* (*g y*) (*g x*) < *e* **unfolding** *dist-norm* **using** *n*(*1*)[*THEN bspec*[**where** *x*=*y*], *OF* ‹*y*∈*s*›]
unfolding *norm-minus-cancel*[*of f n y* − *g y*, *THEN sym*] **using** *norm-triangle-lt*[*of f n y* − *g x g y* − *f n y e*] **by** (*auto simp add*: *uminus-add-conv-diff*) **}**
hence ∃ *d*>*0*. ∀ *x'*∈*s*. *dist x' x* < *d* ⟶ *dist* (*g x'*) (*g x*) < *e* **using** ‹*d*>*0*› **by** *auto* **}**
thus *?thesis* **unfolding** *continuous-on-iff* **by** *auto*
qed

16.21 Topological stuff lifted from and dropped to R

lemma *open-real*:
fixes *s* :: *real set* **shows**
open s ⟷
(∀ *x* ∈ *s*. ∃ *e*>*0*. ∀ *x'*. *abs*(*x'* − *x*) < *e* −−> *x'* ∈ *s*) (**is** *?lhs* = *?rhs*)
unfolding *open-dist dist-norm* **by** *simp*

lemma *islimpt-approachable-real*:
fixes *s* :: *real set*
shows *x islimpt s* ⟷ (∀ *e*>*0*. ∃ *x'*∈ *s*. *x'* ≠ *x* ∧ *abs*(*x'* − *x*) < *e*)
unfolding *islimpt-approachable dist-norm* **by** *simp*

lemma *closed-real*:
fixes *s* :: *real set*
shows *closed s* ⟷
(∀ *x*. (∀ *e*>*0*. ∃ *x'* ∈ *s*. *x'* ≠ *x* ∧ *abs*(*x'* − *x*) < *e*)
−−> *x* ∈ *s*)
unfolding *closed-limpt islimpt-approachable dist-norm* **by** *simp*

lemma *continuous-at-real-range*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *real*
shows *continuous* (*at x*) *f* ⟷ (∀ *e*>*0*. ∃ *d*>*0*.
∀ *x'*. *norm*(*x'* − *x*) < *d* −−> *abs*(*f x'* − *f x*) < *e*)
unfolding *continuous-at* **unfolding** *Lim-at*
unfolding *dist-nz*[*THEN sym*] **unfolding** *dist-norm* **apply** *auto*
apply(*erule-tac x*=*e* **in** *allE*) **apply** *auto* **apply** (*rule-tac x*=*d* **in** *exI*) **apply** *auto* **apply** (*erule-tac x*=*x'* **in** *allE*) **apply** *auto*
apply(*erule-tac x*=*e* **in** *allE*) **by** *auto*

lemma *continuous-on-real-range*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *real*
shows *continuous-on s f* ⟷ (∀ *x* ∈ *s*. ∀ *e*>*0*. ∃ *d*>*0*. (∀ *x'* ∈ *s*. *norm*(*x'* − *x*) < *d* −−> *abs*(*f x'* − *f x*) < *e*))
unfolding *continuous-on-iff dist-norm* **by** *simp*

lemma *continuous-at-norm*: *continuous* (*at x*) *norm*
unfolding *continuous-at* **by** (*intro tendsto-intros*)

lemma *continuous-on-norm*: *continuous-on s norm*
unfolding *continuous-on* **by** (*intro ballI tendsto-intros*)

lemma *continuous-at-component*: *continuous* (*at a*) (λ*x*. *x* $ *i*)
unfolding *continuous-at* **by** (*intro tendsto-intros*)

lemma *continuous-on-component*: *continuous-on s* (λ*x*. *x* $ *i*)
unfolding *continuous-on-def* **by** (*intro ballI tendsto-intros*)

lemma *continuous-at-infnorm*: *continuous* (*at x*) *infnorm*
unfolding *continuous-at Lim-at o-def* **unfolding** *dist-norm*
apply *auto* **apply** (*rule-tac x=e* **in** *exI*) **apply** *auto*
using *order-trans*[*OF real-abs-sub-infnorm infnorm-le-norm*, *of - x*] **by** (*metis xt1*(7))

Hence some handy theorems on distance, diameter etc. of/from a set.

lemma *compact-attains-sup*:
fixes *s* :: *real set*
assumes *compact s* *s* ≠ {}
shows ∃ *x* ∈ *s*. ∀ *y* ∈ *s*. *y* ≤ *x*
proof−
from *assms*(1) **have** *a*:*bounded s closed s* **unfolding** *compact-eq-bounded-closed* **by** *auto*
{ fix *e*::*real* **assume** *as*: ∀ *x*∈*s*. *x* ≤ *Sup s Sup s* ∉ *s* 0 < *e* ∀ *x'*∈*s*. *x'* = *Sup s* ∨ ¬ *Sup s* − *x'* < *e*
have *isLub UNIV s* (*Sup s*) **using** *Sup*[*OF assms*(2)] **unfolding** *setle-def* **using** *as*(1) **by** *auto*
moreover have *isUb UNIV s* (*Sup s* − *e*) **unfolding** *isUb-def* **unfolding** *setle-def* **using** *as*(4,2) **by** *auto*
ultimately have *False* **using** *isLub-le-isUb*[*of UNIV s Sup s Sup s* − *e*] **using** ‹*e*>0› **by** *auto* **}**
thus *?thesis* **using** *bounded-has-Sup*(1)[*OF a*(1) *assms*(2)] **using** *a*(2)[*unfolded closed-real*, *THEN spec*[**where** *x=Sup s*]]
apply(*rule-tac x=Sup s* **in** *bexI*) **by** *auto*
qed

lemma *Inf*:
fixes *S* :: *real set*
shows *S* ≠ {} ==> (∃ *b*. *b* <=* *S*) ==> *isGlb UNIV S* (*Inf S*)
by (*auto simp add*: *isLb-def setle-def setge-def isGlb-def greatestP-def*)

lemma *compact-attains-inf*:
fixes *s* :: *real set*
assumes *compact s s* ≠ {} **shows** ∃ *x* ∈ *s*. ∀ *y* ∈ *s*. *x* ≤ *y*
proof−
from *assms*(1) **have** *a*:*bounded s closed s* **unfolding** *compact-eq-bounded-closed* **by** *auto*
{ fix *e*::*real* **assume** *as*: ∀ *x*∈*s*. *x* ≥ *Inf s* *Inf s* ∉ *s* 0 < *e*
∀ *x'*∈*s*. *x'* = *Inf s* ∨ ¬ *abs* (*x'* − *Inf s*) < *e*

have *isGlb UNIV s* (*Inf s*) **using** *Inf*[*OF assms*(*2*)] **unfolding** *setge-def* **using** *as*(*1*) **by** *auto*
moreover
{ fix *x* **assume** *x* ∈ *s*
hence ∗:*abs* (*x* − *Inf s*) = *x* − *Inf s* **using** *as*(*1*)[*THEN bspec*[**where** *x*=*x*]] **by** *auto*
have *Inf s* + *e* ≤ *x* **using** *as*(*4*)[*THEN bspec*[**where** *x*=*x*]] **using** *as*(*2*) ‹*x*∈*s*› **unfolding** ∗ **by** *auto* **}**
hence *isLb UNIV s* (*Inf s* + *e*) **unfolding** *isLb-def* **and** *setge-def* **by** *auto*
ultimately have *False* **using** *isGlb-le-isLb*[*of UNIV s Inf s Inf s* + *e*] **using** ‹*e*>*0*› **by** *auto* **}**
thus *?thesis* **using** *bounded-has-Inf*(*1*)[*OF a*(*1*) *assms*(*2*)] **using** *a*(*2*)[*unfolded closed-real*, *THEN spec*[**where** *x*=*Inf s*]]
apply(*rule-tac x*=*Inf s* **in** *bexI*) **by** *auto*
qed

lemma *continuous-attains-sup*:
fixes *f* :: *'a*::*metric-space* ⇒ *real*
shows *compact s* ⟹ *s* ≠ {} ⟹ *continuous-on s f*
==> (∃ *x* ∈ *s*. ∀ *y* ∈ *s*. *f y* ≤ *f x*)
using *compact-attains-sup*[*of f ' s*]
using *compact-continuous-image*[*of s f*] **by** *auto*

lemma *continuous-attains-inf*:
fixes *f* :: *'a*::*metric-space* ⇒ *real*
shows *compact s* ⟹ *s* ≠ {} ⟹ *continuous-on s f*
⟹ (∃ *x* ∈ *s*. ∀ *y* ∈ *s*. *f x* ≤ *f y*)
using *compact-attains-inf*[*of f ' s*]
using *compact-continuous-image*[*of s f*] **by** *auto*

lemma *distance-attains-sup*:
assumes *compact s s* ≠ {}
shows ∃ *x* ∈ *s*. ∀ *y* ∈ *s*. *dist a y* ≤ *dist a x*
proof (*rule continuous-attains-sup* [*OF assms*])
{ fix *x* **assume** *x*∈*s*
have (*dist a* −−−> *dist a x*) (*at x within s*)
by (*intro tendsto-dist tendsto-const Lim-at-within Lim-ident-at*)
}
thus *continuous-on s* (*dist a*)
unfolding *continuous-on* **..**
qed

For *minimal* distance, we only need closure, not compactness.

lemma *distance-attains-inf*:
fixes *a* :: *'a*::*heine-borel*
assumes *closed s* *s* ≠ {}
shows ∃ *x* ∈ *s*. ∀ *y* ∈ *s*. *dist a x* ≤ *dist a y*
proof−
from *assms*(*2*) **obtain** *b* **where** *b*∈*s* **by** *auto*

let *?B* = *cball a* (*dist b a*) ∩ *s*
have *b* ∈ *?B* **using** ‹*b*∈*s*› **by** (*simp add*: *dist-commute*)
hence *?B* ≠ {} **by** *auto*
moreover
{ fix *x* **assume** *x*∈*?B*
fix *e*::*real* **assume** *e*>*0*
{ fix *x'* **assume** *x'*∈*?B* **and** *as*:*dist x' x* < *e*
from *as* **have** |*dist a x'* − *dist a x*| < *e*
unfolding *abs-less-iff minus-diff-eq*
using *dist-triangle2* [*of a x' x*]
using *dist-triangle* [*of a x x'*]
by *arith*
}
hence ∃ *d*>*0*. ∀ *x'*∈*?B*. *dist x' x* < *d* ⟶ |*dist a x'* − *dist a x*| < *e*
using ‹*e*>*0*› **by** *auto*
}
hence *continuous-on* (*cball a* (*dist b a*) ∩ *s*) (*dist a*)
unfolding *continuous-on Lim-within dist-norm real-norm-def*
by *fast*
moreover have *compact ?B*
using *compact-cball*[*of a dist b a*]
unfolding *compact-eq-bounded-closed*
using *bounded-Int* **and** *closed-Int* **and** *assms*(*1*) **by** *auto*
ultimately obtain *x* **where** *x*∈*cball a* (*dist b a*) ∩ *s* ∀ *y*∈*cball a* (*dist b a*) ∩ *s*. *dist a x* ≤ *dist a y*
using *continuous-attains-inf*[*of ?B dist a*] **by** *fastsimp*
thus *?thesis* **by** *fastsimp*
qed

16.22 Pasted sets

lemma *bounded-Times*:
assumes *bounded s bounded t* **shows** *bounded* (*s* × *t*)
proof−
obtain *x y a b* **where** ∀ *z*∈*s*. *dist x z* ≤ *a* ∀ *z*∈*t*. *dist y z* ≤ *b*
using *assms* [*unfolded bounded-def*] **by** *auto*
then have ∀ *z*∈*s* × *t*. *dist* (*x*, *y*) *z* ≤ *sqrt* (*a*² + *b*²)
by (*auto simp add*: *dist-Pair-Pair real-sqrt-le-mono add-mono power-mono*)
thus *?thesis* **unfolding** *bounded-any-center* [**where** *a*=(*x*, *y*)] **by** *auto*
qed

lemma *mem-Times-iff*: *x* ∈ *A* × *B* ⟷ *fst x* ∈ *A* ∧ *snd x* ∈ *B*
by (*induct x*) *simp*

lemma *compact-Times*: *compact s* ⟹ *compact t* ⟹ *compact* (*s* × *t*)
unfolding *compact-def*
apply *clarify*
apply (*drule-tac x*=*fst* ∘ *f* **in** *spec*)
apply (*drule mp*, *simp add*: *mem-Times-iff*)

apply (*clarify*, *rename-tac l1 r1*)
apply (*drule-tac x=snd ∘ f ∘ r1* **in** *spec*)
apply (*drule mp*, *simp add*: *mem-Times-iff*)
apply (*clarify*, *rename-tac l2 r2*)
apply (*rule-tac x=(l1*, *l2*) **in** *rev-bexI*, *simp*)
apply (*rule-tac x=r1 ∘ r2* **in** *exI*)
apply (*rule conjI*, *simp add*: *subseq-def*)
apply (*drule-tac r=r2* **in** *lim-subseq* [*COMP swap-prems-rl*], *assumption*)
apply (*drule* (*1*) *tendsto-Pair*) **back**
apply (*simp add*: *o-def*)
done

Hence some useful properties follow quite easily.

lemma *compact-scaling*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *compact s* **shows** *compact* ((λ*x*. *c* ∗$_R$ *x*) ‘ *s*)
proof−
let *?f* = λ*x*. *scaleR c x*
have ∗:*bounded-linear ?f* **by** (*rule scaleR.bounded-linear-right*)
show *?thesis* **using** *compact-continuous-image*[*of s ?f*] *continuous-at-imp-continuous-on*[*of s ?f*]
using *linear-continuous-at*[*OF* ∗] *assms* **by** *auto*
qed

lemma *compact-negations*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *compact s* **shows** *compact* ((λ*x*. −*x*) ‘ *s*)
using *compact-scaling* [*OF assms*, *of* − *1*] **by** *auto*

lemma *compact-sums*:
fixes *s t* :: *'a*::*real-normed-vector set*
assumes *compact s compact t* **shows** *compact* {*x* + *y* | *x y*. *x* ∈ *s* ∧ *y* ∈ *t*}
proof−
have ∗:{*x* + *y* | *x y*. *x* ∈ *s* ∧ *y* ∈ *t*} = (λ*z*. *fst z* + *snd z*) ‘ (*s* × *t*)
apply *auto* **unfolding** *image-iff* **apply**(*rule-tac x=*(*xa*, *y*) **in** *bexI*) **by** *auto*
have *continuous-on* (*s* × *t*) (λ*z*. *fst z* + *snd z*)
unfolding *continuous-on* **by** (*rule ballI*) (*intro tendsto-intros*)
thus *?thesis* **unfolding** ∗ **using** *compact-continuous-image compact-Times* [*OF assms*] **by** *auto*
qed

lemma *compact-differences*:
fixes *s t* :: *'a*::*real-normed-vector set*
assumes *compact s compact t* **shows** *compact* {*x* − *y* | *x y*. *x* ∈ *s* ∧ *y* ∈ *t*}
proof−
have {*x* − *y* | *x y*. *x*∈*s* ∧ *y* ∈ *t*} = {*x* + *y* | *x y*. *x* ∈ *s* ∧ *y* ∈ (*uminus* ‘ *t*)}
apply *auto* **apply**(*rule-tac x*= *xa* **in** *exI*) **apply** *auto* **apply**(*rule-tac x=xa* **in** *exI*) **by** *auto*
thus *?thesis* **using** *compact-sums*[*OF assms*(*1*) *compact-negations*[*OF assms*(*2*)]]

by *auto*
qed

lemma *compact-translation*:
fixes s :: *'a*::*real-normed-vector set*
assumes *compact* s **shows** *compact* $((\lambda x.\ a + x)\ `\ s)$
proof−
have $\{x + y\ |x\ y.\ x \in s \wedge y \in \{a\}\} = (\lambda x.\ a + x)\ `\ s$ **by** *auto*
thus *?thesis* **using** *compact-sums*[*OF assms compact-sing*[*of a*]] **by** *auto*
qed

lemma *compact-affinity*:
fixes s :: *'a*::*real-normed-vector set*
assumes *compact* s **shows** *compact* $((\lambda x.\ a + c *_R x)\ `\ s)$
proof−
have *op* $+$ a ` *op* $*_R$ $c$ ` $s = (\lambda x.\ a + c *_R x)\ `\ s$ **by** *auto*
thus *?thesis* **using** *compact-translation*[*OF compact-scaling*[*OF assms*], *of a c*]
by *auto*
qed

Hence we get the following.

lemma *compact-sup-maxdistance*:
fixes s :: *'a*::*real-normed-vector set*
assumes *compact* s $s \neq \{\}$
shows $\exists x{\in}s.\ \exists y{\in}s.\ \forall u{\in}s.\ \forall v{\in}s.\ norm(u - v) \leq norm(x - y)$
proof−
have $\{x - y \mid x\ y\ .\ x{\in}s \wedge y{\in}s\} \neq \{\}$ **using** ‹$s \neq \{\}$› **by** *auto*
then obtain x **where** $x{:}x{\in}\{x - y\ |x\ y.\ x \in s \wedge y \in s\}\ \forall y{\in}\{x - y\ |x\ y.\ x \in s \wedge y \in s\}.\ norm\ y \leq norm\ x$
using *compact-differences*[*OF assms*(*1*) *assms*(*1*)]
using *distance-attains-sup*[**where** *'a*=*'a*, *unfolded dist-norm*, *of* $\{x - y \mid x\ y\ .\ x{\in}s \wedge y{\in}s\}$ *0*] **by** *auto*
from $x(1)$ **obtain** $a\ b$ **where** $a{\in}s\ b{\in}s\ x = a - b$ **by** *auto*
thus *?thesis* **using** $x(2)$[*unfolded* ‹$x = a - b$›] **by** *blast*
qed

We can state this in terms of diameter of a set.

definition *diameter* s = (*if* $s = \{\}$ *then* *0*::*real else Sup* $\{norm(x - y) \mid x\ y.\ x \in s \wedge y \in s\}$)

lemma *diameter-bounded*:
assumes *bounded* s
shows $\forall x{\in}s.\ \forall y{\in}s.\ norm(x - y) \leq diameter\ s$
$\forall d{>}0.\ d < diameter\ s \longrightarrow (\exists x{\in}s.\ \exists y{\in}s.\ norm(x - y) > d)$
proof−
let *?D* = $\{norm\ (x - y)\ |x\ y.\ x \in s \wedge y \in s\}$
obtain a **where** $a{:}\forall x{\in}s.\ norm\ x \leq a$ **using** *assms*[*unfolded bounded-iff*] **by** *auto*

```
  { fix x y assume x ∈ s y ∈ s
    hence norm (x − y) ≤ 2 ∗ a using norm-triangle-ineq[of x −y, unfolded
norm-minus-cancel] a[THEN bspec[where x=x]] a[THEN bspec[where x=y]] by
(auto simp add: field-simps) }
  note ∗ = this
  { fix x y assume x∈s y∈s hence s ≠ {} by auto
    have norm(x − y) ≤ diameter s unfolding diameter-def using ‹s≠{}› ∗[OF
‹x∈s› ‹y∈s›] ‹x∈s› ‹y∈s›
      by simp (blast intro!: Sup-upper ∗) }
  moreover
  { fix d::real assume d>0 d < diameter s
    hence s≠{} unfolding diameter-def by auto
    have ∃ d′ ∈ ?D. d′ > d
    proof(rule ccontr)
      assume ¬ (∃ d′∈{norm (x − y) |x y. x ∈ s ∧ y ∈ s}. d < d′)
      hence ∀ d′∈?D. d′ ≤ d by auto (metis not-leE)
      thus False using ‹d < diameter s› ‹s≠{}›
        apply (auto simp add: diameter-def)
        apply (drule Sup-real-iff [THEN [2] rev-iffD2])
        apply (auto, force)
        done
    qed
    hence ∃ x∈s. ∃ y∈s. norm(x − y) > d by auto }
  ultimately show ∀ x∈s. ∀ y∈s. norm(x − y) ≤ diameter s
      ∀ d>0. d < diameter s −−> (∃ x∈s. ∃ y∈s. norm(x − y) > d) by auto
qed

lemma diameter-bounded-bound:
 bounded s ⟹ x ∈ s ⟹ y ∈ s ==> norm(x − y) ≤ diameter s
  using diameter-bounded by blast

lemma diameter-compact-attained:
  fixes s :: 'a::real-normed-vector set
  assumes compact s  s ≠ {}
  shows ∃ x∈s. ∃ y∈s. (norm(x − y) = diameter s)
proof−
  have b:bounded s using assms(1) by (rule compact-imp-bounded)
  then obtain x y where xys:x∈s y∈s and xy:∀ u∈s. ∀ v∈s. norm (u − v) ≤
norm (x − y) using compact-sup-maxdistance[OF assms] by auto
  hence diameter s ≤ norm (x − y)
    unfolding diameter-def by clarsimp (rule Sup-least, fast+)
  thus ?thesis
    by (metis b diameter-bounded-bound order-antisym xys)
qed
```

Related results with closure as the conclusion.

```
lemma closed-scaling:
  fixes s :: 'a::real-normed-vector set
  assumes closed s shows closed ((λx. c ∗R x) ‘ s)
```

proof(*cases s={}*)
case *True* **thus** *?thesis* **by** *auto*
next
case *False*
show *?thesis*
proof(*cases c=0*)
have *∗:(λx. 0) ‘ s = {0}* **using** ‹*s≠{}*› **by** *auto*
case *True* **thus** *?thesis* **apply** *auto* **unfolding** *∗* **by** *auto*
next
case *False*
{ fix *x l* **assume** *as:∀ n::nat. x n ∈ scaleR c ‘ s (x −−−> l) sequentially*
{ fix *n::nat* **have** *scaleR (1 / c) (x n) ∈ s*
using *as(1)[THEN spec*[**where** *x=n*]]
using ‹*c≠0*› **by** (*auto simp add*: *vector-smult-assoc*)
}
moreover
{ fix *e::real* **assume** *e>0*
hence *0 < e ∗|c|* **using** ‹*c≠0*› *mult-pos-pos*[*of e abs c*] **by** *auto*
then obtain *N* **where** *∀ n≥N. dist (x n) l < e ∗ |c|*
using *as(2)*[*unfolded Lim-sequentially*, *THEN spec*[**where** *x=e ∗ abs c*]]
by *auto*
hence *∃ N. ∀ n≥N. dist (scaleR (1 / c) (x n)) (scaleR (1 / c) l) < e*
unfolding *dist-norm* **unfolding** *scaleR-right-diff-distrib*[*THEN sym*]
using *mult-imp-div-pos-less*[*of abs c - e*] ‹*c≠0*› **by** *auto* **}**
hence *((λn. scaleR (1 / c) (x n)) −−−> scaleR (1 / c) l) sequentially*
unfolding *Lim-sequentially* **by** *auto*
ultimately have *l ∈ scaleR c ‘ s*
using *assms*[*unfolded closed-sequential-limits*, *THEN spec*[**where** *x=λn. scaleR (1/c) (x n)*], *THEN spec*[**where** *x=scaleR (1/c) l*]]
unfolding *image-iff* **using** ‹*c≠0*› **apply**(*rule-tac x=scaleR (1 / c) l* **in** *bexI*) **by** *auto* **}**
thus *?thesis* **unfolding** *closed-sequential-limits* **by** *fast*
qed
qed

lemma *closed-negations*:
fixes *s :: 'a::real-normed-vector set*
assumes *closed s* **shows** *closed ((λx. −x) ‘ s)*
using *closed-scaling*[*OF assms*, *of − 1*] **by** *simp*

lemma *compact-closed-sums*:
fixes *s :: 'a::real-normed-vector set*
assumes *compact s closed t* **shows** *closed {x + y | x y. x ∈ s ∧ y ∈ t}*
proof−
let *?S = {x + y |x y. x ∈ s ∧ y ∈ t}*
{ fix *x l* **assume** *as:∀ n. x n ∈ ?S (x −−−> l) sequentially*
from *as(1)* **obtain** *f* **where** *f:∀ n. x n = fst (f n) + snd (f n) ∀ n. fst (f n) ∈ s ∀ n. snd (f n) ∈ t*
using *choice*[*of λn y. x n = (fst y) + (snd y) ∧ fst y ∈ s ∧ snd y ∈ t*] **by**

auto
obtain l' r **where** $l' \in s$ **and** r:*subseq* r **and** lr:$(((\lambda n.\ fst\ (f\ n)) \circ r) \longrightarrow l')$ *sequentially*
using *assms*(*1*)[*unfolded compact-def*, *THEN spec*[**where** $x=\lambda\ n.\ fst\ (f\ n)$]]
using $f(2)$ **by** *auto*
have $((\lambda n.\ snd\ (f\ (r\ n))) \longrightarrow l - l')$ *sequentially*
using *Lim-sub*[*OF lim-subseq*[*OF r as*(*2*)] *lr*] **and** $f(1)$ **unfolding** *o-def* **by** *auto*
hence $l - l' \in t$
using *assms*(*2*)[*unfolded closed-sequential-limits*, *THEN spec*[**where** $x=\lambda\ n.\ snd\ (f\ (r\ n))$], *THEN spec*[**where** $x=l - l'$]]
using $f(3)$ **by** *auto*
hence $l \in ?S$ **using** $\langle l' \in s \rangle$ **apply** *auto* **apply**(*rule-tac* $x=l'$ **in** *exI*) **apply**(*rule-tac* $x=l - l'$ **in** *exI*) **by** *auto*
}
thus *?thesis* **unfolding** *closed-sequential-limits* **by** *fast*
qed

lemma *closed-compact-sums*:
fixes $s\ t$ :: *'a*::*real-normed-vector set*
assumes *closed* s *compact* t
shows *closed* $\{x + y \mid x\ y.\ x \in s \wedge y \in t\}$
proof−
have $\{x + y \mid x\ y.\ x \in t \wedge y \in s\} = \{x + y \mid x\ y.\ x \in s \wedge y \in t\}$ **apply** *auto*
apply(*rule-tac* $x=y$ **in** *exI*) **apply** *auto* **apply**(*rule-tac* $x=y$ **in** *exI*) **by** *auto*
thus *?thesis* **using** *compact-closed-sums*[*OF assms*(*2*,*1*)] **by** *simp*
qed

lemma *compact-closed-differences*:
fixes $s\ t$ :: *'a*::*real-normed-vector set*
assumes *compact* s *closed* t
shows *closed* $\{x - y \mid x\ y.\ x \in s \wedge y \in t\}$
proof−
have $\{x + y \mid x\ y.\ x \in s \wedge y \in uminus\ `\ t\} = \{x - y \mid x\ y.\ x \in s \wedge y \in t\}$
apply *auto* **apply**(*rule-tac* $x=xa$ **in** *exI*) **apply** *auto* **apply**(*rule-tac* $x=xa$ **in** *exI*) **by** *auto*
thus *?thesis* **using** *compact-closed-sums*[*OF assms*(*1*) *closed-negations*[*OF assms*(*2*)]] **by** *auto*
qed

lemma *closed-compact-differences*:
fixes $s\ t$ :: *'a*::*real-normed-vector set*
assumes *closed* s *compact* t
shows *closed* $\{x - y \mid x\ y.\ x \in s \wedge y \in t\}$
proof−
have $\{x + y \mid x\ y.\ x \in s \wedge y \in uminus\ `\ t\} = \{x - y \mid x\ y.\ x \in s \wedge y \in t\}$
apply *auto* **apply**(*rule-tac* $x=xa$ **in** *exI*) **apply** *auto* **apply**(*rule-tac* $x=xa$ **in** *exI*) **by** *auto*
thus *?thesis* **using** *closed-compact-sums*[*OF assms*(*1*) *compact-negations*[*OF assms*(*2*)]]

by *simp*
qed

lemma *closed-translation*:
fixes *a* :: *'a*::*real-normed-vector*
assumes *closed s* **shows** *closed* ((λ*x*. *a* + *x*) ‘ *s*)
proof−
have {*a* + *y* |*y*. *y* ∈ *s*} = (*op* + *a* ‘ *s*) **by** *auto*
thus *?thesis* **using** *compact-closed-sums*[*OF compact-sing*[*of a*] *assms*] **by** *auto*
qed

lemma *translation-Compl*:
fixes *a* :: *'a*::*ab-group-add*
shows (λ*x*. *a* + *x*) ‘ (− *t*) = − ((λ*x*. *a* + *x*) ‘ *t*)
apply (*auto simp add*: *image-iff*) **apply**(*rule-tac x*=*x* − *a* **in** *bexI*) **by** *auto*

lemma *translation-UNIV*:
fixes *a* :: *'a*::*ab-group-add* **shows** *range* (λ*x*. *a* + *x*) = *UNIV*
apply (*auto simp add*: *image-iff*) **apply**(*rule-tac x*=*x* − *a* **in** *exI*) **by** *auto*

lemma *translation-diff*:
fixes *a* :: *'a*::*ab-group-add*
shows (λ*x*. *a* + *x*) ‘ (*s* − *t*) = ((λ*x*. *a* + *x*) ‘ *s*) − ((λ*x*. *a* + *x*) ‘ *t*)
by *auto*

lemma *closure-translation*:
fixes *a* :: *'a*::*real-normed-vector*
shows *closure* ((λ*x*. *a* + *x*) ‘ *s*) = (λ*x*. *a* + *x*) ‘ (*closure s*)
proof−
have ∗:*op* + *a* ‘ (− *s*) = − *op* + *a* ‘ *s*
apply *auto* **unfolding** *image-iff* **apply**(*rule-tac x*=*x* − *a* **in** *bexI*) **by** *auto*
show *?thesis* **unfolding** *closure-interior translation-Compl*
using *interior-translation*[*of a* − *s*] **unfolding** ∗ **by** *auto*
qed

lemma *frontier-translation*:
fixes *a* :: *'a*::*real-normed-vector*
shows *frontier*((λ*x*. *a* + *x*) ‘ *s*) = (λ*x*. *a* + *x*) ‘ (*frontier s*)
unfolding *frontier-def translation-diff interior-translation closure-translation* **by** *auto*

16.23 Separation between points and sets.

lemma *separate-point-closed*:
fixes *s* :: *'a*::*heine-borel set*
shows *closed s* ⟹ *a* ∉ *s* ==> (∃ *d*>*0*. ∀ *x*∈*s*. *d* ≤ *dist a x*)
proof(*cases s* = {})
case *True*
thus *?thesis* **by**(*auto intro*!: *exI*[**where** *x*=*1*])

next
case *False*
assume *closed s a* $\notin$ *s*
then obtain *x* **where** *x*$\in$*s* $\forall$ *y*$\in$*s*. *dist a x* $\leq$ *dist a y* **using** ‹*s* $\neq$ {}› *distance-attains-inf* [*of s a*] **by** *blast*
with ‹*x*$\in$*s*› **show** *?thesis* **using** *dist-pos-lt*[*of a x*] **and**‹*a* $\notin$ *s*› **by** *blast*
qed

lemma *separate-compact-closed*:
fixes *s t* :: *'a*::{*heine-borel*, *real-normed-vector*} *set*

assumes *compact s* **and** *closed t* **and** *s* $\cap$ *t* = {}
shows $\exists$ *d*>*0*. $\forall$ *x*$\in$*s*. $\forall$ *y*$\in$*t*. *d* $\leq$ *dist x y*
proof−
have *0* $\notin$ {*x* − *y* |*x y*. *x* $\in$ *s* $\wedge$ *y* $\in$ *t*} **using** *assms*(*3*) **by** *auto*
then obtain *d* **where** *d*>*0* **and** *d*:$\forall$ *x*$\in${*x* − *y* |*x y*. *x* $\in$ *s* $\wedge$ *y* $\in$ *t*}. *d* $\leq$ *dist 0 x*
using *separate-point-closed*[*OF compact-closed-differences*[*OF assms*(*1*,*2*)], *of 0*] **by** *auto*
{ fix *x y* **assume** *x*$\in$*s y*$\in$*t*
hence *x* − *y* $\in$ {*x* − *y* |*x y*. *x* $\in$ *s* $\wedge$ *y* $\in$ *t*} **by** *auto*
hence *d* $\leq$ *dist* (*x* − *y*) *0* **using** *d*[*THEN bspec*[**where** *x*=*x* − *y*]] **using** *dist-commute*
by (*auto simp add*: *dist-commute*)
hence *d* $\leq$ *dist x y* **unfolding** *dist-norm* **by** *auto* **}**
thus *?thesis* **using** ‹*d*>*0*› **by** *auto*
qed

lemma *separate-closed-compact*:
fixes *s t* :: *'a*::{*heine-borel*, *real-normed-vector*} *set*
assumes *closed s* **and** *compact t* **and** *s* $\cap$ *t* = {}
shows $\exists$ *d*>*0*. $\forall$ *x*$\in$*s*. $\forall$ *y*$\in$*t*. *d* $\leq$ *dist x y*
proof−
have ∗:*t* $\cap$ *s* = {} **using** *assms*(*3*) **by** *auto*
show *?thesis* **using** *separate-compact-closed*[*OF assms*(*2*,*1*) ∗]
apply *auto* **apply**(*rule-tac x*=*d* **in** *exI*) **apply** *auto* **apply** (*erule-tac x*=*y* **in** *ballE*)
by (*auto simp add*: *dist-commute*)
qed

16.24 Intervals

lemma *interval*: **fixes** *a* :: *'a*::*ord*^*'n* **shows**
{*a* <..< *b*} = {*x*::*'a*^*'n*. $\forall$ *i*. *a*$*i* < *x*$*i* $\wedge$ *x*$*i* < *b*$*i*} **and**
{*a* .. *b*} = {*x*::*'a*^*'n*. $\forall$ *i*. *a*$*i* $\leq$ *x*$*i* $\wedge$ *x*$*i* $\leq$ *b*$*i*}
by (*auto simp add*: *expand-set-eq vector-less-def vector-le-def*)

lemma *mem-interval*: **fixes** *a* :: *'a*::*ord*^*'n* **shows**
x $\in$ {*a*<..<*b*} $\longleftrightarrow$ ($\forall$ *i*. *a*$*i* < *x*$*i* $\wedge$ *x*$*i* < *b*$*i*)

$x \in \{a\ ..\ b\} \longleftrightarrow (\forall\, i.\ a\$i \leq x\$i \wedge x\$i \leq b\$i)$
using *interval*[*of a b*] **by**(*auto simp add*: *expand-set-eq vector-less-def vector-le-def*)

lemma *interval-eq-empty*: **fixes** $a :: real\hat{}'n$ **shows**
$(\{a <..< b\} = \{\} \longleftrightarrow (\exists\, i.\ b\$i \leq a\$i))$ (**is** *?th1*) **and**
$(\{a\ ..\ b\} = \{\} \longleftrightarrow (\exists\, i.\ b\$i < a\$i))$ (**is** *?th2*)
proof−
{ fix *i x* **assume** *as*:$b\$i \leq a\$i$ **and** *x*:$x \in \{a <..< b\}$
hence $a\ \$\ i < x\ \$\ i \wedge x\ \$\ i < b\ \$\ i$ **unfolding** *mem-interval* **by** *auto*
hence $a\$i < b\$i$ **by** *auto*
hence *False* **using** *as* **by** *auto* **}**
moreover
{ assume *as*:$\forall\, i.\ \neg\ (b\$i \leq a\$i)$
let *?x* $= (1/2) *_R (a + b)$
{ fix *i*
have $a\$i < b\$i$ **using** *as*[*THEN spec*[**where** *x*=*i*]] **by** *auto*
hence $a\$i < ((1/2) *_R (a+b))\ \$\ i\ ((1/2) *_R (a+b))\ \$\ i < b\$i$
unfolding *vector-smult-component* **and** *vector-add-component*
by *auto* **}**
hence $\{a <..< b\} \neq \{\}$ **using** *mem-interval*(*1*)[*of ?x a b*] **by** *auto* **}**
ultimately show *?th1* **by** *blast*

{ fix *i x* **assume** *as*:$b\$i < a\$i$ **and** *x*:$x \in \{a\ ..\ b\}$
hence $a\ \$\ i \leq x\ \$\ i \wedge x\ \$\ i \leq b\ \$\ i$ **unfolding** *mem-interval* **by** *auto*
hence $a\$i \leq b\$i$ **by** *auto*
hence *False* **using** *as* **by** *auto* **}**
moreover
{ assume *as*:$\forall\, i.\ \neg\ (b\$i < a\$i)$
let *?x* $= (1/2) *_R (a + b)$
{ fix *i*
have $a\$i \leq b\$i$ **using** *as*[*THEN spec*[**where** *x*=*i*]] **by** *auto*
hence $a\$i \leq ((1/2) *_R (a+b))\ \$\ i\ ((1/2) *_R (a+b))\ \$\ i \leq b\$i$
unfolding *vector-smult-component* **and** *vector-add-component*
by *auto* **}**
hence $\{a\ ..\ b\} \neq \{\}$ **using** *mem-interval*(*2*)[*of ?x a b*] **by** *auto* **}**
ultimately show *?th2* **by** *blast*
qed

lemma *interval-ne-empty*: **fixes** $a :: real\hat{}'n$ **shows**
$\{a\ ..\ b\} \neq \{\} \longleftrightarrow (\forall\, i.\ a\$i \leq b\$i)$ **and**
$\{a <..< b\} \neq \{\} \longleftrightarrow (\forall\, i.\ a\$i < b\$i)$
unfolding *interval-eq-empty*[*of a b*] **by** (*auto simp add*: *not-less not-le*)

lemma *subset-interval-imp*: **fixes** $a :: real\hat{}'n$ **shows**
$(\forall\, i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i) \Longrightarrow \{c\ ..\ d\} \subseteq \{a\ ..\ b\}$ **and**
$(\forall\, i.\ a\$i < c\$i \wedge d\$i < b\$i) \Longrightarrow \{c\ ..\ d\} \subseteq \{a<..<b\}$ **and**
$(\forall\, i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i) \Longrightarrow \{c<..<d\} \subseteq \{a\ ..\ b\}$ **and**
$(\forall\, i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i) \Longrightarrow \{c<..<d\} \subseteq \{a<..<b\}$
unfolding *subset-eq*[*unfolded Ball-def*] **unfolding** *mem-interval*

by (*auto intro*: *order-trans less-le-trans le-less-trans less-imp-le*)

lemma *interval-sing*: **fixes** a :: *'a::linorder^'n* **shows**
$\{a\ ..\ a\} = \{a\} \wedge \{a<..<a\} = \{\}$
apply(*auto simp add*: *expand-set-eq vector-less-def vector-le-def Cart-eq*)
apply (*simp add*: *order-eq-iff*)
apply (*auto simp add*: *not-less less-imp-le*)
done

lemma *interval-open-subset-closed*: **fixes** a :: *'a::preorder^'n* **shows**
$\{a<..<b\} \subseteq \{a\ ..\ b\}$
proof(*simp add*: *subset-eq*, *rule*)
fix x
assume x:$x \in \{a<..<b\}$
{ fix i
have $a\ \$\ i \leq x\ \$\ i$
using x *order-less-imp-le*[*of* $a\$i$ $x\$i$]
by(*simp add*: *expand-set-eq vector-less-def vector-le-def Cart-eq*)
}
moreover
{ fix i
have $x\ \$\ i \leq b\ \$\ i$
using x *order-less-imp-le*[*of* $x\$i$ $b\$i$]
by(*simp add*: *expand-set-eq vector-less-def vector-le-def Cart-eq*)
}
ultimately
show $a \leq x \wedge x \leq b$
by(*simp add*: *expand-set-eq vector-less-def vector-le-def Cart-eq*)
qed

lemma *subset-interval*: **fixes** a :: *real^'n* **shows**
$\{c\ ..\ d\} \subseteq \{a\ ..\ b\} \longleftrightarrow (\forall i.\ c\$i \leq d\$i) \longrightarrow (\forall i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i)$ (**is** *?th1*) **and**
$\{c\ ..\ d\} \subseteq \{a<..<b\} \longleftrightarrow (\forall i.\ c\$i \leq d\$i) \longrightarrow (\forall i.\ a\$i < c\$i \wedge d\$i < b\$i)$ (**is** *?th2*) **and**
$\{c<..<d\} \subseteq \{a\ ..\ b\} \longleftrightarrow (\forall i.\ c\$i < d\$i) \longrightarrow (\forall i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i)$ (**is** *?th3*) **and**
$\{c<..<d\} \subseteq \{a<..<b\} \longleftrightarrow (\forall i.\ c\$i < d\$i) \longrightarrow (\forall i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i)$ (**is** *?th4*)
proof−
show *?th1* **unfolding** *subset-eq* **and** *Ball-def* **and** *mem-interval* **by** (*auto intro*: *order-trans*)
show *?th2* **unfolding** *subset-eq* **and** *Ball-def* **and** *mem-interval* **by** (*auto intro*: *le-less-trans less-le-trans order-trans less-imp-le*)
{ assume *as*: $\{c<..<d\} \subseteq \{a\ ..\ b\}\ \forall i.\ c\$i < d\$i$
hence $\{c<..<d\} \neq \{\}$ **unfolding** *interval-eq-empty* **by** (*auto*, *drule-tac* x=i **in** *spec*, *simp*)
fix i

{ let *?x* = (*χ j*. (*if j=i then* ((*min* (*a$j*) (*d$j*))+*c$j*)/*2 else* (*c$j*+*d$j*)/*2*))::*real^'n*
assume *as2*: *a$i* > *c$i*
{ fix *j*
have *c $ j* < *?x $ j* ∧ *?x $ j* < *d $ j* **unfolding** *Cart-lambda-beta*
apply(*cases j=i*) **using** *as*(*2*)[*THEN spec*[**where** *x=j*]]
by (*auto simp add*: *as2*) **}**
hence *?x*∈{*c*<..<*d*} **unfolding** *mem-interval* **by** *auto*
moreover
have *?x*∉{*a* .. *b*}
unfolding *mem-interval* **apply** *auto* **apply**(*rule-tac x=i* **in** *exI*)
using *as*(*2*)[*THEN spec*[**where** *x=i*]] **and** *as2*
by *auto*
ultimately have *False* **using** *as* **by** *auto* **}**
hence *a$i* ≤ *c$i* **by**(*rule ccontr*)*auto*
moreover
{ let *?x* = (*χ j*. (*if j=i then* ((*max* (*b$j*) (*c$j*))+*d$j*)/*2 else* (*c$j*+*d$j*)/*2*))::*real^'n*
assume *as2*: *b$i* < *d$i*
{ fix *j*
have *d $ j* > *?x $ j* ∧ *?x $ j* > *c $ j* **unfolding** *Cart-lambda-beta*
apply(*cases j=i*) **using** *as*(*2*)[*THEN spec*[**where** *x=j*]]
by (*auto simp add*: *as2*) **}**
hence *?x*∈{*c*<..<*d*} **unfolding** *mem-interval* **by** *auto*
moreover
have *?x*∉{*a* .. *b*}
unfolding *mem-interval* **apply** *auto* **apply**(*rule-tac x=i* **in** *exI*)
using *as*(*2*)[*THEN spec*[**where** *x=i*]] **and** *as2*
by *auto*
ultimately have *False* **using** *as* **by** *auto* **}**
hence *b$i* ≥ *d$i* **by**(*rule ccontr*)*auto*
ultimately
have *a$i* ≤ *c$i* ∧ *d$i* ≤ *b$i* **by** *auto*
} note *part1* = *this*
thus *?th3* **unfolding** *subset-eq* **and** *Ball-def* **and** *mem-interval* **apply** *auto* **apply** (*erule-tac x=ia* **in** *allE*, *simp*)+ **by** (*erule-tac x=i* **in** *allE*, *erule-tac x=i* **in** *allE*, *simp*)+
{ assume *as*:{*c*<..<*d*} ⊆ {*a*<..<*b*} ∀ *i*. *c$i* < *d$i*
fix *i*
from *as*(*1*) **have** {*c*<..<*d*} ⊆ {*a*..*b*} **using** *interval-open-subset-closed*[*of a b*] **by** *auto*
hence *a$i* ≤ *c$i* ∧ *d$i* ≤ *b$i* **using** *part1* **and** *as*(*2*) **by** *auto* **} note** ∗ = *this*
thus *?th4* **unfolding** *subset-eq* **and** *Ball-def* **and** *mem-interval* **apply** *auto* **apply** (*erule-tac x=ia* **in** *allE*, *simp*)+ **by** (*erule-tac x=i* **in** *allE*, *erule-tac x=i* **in** *allE*, *simp*)+
qed

lemma *disjoint-interval*: **fixes** *a*::*real^'n* **shows**
{*a* .. *b*} ∩ {*c* .. *d*} = {} ⟷ (∃ *i*. (*b$i* < *a$i* ∨ *d$i* < *c$i* ∨ *b$i* < *c$i* ∨ *d$i* < *a$i*)) (**is** *?th1*) **and**

$\{a\ ..\ b\} \cap \{c<..<d\} = \{\} \longleftrightarrow (\exists\, i.\ (b\$i < a\$i \vee d\$i \leq c\$i \vee b\$i \leq c\$i \vee d\$i \leq a\$i))$ (**is** *?th2*) **and**
$\{a<..<b\} \cap \{c\ ..\ d\} = \{\} \longleftrightarrow (\exists\, i.\ (b\$i \leq a\$i \vee d\$i < c\$i \vee b\$i \leq c\$i \vee d\$i \leq a\$i))$ (**is** *?th3*) **and**
$\{a<..<b\} \cap \{c<..<d\} = \{\} \longleftrightarrow (\exists\, i.\ (b\$i \leq a\$i \vee d\$i \leq c\$i \vee b\$i \leq c\$i \vee d\$i \leq a\$i))$ (**is** *?th4*)
proof−
let *?z* = $(\chi\ i.\ ((max\ (a\$i)\ (c\$i)) + (min\ (b\$i)\ (d\$i)))\ /\ 2)$::*real^'n*
show *?th1 ?th2 ?th3 ?th4*
unfolding *expand-set-eq* **and** *Int-iff* **and** *empty-iff* **and** *mem-interval* **and** *all-conj-distrib*[*THEN sym*] **and** *eq-False*
apply (*auto elim*!: *allE*[**where** *x*=*?z*])
apply ((*rule-tac x*=*x* **in** *exI*, *force*) | (*rule-tac x*=*i* **in** *exI*, *force*))+
done
qed

lemma *inter-interval*: **fixes** *a* :: *'a*::*linorder^'n* **shows**
$\{a\ ..\ b\} \cap \{c\ ..\ d\} = \{(\chi\ i.\ max\ (a\$i)\ (c\$i))\ ..\ (\chi\ i.\ min\ (b\$i)\ (d\$i))\}$
unfolding *expand-set-eq* **and** *Int-iff* **and** *mem-interval*
by *auto*

lemma *open-interval-lemma*: **fixes** *x* :: *real* **shows**
$a < x \Longrightarrow x < b ==> (\exists\, d{>}0.\ \forall\, x'.\ abs(x' - x) < d \text{ --> } a < x' \wedge x' < b)$
by(*rule-tac x*=*min* $(x - a)$ $(b - x)$ **in** *exI*, *auto*)

lemma *open-interval*[*intro*]: **fixes** *a* :: *real^'n* **shows** *open* $\{a<..<b\}$
proof−
{ fix *x* **assume** *x*:$x \in \{a<..<b\}$
{ fix *i*
have $\exists\, d{>}0.\ \forall\, x'.\ abs\ (x' - (x\$i)) < d \longrightarrow a\$i < x' \wedge x' < b\$i$
using *x*[*unfolded mem-interval*, *THEN spec*[**where** *x*=*i*]]
using *open-interval-lemma*[*of a*$*i x*$*i b*$*i*] **by** *auto* **}**

hence $\forall\, i.\ \exists\, d{>}0.\ \forall\, x'.\ abs\ (x' - (x\$i)) < d \longrightarrow a\$i < x' \wedge x' < b\$i$ **by** *auto*
then obtain *d* **where** *d*:$\forall\, i.\ 0 < d\ i \wedge (\forall\, x'.\ |x' - x\ \$\ i| < d\ i \longrightarrow a\ \$\ i < x' \wedge x' < b\ \$\ i)$
using *bchoice*[*of UNIV* $\lambda i\ d.\ d{>}0 \wedge (\forall\, x'.\ |x' - x\ \$\ i| < d \longrightarrow a\ \$\ i < x' \wedge x' < b\ \$\ i)$] **by** *auto*

let *?d* = *Min* (*range d*)
have **:*finite* (*range d*) *range d* $\neq \{\}$ **by** *auto*
have *?d*>*0* **unfolding** *Min-gr-iff*[*OF* **] **using** *d* **by** *auto*
moreover
{ fix *x'* **assume** *as*:*dist x' x* < *?d*
{ fix *i*
have $|x'\$i - x\ \$\ i| < d\ i$
using *norm-bound-component-lt*[*OF as*[*unfolded dist-norm*], *of i*]

unfolding *vector-minus-component* **and** *Min-gr-iff*[*OF* ∗∗] **by** *auto*
hence $a \$ i < x' \$ i$ $x' \$ i < b \$ i$ **using** *d*[*THEN spec*[**where** *x*=*i*]] **by** *auto* **}**
hence $a < x' \wedge x' < b$ **unfolding** *vector-less-def* **by** *auto* **}**
ultimately have $\exists e{>}0.\ \forall x'.\ dist\ x'\ x < e \longrightarrow x' \in \{a{<..<}b\}$ **by** (*auto*, *rule-tac x*=*?d* **in** *exI*, *simp*)
}
thus *?thesis* **unfolding** *open-dist* **using** *open-interval-lemma* **by** *auto*
qed

lemma *open-interval-real*[*intro*]: **fixes** *a* :: *real* **shows** *open* $\{a{<..<}b\}$
by (*rule open-real-greaterThanLessThan*)

lemma *closed-interval*[*intro*]: **fixes** *a* :: *real^'n* **shows** *closed* $\{a\ ..\ b\}$
proof−
{ fix *x i* **assume** *as*:$\forall\ e{>}0.\ \exists x'{\in}\{a..b\}.\ x' \neq x \wedge dist\ x'\ x < e$
{ assume *xa*:$a\$i > x\$i$
with *as* **obtain** *y* **where** *y*:$y{\in}\{a..b\}$ $y \neq x$ *dist y x* $< a\$i - x\$i$ **by**(*erule-tac x*=$a\$i - x\$i$ **in** *allE*)*auto*
hence *False* **unfolding** *mem-interval* **and** *dist-norm*
using *component-le-norm*[*of y*−*x i*, *unfolded vector-minus-component*] **and** *xa* **by**(*auto elim*!: *allE*[**where** *x*=*i*])
} **hence** $a\$i \leq x\$i$ **by**(*rule ccontr*)*auto*
moreover
{ assume *xb*:$b\$i < x\$i$
with *as* **obtain** *y* **where** *y*:$y{\in}\{a..b\}$ $y \neq x$ *dist y x* $< x\$i - b\$i$ **by**(*erule-tac x*=$x\$i - b\$i$ **in** *allE*)*auto*
hence *False* **unfolding** *mem-interval* **and** *dist-norm*
using *component-le-norm*[*of y*−*x i*, *unfolded vector-minus-component*] **and** *xb* **by**(*auto elim*!: *allE*[**where** *x*=*i*])
} **hence** $x\$i \leq b\$i$ **by**(*rule ccontr*)*auto*
ultimately
have $a \$ i \leq x \$ i \wedge x \$ i \leq b \$ i$ **by** *auto* **}**
thus *?thesis* **unfolding** *closed-limpt islimpt-approachable mem-interval* **by** *auto*
qed

lemma *interior-closed-interval*[*intro*]: **fixes** *a* :: *real^'n* **shows**
interior $\{a\ ..\ b\} = \{a{<..<}b\}$ (**is** *?L* = *?R*)
proof(*rule subset-antisym*)
show *?R* ⊆ *?L* **using** *interior-maximal*[*OF interval-open-subset-closed open-interval*]
by *auto*
next
{ fix *x* **assume** $\exists T.\ open\ T \wedge x \in T \wedge T \subseteq \{a..b\}$
then obtain *s* **where** *s*:*open s x* ∈ *s s* ⊆ $\{a..b\}$ **by** *auto*
then obtain *e* **where** *e*>*0* **and** *e*:$\forall x'.\ dist\ x'\ x < e \longrightarrow x' \in \{a..b\}$ **unfolding** *open-dist* **and** *subset-eq* **by** *auto*
{ fix *i*
have *dist* $(x - (e\ /\ 2) *_R basis\ i)\ x < e$
dist $(x + (e\ /\ 2) *_R basis\ i)\ x < e$

unfolding *dist-norm* **apply** *auto*
unfolding *norm-minus-cancel* **using** *norm-basis*[*of i*] **and** ‹*e*>*0*› **by** *auto*
hence *a* $ *i* ≤ (*x* − (*e* / *2*) ∗$_R$ *basis i*) $ *i*
(*x* + (*e* / *2*) ∗$_R$ *basis i*) $ *i* ≤ *b* $ *i*
using *e*[*THEN spec*[**where** *x*=*x* − (*e*/*2*) ∗$_R$ *basis i*]]
and *e*[*THEN spec*[**where** *x*=*x* + (*e*/*2*) ∗$_R$ *basis i*]]
unfolding *mem-interval* **by** (*auto elim*!: *allE*[**where** *x*=*i*])
hence *a* $ *i* < *x* $ *i* **and** *x* $ *i* < *b* $ *i*
unfolding *vector-minus-component* **and** *vector-add-component*
unfolding *vector-smult-component* **and** *basis-component* **using** ‹*e*>*0*› **by** *auto* **}**
hence *x* ∈ {*a*<..<*b*} **unfolding** *mem-interval* **by** *auto* **}**
thus *?L* ⊆ *?R* **unfolding** *interior-def* **and** *subset-eq* **by** *auto*
qed

lemma *bounded-closed-interval*: **fixes** *a* :: *real*^'*n* **shows**
bounded {*a* .. *b*}
proof−
let *?b* = ∑*i*∈*UNIV*. |*a*$*i*| + |*b*$*i*|
{ fix *x*::*real*^'*n* **assume** *x*:∀ *i*. *a* $ *i* ≤ *x* $ *i* ∧ *x* $ *i* ≤ *b* $ *i*
{ fix *i*
have |*x*$*i*| ≤ |*a*$*i*| + |*b*$*i*| **using** *x*[*THEN spec*[**where** *x*=*i*]] **by** *auto* **}**
hence (∑*i*∈*UNIV*. |*x* $ *i*|) ≤ *?b* **by**(*rule setsum-mono*)
hence *norm x* ≤ *?b* **using** *norm-le-l1*[*of x*] **by** *auto* **}**
thus *?thesis* **unfolding** *interval* **and** *bounded-iff* **by** *auto*
qed

lemma *bounded-interval*: **fixes** *a* :: *real*^'*n* **shows**
bounded {*a* .. *b*} ∧ *bounded* {*a*<..<*b*}
using *bounded-closed-interval*[*of a b*]
using *interval-open-subset-closed*[*of a b*]
using *bounded-subset*[*of* {*a*..*b*} {*a*<..<*b*}]
by *simp*

lemma *not-interval-univ*: **fixes** *a* :: *real*^'*n* **shows**
({*a* .. *b*} ≠ *UNIV*) ∧ ({*a*<..<*b*} ≠ *UNIV*)
using *bounded-interval*[*of a b*]
by *auto*

lemma *compact-interval*: **fixes** *a* :: *real*^'*n* **shows**
compact {*a* .. *b*}
using *bounded-closed-imp-compact* **using** *bounded-interval*[*of a b*] **using** *closed-interval*[*of a b*] **by** *auto*

lemma *open-interval-midpoint*: **fixes** *a* :: *real*^'*n*
assumes {*a*<..<*b*} ≠ {} **shows** ((*1*/*2*) ∗$_R$ (*a* + *b*)) ∈ {*a*<..<*b*}
proof−
{ fix *i*
have *a* $ *i* < ((*1* / *2*) ∗$_R$ (*a* + *b*)) $ *i* ∧ ((*1* / *2*) ∗$_R$ (*a* + *b*)) $ *i* < *b* $ *i*

```
    using assms[unfolded interval-ne-empty, THEN spec[where x=i]]
    unfolding vector-smult-component and vector-add-component
    by auto }
  thus ?thesis unfolding mem-interval by auto
qed

lemma open-closed-interval-convex: fixes x :: real^'n
  assumes x:x ∈ {a<..<b} and y:y ∈ {a .. b} and e:0 < e e ≤ 1
  shows (e *R x + (1 − e) *R y) ∈ {a<..<b}
proof−
  { fix i
    have a $ i = e * a$i + (1 − e) * a$i unfolding left-diff-distrib by simp
    also have ... < e * x $ i + (1 − e) * y $ i apply(rule add-less-le-mono)
     using e unfolding mult-less-cancel-left and mult-le-cancel-left apply simp-all
      using x unfolding mem-interval apply simp
      using y unfolding mem-interval apply simp
      done
    finally have a $ i < (e *R x + (1 − e) *R y) $ i by auto
    moreover {
    have b $ i = e * b$i + (1 − e) * b$i unfolding left-diff-distrib by simp
    also have ... > e * x $ i + (1 − e) * y $ i apply(rule add-less-le-mono)
     using e unfolding mult-less-cancel-left and mult-le-cancel-left apply simp-all
      using x unfolding mem-interval apply simp
      using y unfolding mem-interval apply simp
      done
    finally have (e *R x + (1 − e) *R y) $ i < b $ i by auto
    } ultimately have a $ i < (e *R x + (1 − e) *R y) $ i ∧ (e *R x + (1 −
e) *R y) $ i < b $ i by auto }
  thus ?thesis unfolding mem-interval by auto
qed

lemma closure-open-interval: fixes a :: real^'n
  assumes {a<..<b} ≠ {}
  shows closure {a<..<b} = {a .. b}
proof−
  have ab:a < b using assms[unfolded interval-ne-empty] unfolding vector-less-def
by auto
  let ?c = (1 / 2) *R (a + b)
  { fix x assume as:x ∈ {a .. b}
    def f == λn::nat. x + (inverse (real n + 1)) *R (?c − x)
    { fix n assume fn:f n < b ⟶ a < f n ⟶ f n = x and xc:x ≠ ?c
      have *:0 < inverse (real n + 1) inverse (real n + 1) ≤ 1 unfolding
inverse-le-1-iff by auto
      have (inverse (real n + 1)) *R ((1 / 2) *R (a + b)) + (1 − inverse (real n
+ 1)) *R x =
        x + (inverse (real n + 1)) *R (((1 / 2) *R (a + b)) − x)
        by (auto simp add: algebra-simps)
    hence f n < b and a < f n using open-closed-interval-convex[OF open-interval-midpoint[OF
assms] as *] unfolding f-def by auto
```

hence *False* **using** *fn* **unfolding** *f-def* **using** *xc* **by**(*auto simp add*: *vector-ssub-ldistrib*)
}
moreover
{ assume ¬ (*f* −−−> *x*) *sequentially*
{ fix *e*::*real* **assume** *e*>*0*
hence ∃ *N*::*nat*. *inverse* (*real* (*N* + *1*)) < *e* **using** *real-arch-inv*[*of e*] **apply**
(*auto simp add*: *Suc-pred'*) **apply**(*rule-tac x*=*n* − *1* **in** *exI*) **by** *auto*
then obtain *N*::*nat* **where** *inverse* (*real* (*N* + *1*)) < *e* **by** *auto*
hence ∀ *n*≥*N*. *inverse* (*real n* + *1*) < *e* **by** (*auto*, *metis Suc-le-mono le-SucE*
less-imp-inverse-less nat-le-real-less order-less-trans real-of-nat-Suc real-of-nat-Suc-gt-zero)
hence ∃ *N*::*nat*. ∀ *n*≥*N*. *inverse* (*real n* + *1*) < *e* **by** *auto* **}**
hence ((λ*n*. *inverse* (*real n* + *1*)) −−−> *0*) *sequentially*
unfolding *Lim-sequentially* **by**(*auto simp add*: *dist-norm*)
hence (*f* −−−> *x*) *sequentially* **unfolding** *f-def*
using *Lim-add*[*OF Lim-const*, *of* λ*n*::*nat*. (*inverse* (*real n* + *1*)) ∗*R* ((*1* /
2) ∗*R* (*a* + *b*) − *x*) *0 sequentially x*]
using *Lim-vmul*[*of* λ*n*::*nat*. *inverse* (*real n* + *1*) *0 sequentially* ((*1* / *2*) ∗*R*
(*a* + *b*) − *x*)] **by** *auto* **}**
ultimately have *x* ∈ *closure* {*a*<..<*b*}
using *as* **and** *open-interval-midpoint*[*OF assms*] **unfolding** *closure-def* **un-**
folding *islimpt-sequential* **by**(*cases x*=*?c*)*auto* **}**
thus *?thesis* **using** *closure-minimal*[*OF interval-open-subset-closed closed-interval*,
of a b] **by** *blast*
qed

lemma *bounded-subset-open-interval-symmetric*: **fixes** *s*::(*real*^'*n*) *set*
assumes *bounded s* **shows** ∃ *a*. *s* ⊆ {−*a*<..<*a*}
proof−
obtain *b* **where** *b*>*0* **and** *b*:∀ *x*∈*s*. *norm x* ≤ *b* **using** *assms*[*unfolded bounded-pos*]
by *auto*
def *a* ≡ (χ *i*. *b*+*1*)::*real*^'*n*
{ fix *x* **assume** *x*∈*s*
fix *i*
have (−*a*)$*i* < *x*$*i* **and** *x*$*i* < *a*$*i* **using** *b*[*THEN bspec*[**where** *x*=*x*], *OF*
‹*x*∈*s*›] **and** *component-le-norm*[*of x i*]
unfolding *vector-uminus-component* **and** *a-def* **and** *Cart-lambda-beta* **by**
auto
}
thus *?thesis* **by**(*auto intro*: *exI*[**where** *x*=*a*] *simp add*: *vector-less-def*)
qed

lemma *bounded-subset-open-interval*:
fixes *s* :: (*real* ^ '*n*) *set*
shows *bounded s* ==> (∃ *a b*. *s* ⊆ {*a*<..<*b*})
by (*auto dest*!: *bounded-subset-open-interval-symmetric*)

lemma *bounded-subset-closed-interval-symmetric*:
fixes *s* :: (*real* ^ '*n*) *set*
assumes *bounded s* **shows** ∃ *a*. *s* ⊆ {−*a* .. *a*}

proof−
obtain *a* **where** $s \subseteq \{-a<..<a\}$ **using** *bounded-subset-open-interval-symmetric*[*OF assms*] **by** *auto*
thus *?thesis* **using** *interval-open-subset-closed*[*of* $-a$ *a*] **by** *auto*
qed

lemma *bounded-subset-closed-interval*:
fixes *s* :: (*real* ^ *'n*) *set*
shows *bounded s* ==> $(\exists\, a\ b.\ s \subseteq \{a\ ..\ b\})$
using *bounded-subset-closed-interval-symmetric*[*of s*] **by** *auto*

lemma *frontier-closed-interval*:
fixes *a b* :: *real* ^ -
shows *frontier* $\{a\ ..\ b\} = \{a\ ..\ b\} - \{a<..<b\}$
unfolding *frontier-def* **unfolding** *interior-closed-interval* **and** *closure-closed*[*OF closed-interval*] **..**

lemma *frontier-open-interval*:
fixes *a b* :: *real* ^ -
shows *frontier* $\{a<..<b\} = (\mathit{if}\ \{a<..<b\} = \{\}\ \mathit{then}\ \{\}\ \mathit{else}\ \{a\ ..\ b\} - \{a<..<b\})$
proof(*cases* $\{a<..<b\} = \{\}$)
case *True* **thus** *?thesis* **using** *frontier-empty* **by** *auto*
next
case *False* **thus** *?thesis* **unfolding** *frontier-def* **and** *closure-open-interval*[*OF False*] **and** *interior-open*[*OF open-interval*] **by** *auto*
qed

lemma *inter-interval-mixed-eq-empty*: **fixes** *a* :: *real^'n*
assumes $\{c<..<d\} \neq \{\}$ **shows** $\{a<..<b\} \cap \{c\ ..\ d\} = \{\} \longleftrightarrow \{a<..<b\} \cap \{c<..<d\} = \{\}$
unfolding *closure-open-interval*[*OF assms*, *THEN sym*] **unfolding** *open-inter-closure-eq-empty*[*OF open-interval*] **..**

lemma *closed-interval-left*: **fixes** *b*::*real^'n*
shows *closed* $\{x$::*real^'n*$.\ \forall\, i.\ x\$i \leq b\$i\}$
proof−
{ fix *i*
fix *x*::*real^'n* **assume** *x*:$\forall\, e>0.\ \exists\, x' \in \{x.\ \forall\, i.\ x\ \$\ i \leq b\ \$\ i\}.\ x' \neq x \wedge \mathit{dist}\ x'\ x < e$
{ assume $x\$i > b\$i$
then obtain *y* **where** $y\ \$\ i \leq b\ \$\ i\ \ y \neq x\ \ \mathit{dist}\ y\ x < x\$i - b\$i$ **using** *x*[*THEN spec*[**where** $x{=}x\$i - b\$i$]] **by** *auto*
hence *False* **using** *component-le-norm*[*of* $y - x$ *i*] **unfolding** *dist-norm* **and** *vector-minus-component* **by** *auto* **}**
hence $x\$i \leq b\$i$ **by**(*rule ccontr*)*auto* **}**
thus *?thesis* **unfolding** *closed-limpt* **unfolding** *islimpt-approachable* **by** *blast*

qed

lemma *closed-interval-right*: **fixes** *a*::*real^'n*
shows *closed* {*x*::*real^'n*. ∀ *i*. *a*$*i* ≤ *x*$*i*}
proof−
{ fix *i*
fix *x*::*real^'n* **assume** *x*:∀ *e*>*0*. ∃ *x'*∈{*x*. ∀ *i*. *a* $ *i* ≤ *x* $ *i*}. *x'* ≠ *x* ∧ *dist x' x* < *e*
{ assume *a*$*i* > *x*$*i*
then obtain *y* **where** *a* $ *i* ≤ *y* $ *i* *y* ≠ *x* *dist y x* < *a*$*i* − *x*$*i* **using** *x*[*THEN spec*[**where** *x*=*a*$*i* − *x*$*i*]] **by** *auto*
hence *False* **using** *component-le-norm*[*of y* − *x i*] **unfolding** *dist-norm* **and** *vector-minus-component* **by** *auto* **}**
hence *a*$*i* ≤ *x*$*i* **by**(*rule ccontr*)*auto* **}**
thus *?thesis* **unfolding** *closed-limpt* **unfolding** *islimpt-approachable* **by** *blast*
qed

Intervals in general, including infinite and mixtures of open and closed.

definition *is-interval s* ⟷ (∀ *a*∈*s*. ∀ *b*∈*s*. ∀ *x*. (∀ *i*. ((*a*$*i* ≤ *x*$*i* ∧ *x*$*i* ≤ *b*$*i*) ∨ (*b*$*i* ≤ *x*$*i* ∧ *x*$*i* ≤ *a*$*i*))) ⟶ *x* ∈ *s*)

lemma *is-interval-interval*: *is-interval* {*a* .. *b*::*real^'n*} (**is** *?th1*) *is-interval* {*a*<..<*b*} (**is** *?th2*) **proof** −
have ∗:⋀*x y z*::*real*. *x* < *y* ⟹ *y* < *z* ⟹ *x* < *z* **by** *auto*
show *?th1 ?th2* **unfolding** *is-interval-def mem-interval Ball-def atLeastAtMost-iff*
by(*meson order-trans le-less-trans less-le-trans* ∗)+ **qed**

lemma *is-interval-empty*:
is-interval {}
unfolding *is-interval-def*
by *simp*

lemma *is-interval-univ*:
is-interval UNIV
unfolding *is-interval-def*
by *simp*

16.25 Closure of halfspaces and hyperplanes.

lemma *Lim-inner*:
assumes (*f* −−−> *l*) *net* **shows** ((λ*y*. *inner a* (*f y*)) −−−> *inner a l*) *net*
by (*intro tendsto-intros assms*)

lemma *continuous-at-inner*: *continuous* (*at x*) (*inner a*)
unfolding *continuous-at* **by** (*intro tendsto-intros*)

lemma *continuous-on-inner*:
fixes *s* :: *'a*::*real-inner set*
shows *continuous-on s* (*inner a*)

unfolding *continuous-on* **by** (*rule ballI*) (*intro tendsto-intros*)

lemma *closed-halfspace-le*: *closed* $\{x.\ inner\ a\ x \leq b\}$
proof−
have $\forall x.\ continuous\ (at\ x)\ (inner\ a)$
unfolding *continuous-at* **by** (*rule allI*) (*intro tendsto-intros*)
hence *closed* (*inner a* −‘ $\{..b\}$)
using *closed-real-atMost* **by** (*rule continuous-closed-vimage*)
moreover have $\{x.\ inner\ a\ x \leq b\} = inner\ a$ −‘ $\{..b\}$ **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

lemma *closed-halfspace-ge*: *closed* $\{x.\ inner\ a\ x \geq b\}$
using *closed-halfspace-le*[*of* −*a* −*b*] **unfolding** *inner-minus-left* **by** *auto*

lemma *closed-hyperplane*: *closed* $\{x.\ inner\ a\ x = b\}$
proof−
have $\{x.\ inner\ a\ x = b\} = \{x.\ inner\ a\ x \geq b\} \cap \{x.\ inner\ a\ x \leq b\}$ **by** *auto*
thus *?thesis* **using** *closed-halfspace-le*[*of a b*] **and** *closed-halfspace-ge*[*of b a*]
using *closed-Int* **by** *auto*
qed

lemma *closed-halfspace-component-le*:
shows *closed* $\{x::real\hat{}'n.\ x\$i \leq a\}$
using *closed-halfspace-le*[*of* (*basis i*)::*real*^*'n a*] **unfolding** *inner-basis*[*OF assms*]
by *auto*

lemma *closed-halfspace-component-ge*:
shows *closed* $\{x::real\hat{}'n.\ x\$i \geq a\}$
using *closed-halfspace-ge*[*of a* (*basis i*)::*real*^*'n*] **unfolding** *inner-basis*[*OF assms*]
by *auto*

Openness of halfspaces.

lemma *open-halfspace-lt*: *open* $\{x.\ inner\ a\ x < b\}$
proof−
have $-\{x.\ b \leq inner\ a\ x\} = \{x.\ inner\ a\ x < b\}$ **by** *auto*
thus *?thesis* **using** *closed-halfspace-ge*[*unfolded closed-def*, *of b a*] **by** *auto*
qed

lemma *open-halfspace-gt*: *open* $\{x.\ inner\ a\ x > b\}$
proof−
have $-\{x.\ b \geq inner\ a\ x\} = \{x.\ inner\ a\ x > b\}$ **by** *auto*
thus *?thesis* **using** *closed-halfspace-le*[*unfolded closed-def*, *of a b*] **by** *auto*
qed

lemma *open-halfspace-component-lt*:
shows *open* $\{x::real\hat{}'n.\ x\$i < a\}$
using *open-halfspace-lt*[*of* (*basis i*)::*real*^*'n a*] **unfolding** *inner-basis*[*OF assms*]
by *auto*

lemma *open-halfspace-component-gt*:
shows *open* {*x*::*real^'n*. *x*$*i* > *a*}
using *open-halfspace-gt*[*of a* (*basis i*)::*real^'n*] **unfolding** *inner-basis*[*OF assms*]
by *auto*

This gives a simple derivation of limit component bounds.

lemma *Lim-component-le*: **fixes** *f* :: *'a* ⇒ *real^'n*
assumes (*f* ---> *l*) *net* ¬ (*trivial-limit net*) *eventually* (λ*x*. *f*(*x*)$*i* ≤ *b*) *net*
shows *l*$*i* ≤ *b*
proof−
{ fix *x* **have** *x* ∈ {*x*::*real^'n*. *inner* (*basis i*) *x* ≤ *b*} ⟷ *x*$*i* ≤ *b* **unfolding** *inner-basis* **by** *auto* **} note** ∗ = *this*
show *?thesis* **using** *Lim-in-closed-set*[*of* {*x*. *inner* (*basis i*) *x* ≤ *b*} *f net l*] **unfolding** ∗
using *closed-halfspace-le*[*of* (*basis i*)::*real^'n b*] **and** *assms*(*1*,*2*,*3*) **by** *auto*
qed

lemma *Lim-component-ge*: **fixes** *f* :: *'a* ⇒ *real^'n*
assumes (*f* ---> *l*) *net* ¬ (*trivial-limit net*) *eventually* (λ*x*. *b* ≤ (*f x*)$*i*) *net*
shows *b* ≤ *l*$*i*
proof−
{ fix *x* **have** *x* ∈ {*x*::*real^'n*. *inner* (*basis i*) *x* ≥ *b*} ⟷ *x*$*i* ≥ *b* **unfolding** *inner-basis* **by** *auto* **} note** ∗ = *this*
show *?thesis* **using** *Lim-in-closed-set*[*of* {*x*. *inner* (*basis i*) *x* ≥ *b*} *f net l*] **unfolding** ∗
using *closed-halfspace-ge*[*of b* (*basis i*)::*real^'n*] **and** *assms*(*1*,*2*,*3*) **by** *auto*
qed

lemma *Lim-component-eq*: **fixes** *f* :: *'a* ⇒ *real^'n*
assumes *net*:(*f* ---> *l*) *net* ~(*trivial-limit net*) **and** *ev*:*eventually* (λ*x*. *f*(*x*)$*i* = *b*) *net*
shows *l*$*i* = *b*
using *ev*[*unfolded order-eq-iff eventually-and*] **using** *Lim-component-ge*[*OF net*, *of b i*] **and** *Lim-component-le*[*OF net*, *of i b*] **by** *auto*

Limits relative to a union.

lemma *eventually-within-Un*:
eventually P (*net within* (*s* ∪ *t*)) ⟷
eventually P (*net within s*) ∧ *eventually P* (*net within t*)
unfolding *Limits.eventually-within*
by (*auto elim*!: *eventually-rev-mp*)

lemma *Lim-within-union*:
(*f* ---> *l*) (*net within* (*s* ∪ *t*)) ⟷
(*f* ---> *l*) (*net within s*) ∧ (*f* ---> *l*) (*net within t*)
unfolding *tendsto-def*
by (*auto simp add*: *eventually-within-Un*)

lemma *Lim-topological*:
(*f* −−−> *l*) *net* ⟷
trivial-limit net ∨
(∀ *S*. *open S* ⟶ *l* ∈ *S* ⟶ *eventually* (λ*x*. *f x* ∈ *S*) *net*)
unfolding *tendsto-def trivial-limit-eq* **by** *auto*

lemma *continuous-on-union*:
assumes *closed s closed t continuous-on s f continuous-on t f*
shows *continuous-on* (*s* ∪ *t*) *f*
using *assms* **unfolding** *continuous-on Lim-within-union*
unfolding *Lim-topological trivial-limit-within closed-limpt* **by** *auto*

lemma *continuous-on-cases*:
assumes *closed s closed t continuous-on s f continuous-on t g*
∀ *x*. (*x*∈*s* ∧ ¬ *P x*) ∨ (*x* ∈ *t* ∧ *P x*) ⟶ *f x* = *g x*
shows *continuous-on* (*s* ∪ *t*) (λ*x*. *if P x then f x else g x*)
proof−
let *?h* = (λ*x*. *if P x then f x else g x*)
have ∀ *x*∈*s*. *f x* = (*if P x then f x else g x*) **using** *assms*(*5*) **by** *auto*
hence *continuous-on s ?h* **using** *continuous-on-eq*[*of s f ?h*] **using** *assms*(*3*) **by** *auto*
moreover
have ∀ *x*∈*t*. *g x* = (*if P x then f x else g x*) **using** *assms*(*5*) **by** *auto*
hence *continuous-on t ?h* **using** *continuous-on-eq*[*of t g ?h*] **using** *assms*(*4*) **by** *auto*
ultimately show *?thesis* **using** *continuous-on-union*[*OF assms*(*1*,*2*), *of ?h*] **by** *auto*
qed

Some more convenient intermediate-value theorem formulations.

lemma *connected-ivt-hyperplane*:
assumes *connected s x* ∈ *s y* ∈ *s inner a x* ≤ *b b* ≤ *inner a y*
shows ∃ *z* ∈ *s*. *inner a z* = *b*
proof(*rule ccontr*)
assume *as*:¬ (∃ *z*∈*s*. *inner a z* = *b*)
let *?A* = {*x*. *inner a x* < *b*}
let *?B* = {*x*. *inner a x* > *b*}
have *open ?A open ?B* **using** *open-halfspace-lt* **and** *open-halfspace-gt* **by** *auto*
moreover have *?A* ∩ *?B* = {} **by** *auto*
moreover have *s* ⊆ *?A* ∪ *?B* **using** *as* **by** *auto*
ultimately show *False* **using** *assms*(*1*)[*unfolded connected-def not-ex*, *THEN spec*[**where** *x*=*?A*], *THEN spec*[**where** *x*=*?B*]] **and** *assms*(*2*−*5*) **by** *auto*
qed

lemma *connected-ivt-component*: **fixes** *x*::*real*^'*n* **shows**
connected s ⟹ *x* ∈ *s* ⟹ *y* ∈ *s* ⟹ *x*$*k* ≤ *a* ⟹ *a* ≤ *y*$*k* ⟹ (∃ *z*∈*s*. *z*$*k* = *a*)
using *connected-ivt-hyperplane*[*of s x y* (*basis k*)::*real*^'*n a*] **by** (*auto simp add*: *inner-basis*)

16.26 Homeomorphisms

definition *homeomorphism s t f g* ≡
(∀ *x*∈*s*. (*g*(*f x*) = *x*)) ∧ (*f* ‘ *s* = *t*) ∧ *continuous-on s f* ∧
(∀ *y*∈*t*. (*f*(*g y*) = *y*)) ∧ (*g* ‘ *t* = *s*) ∧ *continuous-on t g*

definition
homeomorphic :: ′*a*::*metric-space set* ⇒ ′*b*::*metric-space set* ⇒ *bool*
(**infixr** *homeomorphic 60*) **where**
homeomorphic-def: *s homeomorphic t* ≡ (∃*f g*. *homeomorphism s t f g*)

lemma *homeomorphic-refl*: *s homeomorphic s*
unfolding *homeomorphic-def*
unfolding *homeomorphism-def*
using *continuous-on-id*
apply(*rule-tac x* = (λ*x*. *x*) **in** *exI*)
apply(*rule-tac x* = (λ*x*. *x*) **in** *exI*)
by *blast*

lemma *homeomorphic-sym*:
s homeomorphic t ⟷ *t homeomorphic s*
unfolding *homeomorphic-def*
unfolding *homeomorphism-def*
by *blast*

lemma *homeomorphic-trans*:
assumes *s homeomorphic t t homeomorphic u* **shows** *s homeomorphic u*
proof−
obtain *f1 g1* **where** *fg1*:∀ *x*∈*s*. *g1* (*f1 x*) = *x* *f1* ‘ *s* = *t continuous-on s f1*
∀ *y*∈*t*. *f1* (*g1 y*) = *y g1* ‘ *t* = *s continuous-on t g1*
using *assms*(*1*) **unfolding** *homeomorphic-def homeomorphism-def* **by** *auto*
obtain *f2 g2* **where** *fg2*:∀ *x*∈*t*. *g2* (*f2 x*) = *x* *f2* ‘ *t* = *u continuous-on t f2*
∀ *y*∈*u*. *f2* (*g2 y*) = *y g2* ‘ *u* = *t continuous-on u g2*
using *assms*(*2*) **unfolding** *homeomorphic-def homeomorphism-def* **by** *auto*

{ fix *x* **assume** *x*∈*s* **hence** (*g1* ∘ *g2*) ((*f2* ∘ *f1*) *x*) = *x* **using** *fg1*(*1*)[*THEN bspec*[**where** *x*=*x*]] **and** *fg2*(*1*)[*THEN bspec*[**where** *x*=*f1 x*]] **and** *fg1*(*2*) **by** *auto*
}
moreover have (*f2* ∘ *f1*) ‘ *s* = *u* **using** *fg1*(*2*) *fg2*(*2*) **by** *auto*
moreover have *continuous-on s* (*f2* ∘ *f1*) **using** *continuous-on-compose*[*OF fg1*(*3*)] **and** *fg2*(*3*) **unfolding** *fg1*(*2*) **by** *auto*
moreover { fix *y* **assume** *y*∈*u* **hence** (*f2* ∘ *f1*) ((*g1* ∘ *g2*) *y*) = *y* **using** *fg2*(*4*)[*THEN bspec*[**where** *x*=*y*]] **and** *fg1*(*4*)[*THEN bspec*[**where** *x*=*g2 y*]] **and** *fg2*(*5*) **by** *auto* **}**
moreover have (*g1* ∘ *g2*) ‘ *u* = *s* **using** *fg1*(*5*) *fg2*(*5*) **by** *auto*
moreover have *continuous-on u* (*g1* ∘ *g2*) **using** *continuous-on-compose*[*OF fg2*(*6*)] **and** *fg1*(*6*) **unfolding** *fg2*(*5*) **by** *auto*
ultimately show *?thesis* **unfolding** *homeomorphic-def homeomorphism-def* **apply**(*rule-tac x*=*f2* ∘ *f1* **in** *exI*) **apply**(*rule-tac x*=*g1* ∘ *g2* **in** *exI*) **by** *auto*
qed

lemma *homeomorphic-minimal*:
s homeomorphic t $\longleftrightarrow$
$(\exists f\ g.\ (\forall x{\in}s.\ f(x) \in t \wedge (g(f(x)) = x)) \wedge$
$(\forall y{\in}t.\ g(y) \in s \wedge (f(g(y)) = y)) \wedge$
continuous-on s f $\wedge$ *continuous-on t g*)
unfolding *homeomorphic-def homeomorphism-def*
apply *auto* **apply** (*rule-tac x=f* **in** *exI*) **apply** (*rule-tac x=g* **in** *exI*)
apply *auto* **apply** (*rule-tac x=f* **in** *exI*) **apply** (*rule-tac x=g* **in** *exI*) **apply** *auto*
unfolding *image-iff*
apply(*erule-tac x=g x* **in** *ballE*) **apply**(*erule-tac x=x* **in** *ballE*)
apply *auto* **apply**(*rule-tac x=g x* **in** *bexI*) **apply** *auto*
apply(*erule-tac x=f x* **in** *ballE*) **apply**(*erule-tac x=x* **in** *ballE*)
apply *auto* **apply**(*rule-tac x=f x* **in** *bexI*) **by** *auto*

Relatively weak hypotheses if a set is compact.

lemma *homeomorphism-compact*:
fixes *f* :: *'a*::*heine-borel* $\Rightarrow$ *'b*::*heine-borel*

assumes *compact s continuous-on s f f ' s = t inj-on f s*
shows $\exists g.$ *homeomorphism s t f g*
proof−
def $g \equiv \lambda x.$ *SOME y.* $y{\in}s \wedge f\ y = x$
have *g*:$\forall x{\in}s.\ g\ (f\ x) = x$ **using** *assms*(*3*) *assms*(*4*)[*unfolded inj-on-def*] **unfolding** *g-def* **by** *auto*
{ fix *y* **assume** $y{\in}t$
then obtain *x* **where** *x*:$f\ x = y\ x{\in}s$ **using** *assms*(*3*) **by** *auto*
hence $g\ (f\ x) = x$ **using** *g* **by** *auto*
hence $f\ (g\ y) = y$ **unfolding** *x*(*1*)[*THEN sym*] **by** *auto* **}**
hence *g'*:$\forall x{\in}t.\ f\ (g\ x) = x$ **by** *auto*
moreover
{ fix *x*
have $x{\in}s \Longrightarrow x \in g\ `\ t$ **using** *g*[*THEN bspec*[**where** *x=x*]] **unfolding** *image-iff* **using** *assms*(*3*) **by**(*auto intro*!: *bexI*[**where** *x=f x*])
moreover
{ assume $x{\in}g\ `\ t$
then obtain *y* **where** *y*:$y{\in}t\ g\ y = x$ **by** *auto*
then obtain *x'* **where** *x'*:$x'{\in}s\ f\ x' = y$ **using** *assms*(*3*) **by** *auto*
hence $x \in s$ **unfolding** *g-def* **using** *someI2*[*of* $\lambda b.\ b{\in}s \wedge f\ b = y\ x'\ \lambda x.\ x{\in}s$] **unfolding** *y*(*2*)[*THEN sym*] **and** *g-def* **by** *auto* **}**
ultimately have $x{\in}s \longleftrightarrow x \in g\ `\ t$ **.. }**
hence $g\ `\ t = s$ **by** *auto*
ultimately
show *?thesis* **unfolding** *homeomorphism-def homeomorphic-def*
apply(*rule-tac x=g* **in** *exI*) **using** *g* **and** *assms*(*3*) **and** *continuous-on-inverse*[*OF assms*(*2*,*1*), *of g*, *unfolded assms*(*3*)] **and** *assms*(*2*) **by** *auto*
qed

lemma *homeomorphic-compact*:

fixes f :: $'a$::*heine-borel* ⇒ $'b$::*heine-borel*

shows *compact s* ⟹ *continuous-on s f* ⟹ (*f* ‘ *s* = *t*) ⟹ *inj-on f s*
⟹ *s homeomorphic t*
unfolding *homeomorphic-def* **by**(*metis homeomorphism-compact*)

Preservation of topological properties.

lemma *homeomorphic-compactness*:
s homeomorphic t ==> (*compact s* ⟷ *compact t*)
unfolding *homeomorphic-def homeomorphism-def*
by (*metis compact-continuous-image*)

Results on translation, scaling etc.

lemma *homeomorphic-scaling*:
fixes *s* :: $'a$::*real-normed-vector set*
assumes *c* ≠ *0* **shows** *s homeomorphic* ((λ*x*. *c* ∗$_R$ *x*) ‘ *s*)
unfolding *homeomorphic-minimal*
apply(*rule-tac x*=λ*x*. *c* ∗$_R$ *x* **in** *exI*)
apply(*rule-tac x*=λ*x*. (*1* / *c*) ∗$_R$ *x* **in** *exI*)
using *assms* **apply** *auto*
using *continuous-on-cmul*[*OF continuous-on-id*] **by** *auto*

lemma *homeomorphic-translation*:
fixes *s* :: $'a$::*real-normed-vector set*
shows *s homeomorphic* ((λ*x*. *a* + *x*) ‘ *s*)
unfolding *homeomorphic-minimal*
apply(*rule-tac x*=λ*x*. *a* + *x* **in** *exI*)
apply(*rule-tac x*=λ*x*. −*a* + *x* **in** *exI*)
using *continuous-on-add*[*OF continuous-on-const continuous-on-id*] **by** *auto*

lemma *homeomorphic-affinity*:
fixes *s* :: $'a$::*real-normed-vector set*
assumes *c* ≠ *0* **shows** *s homeomorphic* ((λ*x*. *a* + *c* ∗$_R$ *x*) ‘ *s*)
proof−
have ∗:*op* + *a* ‘ *op* ∗$_R$ *c* ‘ *s* = (λ*x*. *a* + *c* ∗$_R$ *x*) ‘ *s* **by** *auto*
show *?thesis*
using *homeomorphic-trans*
using *homeomorphic-scaling*[*OF assms*, *of s*]
using *homeomorphic-translation*[*of* (λ*x*. *c* ∗$_R$ *x*) ‘ *s a*] **unfolding** ∗ **by** *auto*
qed

lemma *homeomorphic-balls*:
fixes *a b* ::$'a$::*real-normed-vector*
assumes *0* < *d* *0* < *e*
shows (*ball a d*) *homeomorphic* (*ball b e*) (**is** *?th*)
(*cball a d*) *homeomorphic* (*cball b e*) (**is** *?cth*)
proof−
have ∗:|*e* / *d*| > *0* |*d* / *e*| >*0* **using** *assms* **using** *divide-pos-pos* **by** *auto*
show *?th* **unfolding** *homeomorphic-minimal*

apply(*rule-tac* x=$\lambda x.\ b + (e/d) *_R (x - a)$ **in** *exI*)
apply(*rule-tac* x=$\lambda x.\ a + (d/e) *_R (x - b)$ **in** *exI*)
using *assms* **apply** (*auto simp add*: *dist-commute*)
unfolding *dist-norm*
apply (*auto simp add*: *pos-divide-less-eq mult-strict-left-mono*)
unfolding *continuous-on*
by (*intro ballI tendsto-intros*, *simp*)+
next
have $*$:$|e\ /\ d| > 0\ |d\ /\ e| > 0$ **using** *assms* **using** *divide-pos-pos* **by** *auto*
show *?cth* **unfolding** *homeomorphic-minimal*
apply(*rule-tac* x=$\lambda x.\ b + (e/d) *_R (x - a)$ **in** *exI*)
apply(*rule-tac* x=$\lambda x.\ a + (d/e) *_R (x - b)$ **in** *exI*)
using *assms* **apply** (*auto simp add*: *dist-commute*)
unfolding *dist-norm*
apply (*auto simp add*: *pos-divide-le-eq*)
unfolding *continuous-on*
by (*intro ballI tendsto-intros*, *simp*)+
qed

"Isometry" (up to constant bounds) of injective linear map etc.

lemma *cauchy-isometric*:
fixes x :: *nat* ⇒ *real* ^ *'n*
assumes e:$0 < e$ **and** s:*subspace s* **and** f:*bounded-linear f* **and** *normf*:$\forall x{\in}s.$ $norm(f\ x) \geq e * norm(x)$ **and** xs:$\forall n{::}nat.\ x\ n \in s$ **and** cf:*Cauchy*$(f\ o\ x)$
shows *Cauchy x*
proof−
interpret f: *bounded-linear f* **by** *fact*
{ fix d::*real* **assume** $d>0$
then obtain N **where** N:$\forall n{\geq}N.\ norm\ (f\ (x\ n) - f\ (x\ N)) < e * d$
using *cf*[*unfolded cauchy o-def dist-norm*, *THEN spec*[**where** x=$e*d$]] **and** *e* **and** *mult-pos-pos*[*of e d*] **by** *auto*
{ fix n **assume** $n{\geq}N$
hence $norm\ (f\ (x\ n - x\ N)) < e * d$ **using** N[*THEN spec*[**where** x=n]] **unfolding** *f.diff*[*THEN sym*] **by** *auto*
moreover have $e * norm\ (x\ n - x\ N) \leq norm\ (f\ (x\ n - x\ N))$
using *subspace-sub*[*OF s*, *of x n x N*] **using** *xs*[*THEN spec*[**where** x=N]] **and** *xs*[*THEN spec*[**where** x=n]]
using *normf*[*THEN bspec*[**where** x=$x\ n - x\ N$]] **by** *auto*
ultimately have $norm\ (x\ n - x\ N) < d$ **using** ‹$e>0$›
using *mult-left-less-imp-less*[*of e norm* $(x\ n - x\ N)$ *d*] **by** *auto* **}**
hence $\exists N.\ \forall n{\geq}N.\ norm\ (x\ n - x\ N) < d$ **by** *auto* **}**
thus *?thesis* **unfolding** *cauchy* **and** *dist-norm* **by** *auto*
qed

lemma *complete-isometric-image*:
fixes f :: *real* ^ *-* ⇒ *real* ^ *-*
assumes $0 < e$ **and** s:*subspace s* **and** f:*bounded-linear f* **and** *normf*:$\forall x{\in}s.$ $norm(f\ x) \geq e * norm(x)$ **and** cs:*complete s*
shows *complete*$(f\ `\ s)$

proof−
{ fix *g* **assume** *as*:∀ *n*::*nat*. *g n* ∈ *f* ‘ *s* **and** *cfg*:*Cauchy g*
then obtain *x* **where** ∀ *n*. *x n* ∈ *s* ∧ *g n* = *f* (*x n*)
using *choice*[*of* λ *n xa*. *xa* ∈ *s* ∧ *g n* = *f xa*] **by** *auto*
hence *x*:∀ *n*. *x n* ∈ *s* ∀ *n*. *g n* = *f* (*x n*) **by** *auto*
hence *f* ∘ *x* = *g* **unfolding** *expand-fun-eq* **by** *auto*
then obtain *l* **where** *l*∈*s* **and** *l*:(*x* −−−> *l*) *sequentially*
using *cs*[*unfolded complete-def*, *THEN spec*[**where** *x*=*x*]]
using *cauchy-isometric*[*OF* ‹*0*<*e*› *s f normf*] **and** *cfg* **and** *x*(*1*) **by** *auto*
hence ∃ *l*∈*f* ‘ *s*. (*g* −−−> *l*) *sequentially*
using *linear-continuous-at*[*OF f*, *unfolded continuous-at-sequentially*, *THEN spec*[**where** *x*=*x*], *of l*]
unfolding ‹*f* ∘ *x* = *g*› **by** *auto* **}**
thus *?thesis* **unfolding** *complete-def* **by** *auto*
qed

lemma *dist-0-norm*:
fixes *x* :: ′*a*::*real-normed-vector*
shows *dist 0 x* = *norm x*
unfolding *dist-norm* **by** *simp*

lemma *injective-imp-isometric*: **fixes** *f*::*real*^′*m* ⇒ *real*^′*n*
assumes *s*:*closed s* *subspace s* **and** *f*:*bounded-linear f* ∀ *x*∈*s*. (*f x* = *0*) ⟶ (*x* = *0*)
shows ∃ *e*>*0*. ∀ *x*∈*s*. *norm* (*f x*) ≥ *e* ∗ *norm*(*x*)
proof(*cases s* ⊆ {*0*::*real*^′*m*})
case *True*
{ fix *x* **assume** *x* ∈ *s*
hence *x* = *0* **using** *True* **by** *auto*
hence *norm x* ≤ *norm* (*f x*) **by** *auto* **}**
thus *?thesis* **by**(*auto intro*!: *exI*[**where** *x*=*1*])
next
interpret *f*: *bounded-linear f* **by** *fact*
case *False*
then obtain *a* **where** *a*:*a*≠*0* *a*∈*s* **by** *auto*
from *False* **have** *s* ≠ {} **by** *auto*
let *?S* = {*f x*| *x*. (*x* ∈ *s* ∧ *norm x* = *norm a*)}
let *?S*′ = {*x*::*real*^′*m*. *x*∈*s* ∧ *norm x* = *norm a*}
let *?S*″ = {*x*::*real*^′*m*. *norm x* = *norm a*}

have *?S*″ = *frontier*(*cball 0* (*norm a*)) **unfolding** *frontier-cball* **and** *dist-norm* **by** *auto*
hence *compact ?S*″ **using** *compact-frontier*[*OF compact-cball*, *of 0 norm a*] **by** *auto*
moreover have *?S*′ = *s* ∩ *?S*″ **by** *auto*
ultimately have *compact ?S*′ **using** *closed-inter-compact*[*of s ?S*″] **using** *s*(*1*) **by** *auto*
moreover have ∗:*f* ‘ *?S*′ = *?S* **by** *auto*
ultimately have *compact ?S* **using** *compact-continuous-image*[*OF linear-continuous-on*[*OF*

f(1)], *of ?S'*] **by** *auto*
hence *closed ?S* **using** *compact-imp-closed* **by** *auto*
moreover have *?S* ≠ {} **using** *a* **by** *auto*
ultimately obtain *b'* **where** *b'*∈*?S* ∀ *y*∈*?S*. *norm b'* ≤ *norm y* **using** *distance-attains-inf*[*of ?S 0*] **unfolding** *dist-0-norm* **by** *auto*
then obtain *b* **where** *b*∈*s* **and** *ba*:*norm b* = *norm a* **and** *b*:∀ *x*∈{*x* ∈ *s*. *norm x* = *norm a*}. *norm* (*f b*) ≤ *norm* (*f x*) **unfolding** ∗[*THEN sym*] **unfolding** *image-iff* **by** *auto*

let *?e* = *norm* (*f b*) / *norm b*
have *norm b* > *0* **using** *ba* **and** *a* **and** *norm-ge-zero* **by** *auto*
moreover have *norm* (*f b*) > *0* **using** *f*(*2*)[*THEN bspec*[**where** *x*=*b*], *OF* ‹*b*∈*s*›] **using** ‹*norm b* >*0*› **unfolding** *zero-less-norm-iff* **by** *auto*
ultimately have *0* < *norm* (*f b*) / *norm b* **by**(*simp only*: *divide-pos-pos*)
moreover
{ fix *x* **assume** *x*∈*s*
hence *norm* (*f b*) / *norm b* ∗ *norm x* ≤ *norm* (*f x*)
proof(*cases x*=*0*)
case *True* **thus** *norm* (*f b*) / *norm b* ∗ *norm x* ≤ *norm* (*f x*) **by** *auto*
next
case *False*
hence ∗:*0* < *norm a* / *norm x* **using** ‹*a*≠*0*› **unfolding** *zero-less-norm-iff*[*THEN sym*] **by**(*simp only*: *divide-pos-pos*)
have ∀ *c*. ∀ *x*∈*s*. *c* ∗*R* *x* ∈ *s* **using** *s*[*unfolded subspace-def smult-conv-scaleR*] **by** *auto*
hence (*norm a* / *norm x*) ∗*R* *x* ∈ {*x* ∈ *s*. *norm x* = *norm a*} **using** ‹*x*∈*s*› **and** ‹*x*≠*0*› **by** *auto*
thus *norm* (*f b*) / *norm b* ∗ *norm x* ≤ *norm* (*f x*) **using** *b*[*THEN bspec*[**where** *x*=(*norm a* / *norm x*) ∗*R* *x*]]
unfolding *f.scaleR* **and** *ba* **using** ‹*x*≠*0*› ‹*a*≠*0*›
by (*auto simp add*: *mult-commute pos-le-divide-eq pos-divide-le-eq*)
qed }
ultimately
show *?thesis* **by** *auto*
qed

lemma *closed-injective-image-subspace*:
fixes *f* :: *real* ^ - ⇒ *real* ^ -
assumes *subspace s bounded-linear f* ∀ *x*∈*s*. *f x* = *0* −−> *x* = *0 closed s*
shows *closed*(*f* ‘ *s*)
proof−
obtain *e* **where** *e*>*0* **and** *e*:∀ *x*∈*s*. *e* ∗ *norm x* ≤ *norm* (*f x*) **using** *injective-imp-isometric*[*OF assms*(*4*,*1*,*2*,*3*)] **by** *auto*
show *?thesis* **using** *complete-isometric-image*[*OF* ‹*e*>*0*› *assms*(*1*,*2*) *e*] **and** *assms*(*4*)
unfolding *complete-eq-closed*[*THEN sym*] **by** *auto*
qed

16.27 Some properties of a canonical subspace.

lemma *subspace-substandard*:
subspace {*x*::*real*^-. (∀ *i*. *P i* ⟶ *x*$*i* = *0*)}
unfolding *subspace-def* **by** *auto*

lemma *closed-substandard*:
closed {*x*::*real*^'*n*. ∀ *i*. *P i* −−> *x*$*i* = *0*} (**is** *closed ?A*)
proof−
let *?D* = {*i*. *P i*}
let *?Bs* = {{*x*::*real*^'*n*. *inner* (*basis i*) *x* = *0*}| *i*. *i* ∈ *?D*}
{ fix *x*
{ assume *x*∈*?A*
hence *x*:∀ *i*∈*?D*. *x* $ *i* = *0* **by** *auto*
hence *x*∈ ⋂ *?Bs* **by**(*auto simp add*: *inner-basis x*) **}**
moreover
{ assume *x*:*x*∈⋂ *?Bs*
{ fix *i* **assume** *i*:*i* ∈ *?D*
then obtain *B* **where** *BB*:*B* ∈ *?Bs* **and** *B*:*B* = {*x*::*real*^'*n*. *inner* (*basis i*) *x* = *0*} **by** *auto*
hence *x* $ *i* = *0* **unfolding** *B* **using** *x* **unfolding** *inner-basis* **by** *auto* **}**
hence *x*∈*?A* **by** *auto* **}**
ultimately have *x*∈*?A* ⟷ *x*∈ ⋂ *?Bs* **.. }**
hence *?A* = ⋂ *?Bs* **by** *auto*
thus *?thesis* **by**(*auto simp add*: *closed-Inter closed-hyperplane*)
qed

lemma *dim-substandard*:
shows *dim* {*x*::*real*^'*n*. ∀ *i*. *i* ∉ *d* ⟶ *x*$*i* = *0*} = *card d* (**is** *dim ?A* = -)
proof−
let *?D* = *UNIV*::'*n set*
let *?B* = (*basis*::'*n*⇒*real*^'*n*) ‘ *d*

let *?bas* = *basis*::'*n* ⇒ *real*^'*n*

have *?B* ⊆ *?A* **by** *auto*

moreover
{ fix *x*::*real*^'*n* **assume** *x*∈*?A*
with *finite*[*of d*]
have *x*∈ *span ?B*
proof(*induct d arbitrary*: *x*)
case *empty* **hence** *x*=*0* **unfolding** *Cart-eq* **by** *auto*
thus *?case* **using** *subspace-0*[*OF subspace-span*[*of* {}]] **by** *auto*
next
case (*insert k F*)
hence ∗:∀ *i*. *i* ∉ *insert k F* ⟶ *x* $ *i* = *0* **by** *auto*
have ∗∗:*F* ⊆ *insert k F* **by** *auto*
def *y* ≡ *x* − *x*$*k* ∗$_R$ *basis k*
have *y*:*x* = *y* + (*x*$*k*) ∗$_R$ *basis k* **unfolding** *y-def* **by** *auto*

```
    { fix i assume i':i ∉ F
      hence y $ i = 0 unfolding y-def unfolding vector-minus-component
        and vector-smult-component and basis-component
        using *[THEN spec[where x=i]] by auto }
    hence y ∈ span (basis ` (insert k F)) using insert(3)
      using span-mono[of ?bas ` F ?bas ` (insert k F)]
      using image-mono[OF **, of basis] by auto
    moreover
    have basis k ∈ span (?bas ` (insert k F)) by(rule span-superset, auto)
    hence x$k *R basis k ∈ span (?bas ` (insert k F))
      using span-mul by auto
    ultimately
    have y + x$k *R basis k ∈ span (?bas ` (insert k F))
      using span-add by auto
    thus ?case using y by auto
  qed
 }
 hence ?A ⊆ span ?B by auto

 moreover
 { fix x assume x ∈ ?B
   hence x∈{(basis i)::real^'n |i. i ∈ ?D} using assms by auto  }
 hence independent ?B using independent-mono[OF independent-stdbasis, of ?B]
and assms by auto

 moreover
 have d ⊆ ?D unfolding subset-eq using assms by auto
 hence *:inj-on (basis::'n⇒real^'n) d using subset-inj-on[OF basis-inj, of d] by
auto
 have card ?B = card d unfolding card-image[OF *] by auto

 ultimately show ?thesis using dim-unique[of basis ` d ?A] by auto
qed
```

Hence closure and completeness of all subspaces.

```
lemma closed-subspace-lemma: n ≤ card (UNIV::'n::finite set) ⟹ ∃ A::'n set.
card A = n
apply (induct n)
apply (rule-tac x={} in exI, simp)
apply clarsimp
apply (subgoal-tac ∃x. x ∉ A)
apply (erule exE)
apply (rule-tac x=insert x A in exI, simp)
apply (subgoal-tac A ≠ UNIV, auto)
done

lemma closed-subspace: fixes s::(real^'n) set
  assumes subspace s shows closed s
proof−
```

have *dim s* ≤ *card* (*UNIV* :: *'n set*) **using** *dim-subset-univ* **by** *auto*
then obtain *d*::*'n set* **where** *t*: *card d* = *dim s*
using *closed-subspace-lemma* **by** *auto*
let *?t* = {*x*::*real^'n*. ∀ *i*. *i* ∉ *d* ⟶ *x*$*i* = *0*}
obtain *f* **where** *f*:*bounded-linear f* *f* ' *?t* = *s inj-on f ?t*
using *subspace-isomorphism*[*unfolded linear-conv-bounded-linear*, *OF subspace-substandard*[*of* λ*i*. *i* ∉ *d*] *assms*]
using *dim-substandard*[*of d*] **and** *t* **by** *auto*
interpret *f*: *bounded-linear f* **by** *fact*
have ∀ *x*∈*?t*. *f x* = *0* ⟶ *x* = *0* **using** *f*.*zero* **using** *f*(*3*)[*unfolded inj-on-def*]
by(*erule-tac x=0* **in** *ballE*) *auto*
moreover have *closed ?t* **using** *closed-substandard* **.**
moreover have *subspace ?t* **using** *subspace-substandard* **.**
ultimately show *?thesis* **using** *closed-injective-image-subspace*[*of ?t f*]
unfolding *f*(*2*) **using** *f*(*1*) **by** *auto*
qed

lemma *complete-subspace*:
fixes *s* :: (*real* ^ -) *set* **shows** *subspace s* ==> *complete s*
using *complete-eq-closed closed-subspace*
by *auto*

lemma *dim-closure*:
fixes *s* :: (*real* ^ -) *set*
shows *dim*(*closure s*) = *dim s* (**is** *?dc* = *?d*)
proof−
have *?dc* ≤ *?d* **using** *closure-minimal*[*OF span-inc*, *of s*]
using *closed-subspace*[*OF subspace-span*, *of s*]
using *dim-subset*[*of closure s span s*] **unfolding** *dim-span* **by** *auto*
thus *?thesis* **using** *dim-subset*[*OF closure-subset*, *of s*] **by** *auto*
qed

16.28 Affine transformations of intervals

lemma *affinity-inverses*:
assumes *m0*: *m* ≠ (*0*::*'a*::*field*)
shows (λ*x*. *m* ∗*s x* + *c*) *o* (λ*x*. *inverse*(*m*) ∗*s x* + (−(*inverse*(*m*) ∗*s c*))) = *id*
(λ*x*. *inverse*(*m*) ∗*s x* + (−(*inverse*(*m*) ∗*s c*))) *o* (λ*x*. *m* ∗*s x* + *c*) = *id*
using *m0*
apply (*auto simp add*: *expand-fun-eq vector-add-ldistrib vector-smult-assoc*)
by (*simp add*: *vector-smult-lneg*[*symmetric*] *vector-smult-assoc vector-sneg-minus1*[*symmetric*])

lemma *real-affinity-le*:
0 < (*m*::*'a*::*linordered-field*) ==> (*m* ∗ *x* + *c* ≤ *y* ⟷ *x* ≤ *inverse*(*m*) ∗ *y* + −(*c* / *m*))
by (*simp add*: *field-simps inverse-eq-divide*)

lemma *real-le-affinity*:
0 < (*m*::*'a*::*linordered-field*) ==> (*y* ≤ *m* ∗ *x* + *c* ⟷ *inverse*(*m*) ∗ *y* + −(*c* /

$m) \leq x)$
by (*simp add*: *field-simps inverse-eq-divide*)

lemma *real-affinity-lt*:
$0 < (m::'a::linordered\text{-}field) ==> (m * x + c < y \longleftrightarrow x < inverse(m) * y + -(c\ /\ m))$
by (*simp add*: *field-simps inverse-eq-divide*)

lemma *real-lt-affinity*:
$0 < (m::'a::linordered\text{-}field) ==> (y < m * x + c \longleftrightarrow inverse(m) * y + -(c\ /\ m) < x)$
by (*simp add*: *field-simps inverse-eq-divide*)

lemma *real-affinity-eq*:
$(m::'a::linordered\text{-}field) \neq 0 ==> (m * x + c = y \longleftrightarrow x = inverse(m) * y + -(c\ /\ m))$
by (*simp add*: *field-simps inverse-eq-divide*)

lemma *real-eq-affinity*:
$(m::'a::linordered\text{-}field) \neq 0 ==> (y = m * x + c \longleftrightarrow inverse(m) * y + -(c\ /\ m) = x)$
by (*simp add*: *field-simps inverse-eq-divide*)

lemma *vector-affinity-eq*:
assumes *m0*: $(m::'a::field) \neq 0$
shows $m *s x + c = y \longleftrightarrow x = inverse\ m *s y + -(inverse\ m *s c)$
proof
assume *h*: $m *s x + c = y$
hence $m *s x = y - c$ **by** (*simp add*: *field-simps*)
hence $inverse\ m *s (m *s x) = inverse\ m *s (y - c)$ **by** *simp*
then show $x = inverse\ m *s y + - (inverse\ m *s c)$
using *m0* **by** (*simp add*: *vector-smult-assoc vector-ssub-ldistrib*)
next
assume *h*: $x = inverse\ m *s y + - (inverse\ m *s c)$
show $m *s x + c = y$ **unfolding** *h diff-minus*[*symmetric*]
using *m0* **by** (*simp add*: *vector-smult-assoc vector-ssub-ldistrib*)
qed

lemma *vector-eq-affinity*:
$(m::'a::field) \neq 0 ==> (y = m *s x + c \longleftrightarrow inverse(m) *s y + -(inverse(m) *s c) = x)$
using *vector-affinity-eq*[**where** *m*=*m* **and** *x*=*x* **and** *y*=*y* **and** *c*=*c*]
by *metis*

lemma *image-affinity-interval*: **fixes** *m*::*real*
fixes *a b c* :: *real^'n*
shows $(\lambda x.\ m *_R x + c)\ `\ \{a\ ..\ b\} =$
$(if\ \{a\ ..\ b\} = \{\}\ then\ \{\}$
$else\ (if\ 0 \leq m\ then\ \{m *_R a + c\ ..\ m *_R b + c\}$

```
          else {m *R b + c .. m *R a + c}))
proof(cases m=0)
  { fix x assume x ≤ c c ≤ x
   hence x=c unfolding vector-le-def and Cart-eq by (auto intro: order-antisym)
}
  moreover case True
  moreover have c ∈ {m *R a + c..m *R b + c} unfolding True by(auto simp
add: vector-le-def)
  ultimately show ?thesis by auto
next
  case False
  { fix y assume a ≤ y y ≤ b m > 0
    hence m *R a + c ≤ m *R y + c  m *R y + c ≤ m *R b + c
      unfolding vector-le-def by auto
  } moreover
  { fix y assume a ≤ y y ≤ b m < 0
    hence m *R b + c ≤ m *R y + c  m *R y + c ≤ m *R a + c
      unfolding vector-le-def by(auto simp add: mult-left-mono-neg)
  } moreover
  { fix y assume m > 0  m *R a + c ≤ y  y ≤ m *R b + c
    hence y ∈ (λx. m *R x + c) ' {a..b}
      unfolding image-iff Bex-def mem-interval vector-le-def
      apply(auto simp add: vector-smult-assoc pth-3[symmetric]
        intro!: exI[where x=(1 / m) *R (y - c)])
      by(auto simp add: pos-le-divide-eq pos-divide-le-eq mult-commute diff-le-iff)
  } moreover
  { fix y assume m *R b + c ≤ y y ≤ m *R a + c m < 0
    hence y ∈ (λx. m *R x + c) ' {a..b}
      unfolding image-iff Bex-def mem-interval vector-le-def
      apply(auto simp add: vector-smult-assoc pth-3[symmetric]
        intro!: exI[where x=(1 / m) *R (y - c)])
      by(auto simp add: neg-le-divide-eq neg-divide-le-eq mult-commute diff-le-iff)
  }
  ultimately show ?thesis using False by auto
qed

lemma image-smult-interval:(λx. m *R (x::real^'n)) ' {a..b} =
  (if {a..b} = {} then {} else if 0 ≤ m then {m *R a..m *R b} else {m *R b..m
*R a})
  using image-affinity-interval[of m 0 a b] by auto
```

16.29 Banach fixed point theorem (not really topological...)

```
lemma banach-fix:
  assumes s:complete s s ≠ {} and c:0 ≤ c c < 1 and f:(f ' s) ⊆ s and
        lipschitz:∀ x∈s. ∀ y∈s. dist (f x) (f y) ≤ c * dist x y
  shows ∃! x∈s. (f x = x)
proof-
  have 1 - c > 0 using c by auto
```

from *s(2)* **obtain** *z0* **where** *z0* ∈ *s* **by** *auto*
def *z* ≡ λ*n*. (*f* ^^ *n*) *z0*
{ fix *n*::*nat*
have *z n* ∈ *s* **unfolding** *z-def*
proof(*induct n*) **case** *0* **thus** *?case* **using** ‹*z0* ∈*s*› **by** *auto*
next case *Suc* **thus** *?case* **using** *f* **by** *auto* **qed** }
note *z-in-s* = *this*

def *d* ≡ *dist* (*z 0*) (*z 1*)

have *fzn*:⋀*n*. *f* (*z n*) = *z* (*Suc n*) **unfolding** *z-def* **by** *auto*
{ fix *n*::*nat*
have *dist* (*z n*) (*z* (*Suc n*)) ≤ (*c* ^ *n*) * *d*
proof(*induct n*)
case *0* **thus** *?case* **unfolding** *d-def* **by** *auto*
next
case (*Suc m*)
hence *c* * *dist* (*z m*) (*z* (*Suc m*)) ≤ *c* ^ *Suc m* * *d*
using ‹*0* ≤ *c*› **using** *mult-mono1-class.mult-mono1*[*of dist* (*z m*) (*z* (*Suc m*)) *c* ^ *m* * *d c*] **by** *auto*
thus *?case* **using** *lipschitz*[*THEN bspec*[**where** *x*=*z m*], *OF z-in-s*, *THEN bspec*[**where** *x*=*z* (*Suc m*)], *OF z-in-s*]
unfolding *fzn* **and** *mult-le-cancel-left* **by** *auto*
qed
} note *cf-z* = *this*

{ fix *n m*::*nat*
have (*1* − *c*) * *dist* (*z m*) (*z* (*m*+*n*)) ≤ (*c* ^ *m*) * *d* * (*1* − *c* ^ *n*)
proof(*induct n*)
case *0* **show** *?case* **by** *auto*
next
case (*Suc k*)
have (*1* − *c*) * *dist* (*z m*) (*z* (*m* + *Suc k*)) ≤ (*1* − *c*) * (*dist* (*z m*) (*z* (*m* + *k*)) + *dist* (*z* (*m* + *k*)) (*z* (*Suc* (*m* + *k*))))
using *dist-triangle* **and** *c* **by**(*auto simp add*: *dist-triangle*)
also have ... ≤ (*1* − *c*) * (*dist* (*z m*) (*z* (*m* + *k*)) + *c* ^ (*m* + *k*) * *d*)
using *cf-z*[*of m* + *k*] **and** *c* **by** *auto*
also have ... ≤ *c* ^ *m* * *d* * (*1* − *c* ^ *k*) + (*1* − *c*) * *c* ^ (*m* + *k*) * *d*
using *Suc* **by** (*auto simp add*: *field-simps*)
also have ... = (*c* ^ *m*) * (*d* * (*1* − *c* ^ *k*) + (*1* − *c*) * *c* ^ *k* * *d*)
unfolding *power-add* **by** (*auto simp add*: *field-simps*)
also have ... ≤ (*c* ^ *m*) * *d* * (*1* − *c* ^ *Suc k*)
using *c* **by** (*auto simp add*: *field-simps*)
finally show *?case* **by** *auto*
qed
} note *cf-z2* = *this*
{ fix *e*::*real* **assume** *e*>*0*
hence ∃ *N*. ∀ *m n*. *N* ≤ *m* ∧ *N* ≤ *n* ⟶ *dist* (*z m*) (*z n*) < *e*

```
  proof(cases d = 0)
   case True
  hence ⋀n. z n = z0 using cf-z2[of 0] and c unfolding z-def by (auto simp
add: pos-prod-le[OF ‹1 − c > 0›])
   thus ?thesis using ‹e>0› by auto
  next
   case False hence d>0 unfolding d-def using zero-le-dist[of z 0 z 1]
    by (metis False d-def less-le)
   hence 0 < e * (1 − c) / d using ‹e>0› and ‹1−c>0›
    using divide-pos-pos[of e * (1 − c) d] and mult-pos-pos[of e 1 − c] by auto
  then obtain N where N:c ^ N < e * (1 − c) / d using real-arch-pow-inv[of
e * (1 − c) / d c] and c by auto
   { fix m n::nat assume m>n and as:m≥N n≥N
     have *:c ^ n ≤ c ^ N using ‹n≥N› and c using power-decreasing[OF
‹n≥N›, of c] by auto
    have 1 − c ^ (m − n) > 0 using c and power-strict-mono[of c 1 m − n]
using ‹m>n› by auto
    hence **:d * (1 − c ^ (m − n)) / (1 − c) > 0
     using mult-pos-pos[OF ‹d>0›, of 1 − c ^ (m − n)]
     using divide-pos-pos[of d * (1 − c ^ (m − n)) 1 − c]
     using ‹0 < 1 − c› by auto

    have dist (z m) (z n) ≤ c ^ n * d * (1 − c ^ (m − n)) / (1 − c)
        using cf-z2[of n m − n] and ‹m>n› unfolding pos-le-divide-eq[OF
‹1−c>0›]
     by (auto simp add: mult-commute dist-commute)
    also have ... ≤ c ^ N * d * (1 − c ^ (m − n)) / (1 − c)
     using mult-right-mono[OF * order-less-imp-le[OF **]]
     unfolding mult-assoc by auto
    also have ... < (e * (1 − c) / d) * d * (1 − c ^ (m − n)) / (1 − c)
     using mult-strict-right-mono[OF N **] unfolding mult-assoc by auto
   also have ... = e * (1 − c ^ (m − n)) using c and ‹d>0› and ‹1 − c >
0› by auto
    also have ... ≤ e using c and ‹1 − c ^ (m − n) > 0› and ‹e>0› using
mult-right-le-one-le[of e 1 − c ^ (m − n)] by auto
    finally have dist (z m) (z n) < e by auto
   } note * = this
   { fix m n::nat assume as:N≤m N≤n
    hence dist (z n) (z m) < e
    proof(cases n = m)
     case True thus ?thesis using ‹e>0› by auto
    next
      case False thus ?thesis using as and *[of n m] *[of m n] unfolding
nat-neq-iff by (auto simp add: dist-commute)
    qed }
   thus ?thesis by auto
  qed
 }
 hence Cauchy z unfolding cauchy-def by auto
```

then obtain x **where** $x{\in}s$ **and** $x{:}(z \longrightarrow x)$ *sequentially* **using** $s(1)$[*unfolded compact-def complete-def*, *THEN spec*[**where** $x{=}z$]] **and** *z-in-s* **by** *auto*

def $e \equiv$ *dist* $(f\ x)\ x$
have $e = 0$ **proof**(*rule ccontr*)
assume $e \neq 0$ **hence** $e{>}0$ **unfolding** *e-def* **using** *zero-le-dist*[*of f x x*]
by (*metis dist-eq-0-iff dist-nz e-def*)
then obtain N **where** $N{:}\forall\, n{\geq}N.$ *dist* $(z\ n)\ x < e\ /\ 2$
using x[*unfolded Lim-sequentially*, *THEN spec*[**where** $x{=}e/2$]] **by** *auto*
hence $N'{:}$*dist* $(z\ N)\ x < e\ /\ 2$ **by** *auto*

have $*{:}c * dist\ (z\ N)\ x \leq dist\ (z\ N)\ x$ **unfolding** *mult-le-cancel-right2*
using *zero-le-dist*[*of z N x*] **and** c
by (*metis dist-eq-0-iff dist-nz order-less-asym less-le*)
have *dist* $(f\ (z\ N))\ (f\ x) \leq c * dist\ (z\ N)\ x$ **using** *lipschitz*[*THEN bspec*[**where** $x{=}z\ N$], *THEN bspec*[**where** $x{=}x$]]
using *z-in-s*[*of N*] ‹$x{\in}s$› **using** c **by** *auto*
also have $\ldots < e\ /\ 2$ **using** N' **and** c **using** $*$ **by** *auto*
finally show *False* **unfolding** *fzn*
using N[*THEN spec*[**where** $x{=}Suc\ N$]] **and** *dist-triangle-half-r*[*of z* (*Suc N*) *f x e x*]
unfolding *e-def* **by** *auto*
qed
hence $f\ x = x$ **unfolding** *e-def* **by** *auto*
moreover
{ fix y **assume** $f\ y = y\ y{\in}s$
hence *dist x y* $\leq c *$ *dist x y* **using** *lipschitz*[*THEN bspec*[**where** $x{=}x$], *THEN bspec*[**where** $x{=}y$]]
using ‹$x{\in}s$› **and** ‹$f\ x = x$› **by** *auto*
hence *dist x y* $= 0$ **unfolding** *mult-le-cancel-right1*
using c **and** *zero-le-dist*[*of x y*] **by** *auto*
hence $y = x$ **by** *auto*
}
ultimately show *?thesis* **using** ‹$x{\in}s$› **by** *blast+*
qed

16.30 Edelstein fixed point theorem.

lemma *edelstein-fix*:
fixes s :: *'a*::*real-normed-vector set*
assumes s:*compact s* $s \neq \{\}$ **and** $gs{:}(g\ `\ s) \subseteq s$
and *dist*:$\forall\, x{\in}s.\ \forall\, y{\in}s.\ x \neq y \longrightarrow dist\ (g\ x)\ (g\ y) < dist\ x\ y$
shows $\exists!\ x{\in}s.\ g\ x = x$
proof(*cases* $\exists\, x{\in}s.\ g\ x \neq x$)
obtain x **where** $x{\in}s$ **using** $s(2)$ **by** *auto*
case *False* **hence** $g{:}\forall\, x{\in}s.\ g\ x = x$ **by** *auto*
{ fix y **assume** $y{\in}s$
hence $x = y$ **using** ‹$x{\in}s$› **and** *dist*[*THEN bspec*[**where** $x{=}x$], *THEN bspec*[**where** $x{=}y$]]

```
    unfolding g[THEN bspec[where x=x], OF ‹x∈s›]
    unfolding g[THEN bspec[where x=y], OF ‹y∈s›] by auto }
 thus ?thesis using ‹x∈s› and g by blast+
next
 case True
 then obtain x where [simp]:x∈s and g x ≠ x by auto
 { fix x y assume x ∈ s y ∈ s
  hence dist (g x) (g y) ≤ dist x y
    using dist[THEN bspec[where x=x], THEN bspec[where x=y]] by auto }
note dist' = this
 def y ≡ g x
 have [simp]:y∈s unfolding y-def using gs[unfolded image-subset-iff] and ‹x∈s›
by blast
 def f ≡ λn. g ^^ n
 have [simp]:⋀n z. g (f n z) = f (Suc n) z unfolding f-def by auto
 have [simp]:⋀z. f 0 z = z unfolding f-def by auto
 { fix n::nat and z assume z∈s
  have f n z ∈ s unfolding f-def
  proof(induct n)
   case 0 thus ?case using ‹z∈s› by simp
  next
   case (Suc n) thus ?case using gs[unfolded image-subset-iff] by auto
  qed } note fs = this
 { fix m n ::nat assume m≤n
  fix w z assume w∈s z∈s
  have dist (f n w) (f n z) ≤ dist (f m w) (f m z) using ‹m≤n›
  proof(induct n)
   case 0 thus ?case by auto
  next
   case (Suc n)
   thus ?case proof(cases m≤n)
    case True thus ?thesis using Suc(1)
     using dist'[OF fs fs, OF ‹w∈s› ‹z∈s›, of n n] by auto
   next
    case False hence mn:m = Suc n using Suc(2) by simp
    show ?thesis unfolding mn by auto
   qed
  qed } note distf = this

 def h ≡ λn. (f n x, f n y)
 let ?s2 = s × s
 obtain l r where l∈?s2 and r:subseq r and lr:((h ∘ r) ---> l) sequentially
  using compact-Times [OF s(1) s(1), unfolded compact-def, THEN spec[where
x=h]] unfolding h-def
  using fs[OF ‹x∈s›] and fs[OF ‹y∈s›] by blast
 def a ≡ fst l def b ≡ snd l
 have lab:l = (a, b) unfolding a-def b-def by simp
 have [simp]:a∈s b∈s unfolding a-def b-def using ‹l∈?s2› by auto
```

have *lima*:((*fst* ∘ (*h* ∘ *r*)) −−−> *a*) *sequentially*
and *limb*:((*snd* ∘ (*h* ∘ *r*)) −−−> *b*) *sequentially*
using *lr*
unfolding *o-def a-def b-def* **by** (*simp-all add*: *tendsto-intros*)

{ fix *n*::*nat*
have ∗:⋀*fx fy* (*x*::′*a*) *y*. *dist fx fy* ≤ *dist x y* ⟹ ¬ (*dist* (*fx* − *fy*) (*a* − *b*) < *dist a b* − *dist x y*) **unfolding** *dist-norm* **by** *norm*
{ fix *x y* :: ′*a*
have *dist* (−*x*) (−*y*) = *dist x y* **unfolding** *dist-norm*
using *norm-minus-cancel*[*of x* − *y*] **by** (*auto simp add*: *uminus-add-conv-diff*)
} **note** ∗∗ = *this*

{ assume *as*:*dist a b* > *dist* (*f n x*) (*f n y*)
then obtain *Na Nb* **where** ∀ *m*≥*Na*. *dist* (*f* (*r m*) *x*) *a* < (*dist a b* − *dist* (*f n x*) (*f n y*)) / *2*
and ∀ *m*≥*Nb*. *dist* (*f* (*r m*) *y*) *b* < (*dist a b* − *dist* (*f n x*) (*f n y*)) / *2*
using *lima limb* **unfolding** *h-def Lim-sequentially* **by** (*fastsimp simp del*: *less-divide-eq-number-of1*)
hence *dist* (*f* (*r* (*Na* + *Nb* + *n*)) *x* − *f* (*r* (*Na* + *Nb* + *n*)) *y*) (*a* − *b*) < *dist a b* − *dist* (*f n x*) (*f n y*)
apply(*erule-tac x*=*Na*+*Nb*+*n* **in** *allE*)
apply(*erule-tac x*=*Na*+*Nb*+*n* **in** *allE*) **apply** *simp*
using *dist-triangle-add-half*[*of a f* (*r* (*Na* + *Nb* + *n*)) *x dist a b* − *dist* (*f n x*) (*f n y*)
−*b* − *f* (*r* (*Na* + *Nb* + *n*)) *y*]
unfolding ∗∗ **by** (*auto simp add*: *algebra-simps dist-commute*)
moreover
have *dist* (*f* (*r* (*Na* + *Nb* + *n*)) *x* − *f* (*r* (*Na* + *Nb* + *n*)) *y*) (*a* − *b*) ≥ *dist a b* − *dist* (*f n x*) (*f n y*)
using *distf*[*of n r* (*Na*+*Nb*+*n*), *OF* - ‹*x*∈*s*› ‹*y*∈*s*›]
using *subseq-bigger*[*OF r*, *of Na*+*Nb*+*n*]
using ∗[*of f* (*r* (*Na* + *Nb* + *n*)) *x f* (*r* (*Na* + *Nb* + *n*)) *y f n x f n y*] **by** *auto*
ultimately have *False* **by** *simp*
}
hence *dist a b* ≤ *dist* (*f n x*) (*f n y*) **by**(*rule ccontr*)*auto* **}**
note *ab-fn* = *this*

have [*simp*]:*a* = *b* **proof**(*rule ccontr*)
def *e* ≡ *dist a b* − *dist* (*g a*) (*g b*)
assume *a*≠*b* **hence** *e* > *0* **unfolding** *e-def* **using** *dist* **by** *fastsimp*
hence ∃ *n*. *dist* (*f n x*) *a* < *e*/*2* ∧ *dist* (*f n y*) *b* < *e*/*2*
using *lima limb* **unfolding** *Lim-sequentially*
apply (*auto elim*!: *allE*[**where** *x*=*e*/*2*]) **apply**(*rule-tac x*=*r* (*max N Na*) **in** *exI*) **unfolding** *h-def* **by** *fastsimp*
then obtain *n* **where** *n*:*dist* (*f n x*) *a* < *e*/*2* ∧ *dist* (*f n y*) *b* < *e*/*2* **by** *auto*
have *dist* (*f* (*Suc n*) *x*) (*g a*) ≤ *dist* (*f n x*) *a*
using *dist*[*THEN bspec*[**where** *x*=*f n x*], *THEN bspec*[**where** *x*=*a*]] **and** *fs*

by *auto*
moreover have *dist* (*f* (*Suc n*) *y*) (*g b*) ≤ *dist* (*f n y*) *b*
using *dist*[*THEN bspec*[**where** *x*=*f n y*], *THEN bspec*[**where** *x*=*b*]] **and** *fs*
by *auto*
ultimately have *dist* (*f* (*Suc n*) *x*) (*g a*) + *dist* (*f* (*Suc n*) *y*) (*g b*) < *e* **using** *n* **by** *auto*
thus *False* **unfolding** *e-def* **using** *ab-fn*[*of Suc n*] **by** *norm*
qed

have [*simp*]:⋀*n*. *f* (*Suc n*) *x* = *f n y* **unfolding** *f-def y-def* **by**(*induct-tac n*)*auto*
{ fix *x y* **assume** *x*∈*s y*∈*s* **moreover**
fix *e*::*real* **assume** *e*>*0* **ultimately**
have *dist y x* < *e* ⟶ *dist* (*g y*) (*g x*) < *e* **using** *dist* **by** *fastsimp* **}**
hence *continuous-on s g* **unfolding** *continuous-on-iff* **by** *auto*

hence ((*snd* ∘ *h* ∘ *r*) ---> *g a*) *sequentially* **unfolding** *continuous-on-sequentially*
apply (*rule allE*[**where** *x*=λ*n*. (*fst* ∘ *h* ∘ *r*) *n*]) **apply** (*erule ballE*[**where** *x*=*a*])
using *lima* **unfolding** *h-def o-def* **using** *fs*[*OF* ‹*x*∈*s*›] **by** (*auto simp add*: *y-def*)
hence *g a* = *a* **using** *Lim-unique*[*OF trivial-limit-sequentially limb*, *of g a*]
unfolding ‹*a*=*b*› **and** *o-assoc* **by** *auto*
moreover
{ fix *x* **assume** *x*∈*s g x* = *x x*≠*a*
hence *False* **using** *dist*[*THEN bspec*[**where** *x*=*a*], *THEN bspec*[**where** *x*=*x*]]
using ‹*g a* = *a*› **and** ‹*a*∈*s*› **by** *auto* **}**
ultimately show ∃!*x*∈*s*. *g x* = *x* **using** ‹*a*∈*s*› **by** *blast*
qed

end

17 Vec1: Vectors of size 1, 2, or 3

theory *Vec1*
imports *Topology-Euclidean-Space*
begin

Some common special cases.

lemma *forall-1*[*simp*]: (∀ *i*::*1*. *P i*) ⟷ *P 1*
by (*metis num1-eq-iff*)

lemma *ex-1*[*simp*]: (∃ *x*::*1*. *P x*) ⟷ *P 1*
by *auto* (*metis num1-eq-iff*)

lemma *exhaust-2*:
fixes *x* :: *2* **shows** *x* = *1* ∨ *x* = *2*
proof (*induct x*)
case (*of-int z*)

then have *0 <= z* **and** *z < 2* **by** *simp-all*
then have *z = 0 | z = 1* **by** *arith*
then show *?case* **by** *auto*
qed

lemma *forall-2*: $(\forall i{::}2.\ P\ i) \longleftrightarrow P\ 1 \wedge P\ 2$
by (*metis exhaust-2*)

lemma *exhaust-3*:
fixes *x :: 3* **shows** $x = 1 \vee x = 2 \vee x = 3$
proof (*induct x*)
case (*of-int z*)
then have *0 <= z* **and** *z < 3* **by** *simp-all*
then have $z = 0 \vee z = 1 \vee z = 2$ **by** *arith*
then show *?case* **by** *auto*
qed

lemma *forall-3*: $(\forall i{::}3.\ P\ i) \longleftrightarrow P\ 1 \wedge P\ 2 \wedge P\ 3$
by (*metis exhaust-3*)

lemma *UNIV-1* [*simp*]: $UNIV = \{1{::}1\}$
by (*auto simp add*: *num1-eq-iff*)

lemma *UNIV-2*: $UNIV = \{1{::}2, 2{::}2\}$
using *exhaust-2* **by** *auto*

lemma *UNIV-3*: $UNIV = \{1{::}3, 2{::}3, 3{::}3\}$
using *exhaust-3* **by** *auto*

lemma *setsum-1*: $setsum\ f\ (UNIV{::}1\ set) = f\ 1$
unfolding *UNIV-1* **by** *simp*

lemma *setsum-2*: $setsum\ f\ (UNIV{::}2\ set) = f\ 1 + f\ 2$
unfolding *UNIV-2* **by** *simp*

lemma *setsum-3*: $setsum\ f\ (UNIV{::}3\ set) = f\ 1 + f\ 2 + f\ 3$
unfolding *UNIV-3* **by** (*simp add*: *add-ac*)

instantiation *num1* :: *cart-one* **begin**
instance proof
show $CARD(1) = Suc\ 0$ **by** *auto*
qed end

abbreviation *vec1*:: $'a \Rightarrow 'a\ \hat{}\ 1$ **where** $vec1\ x \equiv vec\ x$

abbreviation *dest-vec1*:: $'a\ \hat{}1 \Rightarrow 'a$
where $dest\text{-}vec1\ x \equiv (x\$1)$

lemma *vec1-component*[*simp*]: (*vec1 x*)$*1* = *x*
by *simp*

lemma *vec1-dest-vec1*: *vec1*(*dest-vec1 x*) = *x dest-vec1*(*vec1 y*) = *y*
by (*simp-all add*: *Cart-eq*)

declare *vec1-dest-vec1*(*1*) [*simp*]

lemma *forall-vec1*: (∀ *x*. *P x*) ⟷ (∀ *x*. *P* (*vec1 x*))
by (*metis vec1-dest-vec1*(*1*))

lemma *exists-vec1*: (∃ *x*. *P x*) ⟷ (∃ *x*. *P*(*vec1 x*))
by (*metis vec1-dest-vec1*(*1*))

lemma *vec1-eq*[*simp*]: *vec1 x* = *vec1 y* ⟷ *x* = *y*
by (*metis vec1-dest-vec1*(*2*))

lemma *dest-vec1-eq*[*simp*]: *dest-vec1 x* = *dest-vec1 y* ⟷ *x* = *y*
by (*metis vec1-dest-vec1*(*1*))

17.1 The collapse of the general concepts to dimension one.

lemma *vector-one*: (*x*::′*a* ^*1*) = (χ *i*. (*x*$*1*))
by (*simp add*: *Cart-eq*)

lemma *forall-one*: (∀ (*x*::′*a* ^*1*). *P x*) ⟷ (∀ *x*. *P*(χ *i*. *x*))
apply *auto*
apply (*erule-tac x*= *x*$*1* **in** *allE*)
apply (*simp only*: *vector-one*[*symmetric*])
done

lemma *norm-vector-1*: *norm* (*x* :: -^*1*) = *norm* (*x*$*1*)
by (*simp add*: *norm-vector-def*)

lemma *norm-real*: *norm*(*x*::*real* ^ *1*) = *abs*(*x*$*1*)
by (*simp add*: *norm-vector-1*)

lemma *dist-real*: *dist*(*x*::*real* ^ *1*) *y* = *abs*((*x*$*1*) − (*y*$*1*))
by (*auto simp add*: *norm-real dist-norm*)

17.2 Explicit vector construction from lists.

primrec *from-nat* :: *nat* ⇒ ′*a*::{*monoid-add*,*one*}
where *from-nat 0* = *0* | *from-nat* (*Suc n*) = *1* + *from-nat n*

lemma *from-nat* [*simp*]: *from-nat* = *of-nat*
by (*rule ext*, *induct-tac x*, *simp-all*)

primrec

list-fun :: *nat* ⇒ - *list* ⇒ - ⇒ -
where
list-fun n [] = (λ*x*. *0*)
| *list-fun n* (*x* # *xs*) = *fun-upd* (*list-fun* (*Suc n*) *xs*) (*from-nat n*) *x*

definition *vector l* = (χ *i*. *list-fun 1 l i*)

lemma *vector-1*: (*vector*[*x*]) $*1* = *x*
unfolding *vector-def* **by** *simp*

lemma *vector-2*:
(*vector*[*x*,*y*]) $*1* = *x*
(*vector*[*x*,*y*] :: *'a^2*)$*2* = (*y*::*'a*::*zero*)
unfolding *vector-def* **by** *simp-all*

lemma *vector-3*:
(*vector* [*x*,*y*,*z*] ::(*'a*::*zero*) *^3*)$*1* = *x*
(*vector* [*x*,*y*,*z*] ::(*'a*::*zero*) *^3*)$*2* = *y*
(*vector* [*x*,*y*,*z*] ::(*'a*::*zero*) *^3*)$*3* = *z*
unfolding *vector-def* **by** *simp-all*

lemma *forall-vector-1*: (∀ *v*::*'a*::*zero^1*. *P v*) ⟷ (∀ *x*. *P*(*vector*[*x*]))
apply *auto*
apply (*erule-tac x*=*v*$*1* **in** *allE*)
apply (*subgoal-tac vector* [*v*$*1*] = *v*)
apply *simp*
apply (*vector vector-def*)
apply *simp*
done

lemma *forall-vector-2*: (∀ *v*::*'a*::*zero^2*. *P v*) ⟷ (∀ *x y*. *P*(*vector*[*x*, *y*]))
apply *auto*
apply (*erule-tac x*=*v*$*1* **in** *allE*)
apply (*erule-tac x*=*v*$*2* **in** *allE*)
apply (*subgoal-tac vector* [*v*$*1*, *v*$*2*] = *v*)
apply *simp*
apply (*vector vector-def*)
apply (*simp add*: *forall-2*)
done

lemma *forall-vector-3*: (∀ *v*::*'a*::*zero^3*. *P v*) ⟷ (∀ *x y z*. *P*(*vector*[*x*, *y*, *z*]))
apply *auto*
apply (*erule-tac x*=*v*$*1* **in** *allE*)
apply (*erule-tac x*=*v*$*2* **in** *allE*)
apply (*erule-tac x*=*v*$*3* **in** *allE*)
apply (*subgoal-tac vector* [*v*$*1*, *v*$*2*, *v*$*3*] = *v*)
apply *simp*
apply (*vector vector-def*)

apply (*simp add*: *forall-3*)
done

lemma *range-vec1*[*simp*]:*range vec1* = *UNIV* **apply**(*rule set-ext*,*rule*) **unfolding** *image-iff* **defer**
apply(*rule-tac x*=*dest-vec1 x* **in** *bexI*) **by** *auto*

lemma *dest-vec1-lambda*: *dest-vec1*(*χ i. x i*) = *x 1*
by (*simp*)

lemma *dest-vec1-vec*: *dest-vec1*(*vec x*) = *x*
by (*simp*)

lemma *dest-vec1-sum*: **assumes** *fS*: *finite S*
shows *dest-vec1*(*setsum f S*) = *setsum* (*dest-vec1 o f*) *S*
apply (*induct rule*: *finite-induct*[*OF fS*])
apply *simp*
apply *auto*
done

lemma *norm-vec1* [*simp*]: *norm*(*vec1 x*) = *abs*(*x*)
by (*simp add*: *vec-def norm-real*)

lemma *dist-vec1*: *dist*(*vec1 x*) (*vec1 y*) = *abs*(*x* − *y*)
by (*simp only*: *dist-real vec1-component*)
lemma *abs-dest-vec1*: *norm x* = |*dest-vec1 x*|
by (*metis vec1-dest-vec1*(*1*) *norm-vec1*)

lemmas *vec1-dest-vec1-simps* = *forall-vec1 vec-add*[*THEN sym*] *dist-vec1 vec-sub*[*THEN sym*] *vec1-dest-vec1 norm-vec1 vector-smult-component*
vec1-eq vec-cmul[*THEN sym*] *smult-conv-scaleR*[*THEN sym*] *o-def dist-real-def norm-vec1 real-norm-def*

lemma *bounded-linear-vec1*:*bounded-linear* (*vec1*::*real*⇒*real^1*)
unfolding *bounded-linear-def additive-def bounded-linear-axioms-def*
unfolding *smult-conv-scaleR*[*THEN sym*] **unfolding** *vec1-dest-vec1-simps*
apply(*rule conjI*) **defer apply**(*rule conjI*) **defer apply**(*rule-tac x=1* **in** *exI*)
by *auto*

lemma *linear-vmul-dest-vec1*:
fixes *f*:: *real^- ⇒ real^1*
shows *linear f* ⟹ *linear* (*λx. dest-vec1*(*f x*) ∗*s v*)
unfolding *smult-conv-scaleR*
by (*rule linear-vmul-component*)

lemma *linear-from-scalars*:
assumes *lf*: *linear* (*f*::*real^1 ⇒ real^-*)
shows *f* = (*λx. dest-vec1 x* ∗*s column 1* (*matrix f*))
unfolding *smult-conv-scaleR*

apply (*rule ext*)
apply (*subst matrix-works*[*OF lf*, *symmetric*])
apply (*auto simp add*: *Cart-eq matrix-vector-mult-def column-def mult-commute*)
done

lemma *linear-to-scalars*: **assumes** *lf*: *linear* (*f*::*real* ^'*n* ⇒ *real*^*1*)
shows *f* = (λ*x*. *vec1*(*row 1* (*matrix f*) · *x*))
apply (*rule ext*)
apply (*subst matrix-works*[*OF lf*, *symmetric*])
apply (*simp add*: *Cart-eq matrix-vector-mult-def row-def inner-vector-def mult-commute*)
done

lemma *dest-vec1-eq-0*: *dest-vec1 x* = *0* ⟷ *x* = *0*
by (*simp add*: *dest-vec1-eq*[*symmetric*])

lemma *setsum-scalars*: **assumes** *fS*: *finite S*
shows *setsum f S* = *vec1* (*setsum* (*dest-vec1 o f*) *S*)
unfolding *vec-setsum*[*OF fS*] **by** *simp*

lemma *dest-vec1-wlog-le*: (⋀(*x*::'*a*::*linorder* ^ *1*) *y*. *P x y* ⟷ *P y x*) ⟹ (⋀*x y*. *dest-vec1 x* <= *dest-vec1 y* ==> *P x y*) ⟹ *P x y*
apply (*cases dest-vec1 x* ≤ *dest-vec1 y*)
apply *simp*
apply (*subgoal-tac dest-vec1 y* ≤ *dest-vec1 x*)
apply (*auto*)
done

Lifting and dropping

lemma *continuous-on-o-dest-vec1*: **fixes** *f*::*real* ⇒ '*a*::*real-normed-vector*
assumes *continuous-on* {*a*..*b*::*real*} *f* **shows** *continuous-on* {*vec1 a*..*vec1 b*} (*f o dest-vec1*)
using *assms* **unfolding** *continuous-on-iff* **apply** *safe*
apply(*erule-tac x*=*x*$*1* **in** *ballE*,*erule-tac x*=*e* **in** *allE*) **apply** *safe*
apply(*rule-tac x*=*d* **in** *exI*) **apply** *safe* **unfolding** *o-def dist-real-def dist-real*
apply(*erule-tac x*=*dest-vec1 x'* **in** *ballE*) **by**(*auto simp add*:*vector-le-def*)

lemma *continuous-on-o-vec1*: **fixes** *f*::*real*^*1* ⇒ '*a*::*real-normed-vector*
assumes *continuous-on* {*a*..*b*} *f* **shows** *continuous-on* {*dest-vec1 a*..*dest-vec1 b*} (*f o vec1*)
using *assms* **unfolding** *continuous-on-iff* **apply** *safe*
apply(*erule-tac x*=*vec x* **in** *ballE*,*erule-tac x*=*e* **in** *allE*) **apply** *safe*
apply(*rule-tac x*=*d* **in** *exI*) **apply** *safe* **unfolding** *o-def dist-real-def dist-real*
apply(*erule-tac x*=*vec1 x'* **in** *ballE*) **by**(*auto simp add*:*vector-le-def*)

lemma *continuous-on-vec1*:*continuous-on A* (*vec1*::*real*⇒*real*^*1*)
by(*rule linear-continuous-on*[*OF bounded-linear-vec1*])

lemma *mem-interval-1*: **fixes** *x* :: *real*^*1* **shows**
(*x* ∈ {*a* .. *b*} ⟷ *dest-vec1 a* ≤ *dest-vec1 x* ∧ *dest-vec1 x* ≤ *dest-vec1 b*)

$(x \in \{a<..<b\} \longleftrightarrow$ *dest-vec1* $a <$ *dest-vec1* $x \wedge$ *dest-vec1* $x <$ *dest-vec1* $b)$
by(*simp-all add*: *Cart-eq vector-less-def vector-le-def*)

lemma *vec1-interval*:**fixes** a::*real* **shows**
vec1 ‘ $\{a .. b\} = \{$*vec1* $a ..$ *vec1* $b\}$
vec1 ‘ $\{a<..<b\} = \{$*vec1* $a<..<$*vec1* $b\}$
apply(*rule-tac*[!] *set-ext*) **unfolding** *image-iff vector-less-def* **unfolding** *mem-interval*
unfolding *forall-1* **unfolding** *vec1-dest-vec1-simps*
apply *rule* **defer apply**(*rule-tac* x=*dest-vec1* x **in** *bexI*) **prefer** *3* **apply** *rule*
defer
apply(*rule-tac* x=*dest-vec1* x **in** *bexI*) **by** *auto*

lemma *interval-cases-1*: **fixes** x :: *real^1* **shows**
$x \in \{a .. b\} ==> x \in \{a<..<b\} \vee (x = a) \vee (x = b)$
unfolding *Cart-eq vector-less-def vector-le-def mem-interval* **by**(*auto simp del*:*dest-vec1-eq*)

lemma *in-interval-1*: **fixes** x :: *real^1* **shows**
$(x \in \{a .. b\} \longleftrightarrow$ *dest-vec1* $a \leq$ *dest-vec1* $x \wedge$ *dest-vec1* $x \leq$ *dest-vec1* $b) \wedge$
$(x \in \{a<..<b\} \longleftrightarrow$ *dest-vec1* $a <$ *dest-vec1* $x \wedge$ *dest-vec1* $x <$ *dest-vec1* $b)$
unfolding *Cart-eq vector-less-def vector-le-def mem-interval* **by**(*auto simp del*:*dest-vec1-eq*)

lemma *interval-eq-empty-1*: **fixes** a :: *real^1* **shows**
$\{a .. b\} = \{\} \longleftrightarrow$ *dest-vec1* $b <$ *dest-vec1* a
$\{a<..<b\} = \{\} \longleftrightarrow$ *dest-vec1* $b \leq$ *dest-vec1* a
unfolding *interval-eq-empty* **and** *ex-1* **by** *auto*

lemma *subset-interval-1*: **fixes** a :: *real^1* **shows**
$(\{a .. b\} \subseteq \{c .. d\} \longleftrightarrow$ *dest-vec1* $b <$ *dest-vec1* $a \vee$
dest-vec1 $c \leq$ *dest-vec1* $a \wedge$ *dest-vec1* $a \leq$ *dest-vec1* $b \wedge$ *dest-vec1* b
$\leq$ *dest-vec1* $d)$
$(\{a .. b\} \subseteq \{c<..<d\} \longleftrightarrow$ *dest-vec1* $b <$ *dest-vec1* $a \vee$
dest-vec1 $c <$ *dest-vec1* $a \wedge$ *dest-vec1* $a \leq$ *dest-vec1* $b \wedge$ *dest-vec1* b
$<$ *dest-vec1* $d)$
$(\{a<..<b\} \subseteq \{c .. d\} \longleftrightarrow$ *dest-vec1* $b \leq$ *dest-vec1* $a \vee$
dest-vec1 $c \leq$ *dest-vec1* $a \wedge$ *dest-vec1* $a <$ *dest-vec1* $b \wedge$ *dest-vec1* b
$\leq$ *dest-vec1* $d)$
$(\{a<..<b\} \subseteq \{c<..<d\} \longleftrightarrow$ *dest-vec1* $b \leq$ *dest-vec1* $a \vee$
dest-vec1 $c \leq$ *dest-vec1* $a \wedge$ *dest-vec1* $a <$ *dest-vec1* $b \wedge$ *dest-vec1* b
$\leq$ *dest-vec1* $d)$
unfolding *subset-interval*[*of* a b c d] **unfolding** *forall-1* **by** *auto*

lemma *eq-interval-1*: **fixes** a :: *real^1* **shows**
$\{a .. b\} = \{c .. d\} \longleftrightarrow$
dest-vec1 $b <$ *dest-vec1* $a \wedge$ *dest-vec1* $d <$ *dest-vec1* $c \vee$
dest-vec1 $a =$ *dest-vec1* $c \wedge$ *dest-vec1* $b =$ *dest-vec1* d
unfolding *set-eq-subset*[*of* $\{a .. b\}$ $\{c .. d\}$]
unfolding *subset-interval-1*(*1*)[*of* a b c d]

unfolding *subset-interval-1(1)*[*of c d a b*]
by *auto*

lemma *disjoint-interval-1*: **fixes** *a* :: *real^1* **shows**
$\{a .. b\} \cap \{c .. d\} = \{\} \longleftrightarrow$ *dest-vec1 b* $<$ *dest-vec1 a* $\vee$ *dest-vec1 d* $<$ *dest-vec1 c* $\vee$ *dest-vec1 b* $<$ *dest-vec1 c* $\vee$ *dest-vec1 d* $<$ *dest-vec1 a*
$\{a .. b\} \cap \{c<..<d\} = \{\} \longleftrightarrow$ *dest-vec1 b* $<$ *dest-vec1 a* $\vee$ *dest-vec1 d* $\leq$ *dest-vec1 c* $\vee$ *dest-vec1 b* $\leq$ *dest-vec1 c* $\vee$ *dest-vec1 d* $\leq$ *dest-vec1 a*
$\{a<..<b\} \cap \{c .. d\} = \{\} \longleftrightarrow$ *dest-vec1 b* $\leq$ *dest-vec1 a* $\vee$ *dest-vec1 d* $<$ *dest-vec1 c* $\vee$ *dest-vec1 b* $\leq$ *dest-vec1 c* $\vee$ *dest-vec1 d* $\leq$ *dest-vec1 a*
$\{a<..<b\} \cap \{c<..<d\} = \{\} \longleftrightarrow$ *dest-vec1 b* $\leq$ *dest-vec1 a* $\vee$ *dest-vec1 d* $\leq$ *dest-vec1 c* $\vee$ *dest-vec1 b* $\leq$ *dest-vec1 c* $\vee$ *dest-vec1 d* $\leq$ *dest-vec1 a*
unfolding *disjoint-interval* **and** *ex-1* **by** *auto*

lemma *open-closed-interval-1*: **fixes** *a* :: *real^1* **shows**
$\{a<..<b\} = \{a .. b\} - \{a, b\}$
unfolding *expand-set-eq* **apply** *simp* **unfolding** *vector-less-def* **and** *vector-le-def* **and** *forall-1* **and** *dest-vec1-eq*[*THEN sym*] **by**(*auto simp del*:*dest-vec1-eq*)

lemma *closed-open-interval-1*: *dest-vec1* (*a*::*real^1*) $\leq$ *dest-vec1 b* ==> $\{a .. b\} = \{a<..<b\} \cup \{a,b\}$
unfolding *expand-set-eq* **apply** *simp* **unfolding** *vector-less-def* **and** *vector-le-def* **and** *forall-1* **and** *dest-vec1-eq*[*THEN sym*] **by**(*auto simp del*:*dest-vec1-eq*)

lemma *Lim-drop-le*: **fixes** *f* :: *'a* $\Rightarrow$ *real^1* **shows**
(*f* ---> *l*) *net* $\Longrightarrow$ ~(*trivial-limit net*) $\Longrightarrow$ *eventually* (λx. *dest-vec1* (*f x*) $\leq$ *b*) *net* ==> *dest-vec1 l* $\leq$ *b*
using *Lim-component-le*[*of f l net 1 b*] **by** *auto*

lemma *Lim-drop-ge*: **fixes** *f* :: *'a* $\Rightarrow$ *real^1* **shows**
(*f* ---> *l*) *net* $\Longrightarrow$ ~(*trivial-limit net*) $\Longrightarrow$ *eventually* (λx. *b* $\leq$ *dest-vec1* (*f x*)) *net* ==> *b* $\leq$ *dest-vec1 l*
using *Lim-component-ge*[*of f l net b 1*] **by** *auto*

Also more convenient formulations of monotone convergence.

lemma *bounded-increasing-convergent*: **fixes** *s*::*nat* $\Rightarrow$ *real^1*
assumes *bounded* $\{s\ n|\ n$::*nat*. *True*$\}$ $\forall n$. *dest-vec1*(*s n*) $\leq$ *dest-vec1*(*s*(*Suc n*))
shows $\exists l$. (*s* ---> *l*) *sequentially*
proof−
obtain *a* **where** *a*:$\forall n$. |*dest-vec1* (*s n*)| $\leq$ *a* **using** *assms(1)*[*unfolded bounded-iff abs-dest-vec1*] **by** *auto*
{ fix *m*::*nat*
have $\bigwedge n$. *n*$\geq$*m* $\longrightarrow$ *dest-vec1* (*s m*) $\leq$ *dest-vec1* (*s n*)
apply(*induct-tac n*) **apply** *simp* **using** *assms(2)* **apply**(*erule-tac x*=*na* **in** *allE*) **by**(*auto simp add*: *not-less-eq-eq*) **}**
hence $\forall m\ n$. *m* $\leq$ *n* $\longrightarrow$ *dest-vec1* (*s m*) $\leq$ *dest-vec1* (*s n*) **by** *auto*
then obtain *l* **where** $\forall e$>*0*. $\exists N$. $\forall n \geq N$. |*dest-vec1* (*s n*) − *l*| $<$ *e* **using** *convergent-bounded-monotone*[*OF a*] **unfolding** *monoseq-def* **by** *auto*
thus *?thesis* **unfolding** *Lim-sequentially* **apply**(*rule-tac x*=*vec1 l* **in** *exI*)

unfolding *dist-norm* **unfolding** *abs-dest-vec1* **by** *auto*
qed

lemma *dest-vec1-simps*[*simp*]: **fixes** *a*::*real^1*
shows *a$1 = 0 ⟷ a = 0*
a ≤ b ⟷ dest-vec1 a ≤ dest-vec1 b dest-vec1 (1::*real^1) = 1*
by(*auto simp add*: *vector-le-def Cart-eq*)

lemma *dest-vec1-inverval*:
dest-vec1 ‘ {a .. b} = {dest-vec1 a .. dest-vec1 b}
dest-vec1 ‘ {a<.. b} = {dest-vec1 a<.. dest-vec1 b}
dest-vec1 ‘ {a ..<b} = {dest-vec1 a ..<dest-vec1 b}
dest-vec1 ‘ {a<..<b} = {dest-vec1 a<..<dest-vec1 b}
apply(*rule-tac* [!] *equalityI*)
unfolding *subset-eq Ball-def Bex-def mem-interval-1 image-iff*
apply(*rule-tac* [!] *allI*)**apply**(*rule-tac* [!] *impI*)
apply(*rule-tac*[*2*] *x=vec1 x* **in** *exI*)**apply**(*rule-tac*[*4*] *x=vec1 x* **in** *exI*)
apply(*rule-tac*[*6*] *x=vec1 x* **in** *exI*)**apply**(*rule-tac*[*8*] *x=vec1 x* **in** *exI*)
by (*auto simp add*: *vector-less-def vector-le-def*)

lemma *dest-vec1-setsum*: **assumes** *finite S*
shows *dest-vec1 (setsum f S) = setsum (λx. dest-vec1 (f x)) S*
using *dest-vec1-sum*[*OF assms*] **by** *auto*

lemma *open-dest-vec1-vimage*: *open S ⟹ open (dest-vec1 −‘ S)*
unfolding *open-vector-def forall-1* **by** *auto*

lemma *tendsto-dest-vec1* [*tendsto-intros*]:
(f ---> l) net ⟹ ((λx. dest-vec1 (f x)) ---> dest-vec1 l) net
by(*rule tendsto-Cart-nth*)

lemma *continuous-dest-vec1*: *continuous net f ⟹ continuous net (λx. dest-vec1 (f x))*
unfolding *continuous-def* **by** (*rule tendsto-dest-vec1*)

lemma *forall-dest-vec1*: *(∀ x. P x) ⟷ (∀ x. P(dest-vec1 x))*
apply *safe* **defer apply**(*erule-tac x=vec1 x* **in** *allE*) **by** *auto*

lemma *forall-of-dest-vec1*: *(∀ v. P (λx. dest-vec1 (v x))) ⟷ (∀ x. P x)*
apply *rule* **apply** *rule* **apply**(*erule-tac x=(vec1 ∘ x)* **in** *allE*) **unfolding** *o-def vec1-dest-vec1* **by** *auto*

lemma *forall-of-dest-vec1'*: *(∀ v. P (dest-vec1 v)) ⟷ (∀ x. P x)*
apply *rule* **apply** *rule* **apply**(*erule-tac x=(vec1 x)* **in** *allE*) **defer apply** *rule*
apply(*erule-tac x=dest-vec1 v* **in** *allE*) **unfolding** *o-def vec1-dest-vec1* **by** *auto*

lemma *dist-vec1-0*[*simp*]: *dist(vec1 (x*::*real)) 0 = norm x* **unfolding** *dist-norm*
by *auto*

lemma *bounded-linear-vec1-dest-vec1*: **fixes** f::*real* ⇒ *real*
shows *linear* (*vec1* ∘ f ∘ *dest-vec1*) = *bounded-linear* f (**is** *?l* = *?r*) **proof**−
{ assume *?l* **guess** K **using** *linear-bounded*[*OF* ‹*?l*›] **..**
hence ∃ K. ∀ x. |f x| ≤ |x| ∗ K **apply**(*rule-tac* x=K **in** *exI*)
unfolding *vec1-dest-vec1-simps* **by** (*auto simp add*:*field-simps*) **}**
thus *?thesis* **unfolding** *linear-def bounded-linear-def additive-def bounded-linear-axioms-def o-def*
unfolding *vec1-dest-vec1-simps* **by** *auto* **qed**

lemma *vec1-le*[*simp*]:**fixes** a::*real* **shows** *vec1* a ≤ *vec1* b ⟷ a ≤ b
unfolding *vector-le-def* **by** *auto*
lemma *vec1-less*[*simp*]:**fixes** a::*real* **shows** *vec1* a < *vec1* b ⟷ a < b
unfolding *vector-less-def* **by** *auto*

end

18 Determinants: Traces, Determinant of square matrices and some properties

theory *Determinants*
imports *Euclidean-Space Permutations Vec1*
begin

18.1 First some facts about products

lemma *setprod-insert-eq*: *finite* A ⟹ *setprod* f (*insert* a A) = (*if* a ∈ A *then* *setprod* f A *else* f a ∗ *setprod* f A)
apply *clarsimp*
by(*subgoal-tac insert* a A = A, *auto*)

lemma *setprod-add-split*:
assumes *mn*: (*m*::*nat*) <= n + *1*
shows *setprod* f {m.. n+p} = *setprod* f {m .. n} ∗ *setprod* f {n+*1*..n+p}
proof−
let *?A* = {m .. n+p}
let *?B* = {m .. n}
let *?C* = {n+*1*..n+p}
from *mn* **have** *un*: *?B* ∪ *?C* = *?A* **by** *auto*
from *mn* **have** *dj*: *?B* ∩ *?C* = {} **by** *auto*
have f: *finite ?B finite ?C* **by** *simp-all*
from *setprod-Un-disjoint*[*OF f dj*, *of f*, *unfolded un*] **show** *?thesis* **.**
qed

lemma *setprod-offset*: *setprod* f {(*m*::*nat*) + p .. n + p} = *setprod* (λi. f (i + p)) {m..n}
apply (*rule setprod-reindex-cong*[**where** f=*op* + p])

apply (*auto simp add*: *image-iff Bex-def inj-on-def*)
apply *arith*
apply (*rule ext*)
apply (*simp add*: *add-commute*)
done

lemma *setprod-singleton*: *setprod f* {*x*} = *f x* **by** *simp*

lemma *setprod-singleton-nat-seg*: *setprod f* {*n*..*n*} = *f* (*n*::'*a*::*order*) **by** *simp*

lemma *setprod-numseg*: *setprod f* {*m*..*0*} = (*if m*=*0 then f 0 else 1*)
setprod f {*m* .. *Suc n*} = (*if m* ≤ *Suc n then f* (*Suc n*) ∗ *setprod f* {*m*..*n*}
else setprod f {*m*..*n*})
by (*auto simp add*: *atLeastAtMostSuc-conv*)

lemma *setprod-le*: **assumes** *fS*: *finite S* **and** *fg*: ∀ *x*∈*S*. *f x* ≥ *0* ∧ *f x* ≤ (*g x* :: '*a*::*linordered-idom*)
shows *setprod f S* ≤ *setprod g S*
using *fS fg*
apply(*induct S*)
apply *simp*
apply *auto*
apply (*rule mult-mono*)
apply (*auto intro*: *setprod-nonneg*)
done

lemma *setprod-inversef*: *finite A* ==> *setprod* (*inverse* ∘ *f*) *A* = (*inverse* (*setprod f A*) :: '*a*:: *field-inverse-zero*)
apply (*erule finite-induct*)
apply (*simp*)
apply *simp*
done

lemma *setprod-le-1*: **assumes** *fS*: *finite S* **and** *f*: ∀ *x*∈*S*. *f x* ≥ *0* ∧ *f x* ≤ (*1*::'*a*::*linordered-idom*)
shows *setprod f S* ≤ *1*
using *setprod-le*[*OF fS f*] **unfolding** *setprod-1* **.**

18.2 Trace

definition *trace* :: '*a*::*semiring-1*^'*n*^'*n* ⇒ '*a* **where**
trace A = *setsum* (λ*i*. ((*A*$*i*)$*i*)) (*UNIV*::'*n set*)

lemma *trace-0*: *trace*(*mat 0*) = *0*
by (*simp add*: *trace-def mat-def*)

lemma *trace-I*: *trace*(*mat 1* :: '*a*::*semiring-1*^'*n*^'*n*) = *of-nat*(*CARD*('*n*))
by (*simp add*: *trace-def mat-def*)

lemma *trace-add*: *trace* ((*A*::*'a*::*comm-semiring-1^'n^'n*) + *B*) = *trace A* + *trace B*
by (*simp add*: *trace-def setsum-addf*)

lemma *trace-sub*: *trace* ((*A*::*'a*::*comm-ring-1^'n^'n*) − *B*) = *trace A* − *trace B*
by (*simp add*: *trace-def setsum-subtractf*)

lemma *trace-mul-sym*:*trace* ((*A*::*'a*::*comm-semiring-1^'n^'n*) ** *B*) = *trace* (*B***A*)
apply (*simp add*: *trace-def matrix-matrix-mult-def*)
apply (*subst setsum-commute*)
by (*simp add*: *mult-commute*)

definition *det*:: *'a*::*comm-ring-1^'n^'n* ⇒ *'a* **where**
det A = *setsum* (λ*p*. *of-int* (*sign p*) ∗ *setprod* (λ*i*. *A*$*i*$*p i*) (*UNIV* :: *'n set*)) {*p*. *p permutes* (*UNIV* :: *'n set*)}

lemma *setprod-permute*:
assumes *p*: *p permutes S*
shows *setprod f S* = *setprod* (*f o p*) *S*
proof−
{assume ¬ *finite S* **hence** *?thesis* **by** *simp***}**
moreover
{assume *fS*: *finite S*
then have *?thesis*
apply (*simp add*: *setprod-def cong del*:*strong-setprod-cong*)
apply (*rule ab-semigroup-mult.fold-image-permute*)
apply (*auto simp add*: *p*)
apply *unfold-locales*
done}
ultimately show *?thesis* **by** *blast*
qed

lemma *setproduct-permute-nat-interval*: *p permutes* {*m*::*nat* .. *n*} ==> *setprod f* {*m*..*n*} = *setprod* (*f o p*) {*m*..*n*}
by (*blast intro*!: *setprod-permute*)

lemma *det-transpose*: *det* (*transpose A*) = *det* (*A*::*'a*::*comm-ring-1 ^'n^'n*)

proof−
let *?di* = λ*A i j*. *A*$*i*$*j*
let *?U* = (*UNIV* :: *'n set*)
have *fU*: *finite ?U* **by** *simp*
{fix *p* **assume** *p*: *p* ∈ {*p*. *p permutes ?U*}
from *p* **have** *pU*: *p permutes ?U* **by** *blast*
have *sth*: *sign* (*inv p*) = *sign p*
by (*metis sign-inverse fU p mem-def Collect-def permutation-permutes*)
from *permutes-inj*[*OF pU*]
have *pi*: *inj-on p ?U* **by** (*blast intro*: *subset-inj-on*)
from *permutes-image*[*OF pU*]
have *setprod* (λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) *?U* = *setprod* (λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) (*p* ' *?U*) **by** *simp*
also have ... = *setprod* ((λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) *o p*) *?U*
unfolding *setprod-reindex*[*OF pi*] **..**
also have ... = *setprod* (λ*i*. *?di A i* (*p i*)) *?U*
proof−
{fix *i* **assume** *i*: *i* ∈ *?U*
from *i permutes-inv-o*[*OF pU*] *permutes-in-image*[*OF pU*]
have ((λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) *o p*) *i* = *?di A i* (*p i*)
unfolding *transpose-def* **by** (*simp add*: *expand-fun-eq*)**}**
then show *setprod* ((λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) *o p*) *?U* = *setprod* (λ*i*. *?di A i* (*p i*)) *?U* **by** (*auto intro*: *setprod-cong*)
qed
finally have *of-int* (*sign* (*inv p*)) ∗ (*setprod* (λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) *?U*) = *of-int* (*sign p*) ∗ (*setprod* (λ*i*. *?di A i* (*p i*)) *?U*) **using** *sth*
by *simp***}**
then show *?thesis* **unfolding** *det-def* **apply** (*subst setsum-permutations-inverse*)
apply (*rule setsum-cong2*) **by** *blast*
qed

lemma *det-lowerdiagonal*:
fixes *A* :: *'a*::*comm-ring-1*^(*'n*::{*finite*,*wellorder*})^(*'n*::{*finite*,*wellorder*})
assumes *ld*: ⋀*i j*. *i* < *j* ⟹ *A*$*i*$*j* = *0*
shows *det A* = *setprod* (λ*i*. *A*$*i*$*i*) (*UNIV*:: *'n set*)
proof−
let *?U* = *UNIV*:: *'n set*
let *?PU* = {*p*. *p permutes ?U*}
let *?pp* = λ*p*. *of-int* (*sign p*) ∗ *setprod* (λ*i*. *A*$*i*$*p i*) (*UNIV* :: *'n set*)
have *fU*: *finite ?U* **by** *simp*
from *finite-permutations*[*OF fU*] **have** *fPU*: *finite ?PU* **.**
have *id0*: {*id*} ⊆ *?PU* **by** (*auto simp add*: *permutes-id*)
{fix *p* **assume** *p*: *p* ∈ *?PU* −{*id*}
from *p* **have** *pU*: *p permutes ?U* **and** *pid*: *p* ≠ *id* **by** *blast*+
from *permutes-natset-le*[*OF pU*] *pid* **obtain** *i* **where**
i: *p i* > *i* **by** (*metis not-le*)
from *ld*[*OF i*] **have** *ex*:∃ *i* ∈ *?U*. *A*$*i*$*p i* = *0* **by** *blast*
from *setprod-zero*[*OF fU ex*] **have** *?pp p* = *0* **by** *simp***}**
then have *p0*: ∀ *p* ∈ *?PU* −{*id*}. *?pp p* = *0* **by** *blast*

```
  from setsum-mono-zero-cong-left[OF fPU id0 p0] show ?thesis
    unfolding det-def by (simp add: sign-id)
qed

lemma det-upperdiagonal:
  fixes A :: 'a::comm-ring-1^'n::{finite,wellorder}^'n::{finite,wellorder}
  assumes ld: ⋀i j. i > j ⟹ A$i$j = 0
  shows det A = setprod (λi. A$i$i) (UNIV:: 'n set)
proof−
  let ?U = UNIV:: 'n set
  let ?PU = {p. p permutes ?U}
  let ?pp = (λp. of-int (sign p) * setprod (λi. A$i$p i) (UNIV :: 'n set))
  have fU: finite ?U by simp
  from finite-permutations[OF fU] have fPU: finite ?PU .
  have id0: {id} ⊆ ?PU by (auto simp add: permutes-id)
  {fix p assume p: p ∈ ?PU −{id}
    from p have pU: p permutes ?U and pid: p ≠ id by blast+
    from permutes-natset-ge[OF pU] pid obtain i where
      i: p i < i by (metis not-le)
    from ld[OF i] have ex:∃ i ∈ ?U. A$i$p i = 0 by blast
    from setprod-zero[OF fU ex] have ?pp p = 0 by simp}
  then have p0: ∀ p ∈ ?PU −{id}. ?pp p = 0  by blast
  from    setsum-mono-zero-cong-left[OF fPU id0 p0] show ?thesis
    unfolding det-def by (simp add: sign-id)
qed

lemma det-diagonal:
  fixes A :: 'a::comm-ring-1^'n^'n
  assumes ld: ⋀i j. i ≠ j ⟹ A$i$j = 0
  shows det A = setprod (λi. A$i$i) (UNIV::'n set)
proof−
  let ?U = UNIV:: 'n set
  let ?PU = {p. p permutes ?U}
  let ?pp = λp. of-int (sign p) * setprod (λi. A$i$p i) (UNIV :: 'n set)
  have fU: finite ?U by simp
  from finite-permutations[OF fU] have fPU: finite ?PU .
  have id0: {id} ⊆ ?PU by (auto simp add: permutes-id)
  {fix p assume p: p ∈ ?PU − {id}
    then have p ≠ id by simp
    then obtain i where i: p i ≠ i unfolding expand-fun-eq by auto
    from ld [OF i [symmetric]] have ex:∃ i ∈ ?U. A$i$p i = 0 by blast
    from setprod-zero [OF fU ex] have ?pp p = 0 by simp}
  then have p0: ∀ p ∈ ?PU − {id}. ?pp p = 0  by blast
  from setsum-mono-zero-cong-left[OF fPU id0 p0] show ?thesis
    unfolding det-def by (simp add: sign-id)
qed

lemma det-I: det (mat 1 :: 'a::comm-ring-1^'n^'n) = 1
proof−
```

let *?A = mat 1* :: *'a*::*comm-ring-1^'n^'n*
let *?U = UNIV* :: *'n set*
let *?f = λi j. ?Aij*
{fix *i* **assume** *i*: *i ∈ ?U*
have *?f i i = 1* **using** *i* **by** (*vector mat-def*)**}**
hence *th*: *setprod* (*λi. ?f i i*) *?U = setprod* (*λx. 1*) *?U*
by (*auto intro*: *setprod-cong*)
{fix *i j* **assume** *i*: *i ∈ ?U* **and** *j*: *j ∈ ?U* **and** *ij*: *i ≠ j*
have *?f i j = 0* **using** *i j ij* **by** (*vector mat-def*) **}**
then have *det ?A = setprod* (*λi. ?f i i*) *?U* **using** *det-diagonal*
by *blast*
also have *... = 1* **unfolding** *th setprod-1* **..**
finally show *?thesis* **.**
qed

lemma *det-0*: *det* (*mat 0* :: *'a*::*comm-ring-1^'n^'n*) *= 0*
by (*simp add*: *det-def setprod-zero*)

lemma *det-permute-rows*:
fixes *A* :: *'a*::*comm-ring-1^'n^'n*
assumes *p*: *p permutes* (*UNIV* :: *'n*::*finite set*)
shows *det*(*χ i. A$p i* :: *'a^'n^'n*) *= of-int* (*sign p*) *∗ det A*
apply (*simp add*: *det-def setsum-right-distrib mult-assoc*[*symmetric*])
apply (*subst sum-permutations-compose-right*[*OF p*])
proof(*rule setsum-cong2*)
let *?U = UNIV* :: *'n set*
let *?PU = {p. p permutes ?U}*
fix *q* **assume** *qPU*: *q ∈ ?PU*
have *fU*: *finite ?U* **by** *simp*
from *qPU* **have** *q*: *q permutes ?U* **by** *blast*
from *p q* **have** *pp*: *permutation p* **and** *qp*: *permutation q*
by (*metis fU permutation-permutes*)+
from *permutes-inv*[*OF p*] **have** *ip*: *inv p permutes ?U* **.**
have *setprod* (*λi. A$p i$* (*q o p*) *i*) *?U = setprod* ((*λi. A$p i$*(*q o p*) *i*) *o inv p*) *?U*
by (*simp only*: *setprod-permute*[*OF ip*, *symmetric*])
also have *... = setprod* (*λi. A $* (*p o inv p*) *i $* (*q o* (*p o inv p*)) *i*) *?U*
by (*simp only*: *o-def*)
also have *... = setprod* (*λi. Aiq i*) *?U* **by** (*simp only*: *o-def permutes-inverses*[*OF p*])
finally have *thp*: *setprod* (*λi. A$p i$* (*q o p*) *i*) *?U = setprod* (*λi. Aiq i*) *?U*
by *blast*
show *of-int* (*sign* (*q o p*)) *∗ setprod* (*λi. A$ p i$* (*q o p*) *i*) *?U = of-int* (*sign p*) *∗ of-int* (*sign q*) *∗ setprod* (*λi. Aiq i*) *?U*
by (*simp only*: *thp sign-compose*[*OF qp pp*] *mult-commute of-int-mult*)
qed

lemma *det-permute-columns*:

fixes $A :: 'a::comm\text{-}ring\text{-}1\hat{}'n\hat{}'n$
assumes p: p *permutes* $(UNIV :: 'n\ set)$
shows $det(\chi\ i\ j.\ A\$i\$\ p\ j :: 'a\hat{}'n\hat{}'n) = of\text{-}int\ (sign\ p) * det\ A$
proof−
let $?Ap = \chi\ i\ j.\ A\$i\$\ p\ j :: 'a\hat{}'n\hat{}'n$
let $?At = transpose\ A$
have $of\text{-}int\ (sign\ p) * det\ A = det\ (transpose\ (\chi\ i.\ transpose\ A\ \$\ p\ i))$
unfolding *det-permute-rows*[*OF p*, *of ?At*] *det-transpose* **..**
moreover
have $?Ap = transpose\ (\chi\ i.\ transpose\ A\ \$\ p\ i)$
by (*simp add*: *transpose-def Cart-eq*)
ultimately show *?thesis* **by** *simp*
qed

lemma *det-identical-rows*:
fixes $A :: 'a::linordered\text{-}idom\hat{}'n\hat{}'n$
assumes ij: $i \neq j$
and r: $row\ i\ A = row\ j\ A$
shows $det\ A = 0$
proof−
have tha: $\bigwedge(a::'a)\ b.\ a = b \Longrightarrow b = -\ a \Longrightarrow a = 0$
by *simp*
have *th1*: $of\text{-}int\ (-1) = -\ 1$ **by** (*metis of-int-1 of-int-minus number-of-Min*)
let $?p = Fun.swap\ i\ j\ id$
let $?A = \chi\ i.\ A\ \$\ ?p\ i$
from r **have** $A = ?A$ **by** (*simp add*: *Cart-eq row-def swap-def*)
hence $det\ A = det\ ?A$ **by** *simp*
moreover have $det\ A = -\ det\ ?A$
by (*simp add*: *det-permute-rows*[*OF permutes-swap-id*] *sign-swap-id ij th1*)
ultimately show $det\ A = 0$ **by** (*metis tha*)
qed

lemma *det-identical-columns*:
fixes $A :: 'a::linordered\text{-}idom\hat{}'n\hat{}'n$
assumes ij: $i \neq j$
and r: $column\ i\ A = column\ j\ A$
shows $det\ A = 0$
apply (*subst det-transpose*[*symmetric*])
apply (*rule det-identical-rows*[*OF ij*])
by (*metis row-transpose r*)

lemma *det-zero-row*:
fixes $A :: 'a::\{idom, ring\text{-}char\text{-}0\}\hat{}'n\hat{}'n$
assumes r: $row\ i\ A = 0$
shows $det\ A = 0$
using r
apply (*simp add*: *row-def det-def Cart-eq*)
apply (*rule setsum-0'*)
apply (*auto simp*: *sign-nz*)

done

lemma *det-zero-column*:
fixes *A* :: *'a*::{*idom*,*ring-char-0*}^*'n*^*'n*
assumes *r*: *column i A = 0*
shows *det A = 0*
apply (*subst det-transpose*[*symmetric*])
apply (*rule det-zero-row* [*of i*])
by (*metis row-transpose r*)

lemma *det-row-add*:
fixes *a b c* :: *'n*::*finite* ⇒ - ^ *'n*
shows *det*((χ *i*. *if i = k then a i + b i else c i*)::*'a*::*comm-ring-1*^*'n*^*'n*) =
det((χ *i*. *if i = k then a i else c i*)::*'a*::*comm-ring-1*^*'n*^*'n*) +
det((χ *i*. *if i = k then b i else c i*)::*'a*::*comm-ring-1*^*'n*^*'n*)
unfolding *det-def Cart-lambda-beta setsum-addf*[*symmetric*]
proof (*rule setsum-cong2*)
let *?U = UNIV* :: *'n set*
let *?pU* = {*p*. *p permutes ?U*}
let *?f* = (λ*i*. *if i = k then a i + b i else c i*)::*'n* ⇒ *'a*::*comm-ring-1*^*'n*
let *?g* = (λ *i*. *if i = k then a i else c i*)::*'n* ⇒ *'a*::*comm-ring-1*^*'n*
let *?h* = (λ *i*. *if i = k then b i else c i*)::*'n* ⇒ *'a*::*comm-ring-1*^*'n*
fix *p* **assume** *p*: *p* ∈ *?pU*
let *?Uk = ?U* − {*k*}
from *p* **have** *pU*: *p permutes ?U* **by** *blast*
have *kU*: *?U = insert k ?Uk* **by** *blast*
{fix *j* **assume** *j*: *j* ∈ *?Uk*
from *j* **have** *?f j* $ *p j = ?g j* $ *p j* **and** *?f j* $ *p j*= *?h j* $ *p j*
by *simp-all***}**
then have *th1*: *setprod* (λ*i*. *?f i* $ *p i*) *?Uk = setprod* (λ*i*. *?g i* $ *p i*) *?Uk*
and *th2*: *setprod* (λ*i*. *?f i* $ *p i*) *?Uk = setprod* (λ*i*. *?h i* $ *p i*) *?Uk*
apply −
apply (*rule setprod-cong*, *simp-all*)+
done
have *th3*: *finite ?Uk k* ∉ *?Uk* **by** *auto*
have *setprod* (λ*i*. *?f i* $ *p i*) *?U = setprod* (λ*i*. *?f i* $ *p i*) (*insert k ?Uk*)
unfolding *kU*[*symmetric*] **..**
also have ... = *?f k* $ *p k* ∗ *setprod* (λ*i*. *?f i* $ *p i*) *?Uk*
apply (*rule setprod-insert*)
apply *simp*
by *blast*
also have ... = (*a k* $ *p k* ∗ *setprod* (λ*i*. *?f i* $ *p i*) *?Uk*) + (*b k*$ *p k* ∗ *setprod* (λ*i*. *?f i* $ *p i*) *?Uk*) **by** (*simp add*: *field-simps*)
also have ... = (*a k* $ *p k* ∗ *setprod* (λ*i*. *?g i* $ *p i*) *?Uk*) + (*b k*$ *p k* ∗ *setprod* (λ*i*. *?h i* $ *p i*) *?Uk*) **by** (*metis th1 th2*)
also have ... = *setprod* (λ*i*. *?g i* $ *p i*) (*insert k ?Uk*) + *setprod* (λ*i*. *?h i* $ *p i*) (*insert k ?Uk*)
unfolding *setprod-insert*[*OF th3*] **by** *simp*
finally have *setprod* (λ*i*. *?f i* $ *p i*) *?U = setprod* (λ*i*. *?g i* $ *p i*) *?U* + *setprod*

$(\lambda i.\ ?h\ i\ \$\ p\ i)\ ?U$ **unfolding** $kU[symmetric]$ **.**
then show *of-int (sign p) * setprod (λi. ?f i $ p i) ?U = of-int (sign p) * setprod (λi. ?g i $ p i) ?U + of-int (sign p) * setprod (λi. ?h i $ p i) ?U*
by (*simp add*: *field-simps*)
qed

lemma *det-row-mul*:
fixes *a b :: 'n::finite ⇒ - ^ 'n*
shows *det((χ i. if i = k then c *s a i else b i)::'a::comm-ring-1^'n^'n) =*
c det((χ i. if i = k then a i else b i)::'a::comm-ring-1^'n^'n)*

unfolding *det-def Cart-lambda-beta setsum-right-distrib*
proof (*rule setsum-cong2*)
let *?U = UNIV :: 'n set*
let *?pU = {p. p permutes ?U}*
let *?f = (λi. if i = k then c*s a i else b i)::'n ⇒ 'a::comm-ring-1^'n*
let *?g = (λ i. if i = k then a i else b i)::'n ⇒ 'a::comm-ring-1^'n*
fix *p* **assume** *p*: *p ∈ ?pU*
let *?Uk = ?U − {k}*
from *p* **have** *pU*: *p permutes ?U* **by** *blast*
have *kU*: *?U = insert k ?Uk* **by** *blast*
{fix *j* **assume** *j*: *j ∈ ?Uk*
from *j* **have** *?f j $ p j = ?g j $ p j* **by** *simp***}**
then have *th1*: *setprod (λi. ?f i $ p i) ?Uk = setprod (λi. ?g i $ p i) ?Uk*
apply −
apply (*rule setprod-cong*, *simp-all*)
done
have *th3*: *finite ?Uk k ∉ ?Uk* **by** *auto*
have *setprod (λi. ?f i $ p i) ?U = setprod (λi. ?f i $ p i) (insert k ?Uk)*
unfolding $kU[symmetric]$ **..**
also have *... = ?f k $ p k * setprod (λi. ?f i $ p i) ?Uk*
apply (*rule setprod-insert*)
apply *simp*
by *blast*
also have *... = (c*s a k) $ p k * setprod (λi. ?f i $ p i) ?Uk* **by** (*simp add*: *field-simps*)
also have *... = c* (a k $ p k * setprod (λi. ?g i $ p i) ?Uk)*
unfolding *th1* **by** (*simp add*: *mult-ac*)
also have *... = c* (setprod (λi. ?g i $ p i) (insert k ?Uk))*
unfolding *setprod-insert*[*OF th3*] **by** *simp*
finally have *setprod (λi. ?f i $ p i) ?U = c* (setprod (λi. ?g i $ p i) ?U)*
unfolding $kU[symmetric]$ **.**
then show *of-int (sign p) * setprod (λi. ?f i $ p i) ?U = c * (of-int (sign p) * setprod (λi. ?g i $ p i) ?U)*
by (*simp add*: *field-simps*)
qed

lemma *det-row-0*:
fixes *b :: 'n::finite ⇒ - ^ 'n*

```
  shows det((χ i. if i = k then 0 else b i)::'a::comm-ring-1^'n^'n) = 0
using det-row-mul[of k 0 λi. 1 b]
apply (simp)
  unfolding vector-smult-lzero .

lemma det-row-operation:
  fixes A :: 'a::linordered-idom^'n^'n
  assumes ij: i ≠ j
  shows det (χ k. if k = i then row i A + c *s row j A else row k A) = det A
proof−
  let ?Z = (χ k. if k = i then row j A else row k A) :: 'a ^'n^'n
  have th: row i ?Z = row j ?Z by (vector row-def)
  have th2: ((χ k. if k = i then row i A else row k A) :: 'a^'n^'n) = A
    by (vector row-def)
  show ?thesis
    unfolding det-row-add [of i] det-row-mul[of i] det-identical-rows[OF ij th] th2
    by simp
qed

lemma det-row-span:
  fixes A :: real^'n^'n
  assumes x: x ∈ span {row j A |j. j ≠ i}
  shows det (χ k. if k = i then row i A + x else row k A) = det A
proof−
  let ?U = UNIV :: 'n set
  let ?S = {row j A |j. j ≠ i}
  let ?d = λx. det (χ k. if k = i then x else row k A)
  let ?P = λx. ?d (row i A + x) = det A
  {fix k

    have (if k = i then row i A + 0 else row k A) = row k A by simp}
  then have P0: ?P 0
    apply −
    apply (rule cong[of det, OF refl])
    by (vector row-def)
  moreover
  {fix c z y assume zS: z ∈ ?S and Py: ?P y
    from zS obtain j where j: z = row j A i ≠ j by blast
    let ?w = row i A + y
    have th0: row i A + (c*s z + y) = ?w + c*s z by vector
    have thz: ?d z = 0
      apply (rule det-identical-rows[OF j(2)])
      using j by (vector row-def)
    have ?d (row i A + (c*s z + y)) = ?d (?w + c*s z) unfolding th0 ..
    then have ?P (c*s z + y) unfolding thz Py det-row-mul[of i] det-row-add[of
i]
      by simp }

  ultimately show ?thesis
```

```
    apply −
    apply (rule span-induct-alt[of ?P ?S, OF P0, folded smult-conv-scaleR])
    apply blast
    apply (rule x)
    done
qed
```

```
lemma det-dependent-rows:
  fixes A:: real^'n^'n
  assumes d: dependent (rows A)
  shows det A = 0
proof−
  let ?U = UNIV :: 'n set
  from d obtain i where i: row i A ∈ span (rows A − {row i A})
    unfolding dependent-def rows-def by blast
  {fix j k assume jk: j ≠ k
    and c: row j A = row k A
    from det-identical-rows[OF jk c] have ?thesis .}
  moreover
  {assume H: ⋀ i j. i ≠ j ⟹ row i A ≠ row j A
    have th0: − row i A ∈ span {row j A|j. j ≠ i}
      apply (rule span-neg)
      apply (rule set-rev-mp)
      apply (rule i)
      apply (rule span-mono)
      using H i by (auto simp add: rows-def)
    from det-row-span[OF th0]
    have det A = det (χ k. if k = i then 0 *s 1 else row k A)
      unfolding right-minus vector-smult-lzero ..
    with det-row-mul[of i 0::real λi. 1]
    have det A = 0 by simp}
  ultimately show ?thesis by blast
qed

lemma det-dependent-columns: assumes d: dependent(columns (A::real^'n^'n))
shows det A = 0
by (metis d det-dependent-rows rows-transpose det-transpose)

lemma Cart-lambda-cong: (⋀x. f x = g x) ⟹ (Cart-lambda f::'a^'n) = (Cart-lambda
g :: 'a^'n)
```

apply (*rule iffD1*[*OF Cart-lambda-unique*]) **by** *vector*

lemma *det-linear-row-setsum*:
assumes *fS*: *finite S*
shows *det* ((χ *i*. *if i* = *k then setsum* (*a i*) *S else c i*)::′*a*::*comm-ring-1*^′*n*^′*n*)
= *setsum* (λ*j*. *det* ((χ *i*. *if i* = *k then a i j else c i*)::′*a*^′*n*^′*n*)) *S*
proof(*induct rule*: *finite-induct*[*OF fS*])
case *1* **thus** *?case* **apply** *simp* **unfolding** *setsum-empty det-row-0*[*of k*] **..**
next
case (*2 x F*)
then show *?case* **by** (*simp add*: *det-row-add cong del*: *if-weak-cong*)
qed

lemma *finite-bounded-functions*:
assumes *fS*: *finite S*
shows *finite* {*f*. (∀ *i* ∈ {*1*.. (*k*::*nat*)}. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ {*1* .. *k*} ⟶ *f i* = *i*)}
proof(*induct k*)
case *0*
have *th*: {*f*. ∀ *i*. *f i* = *i*} = {*id*} **by** (*auto intro*: *ext*)
show *?case* **by** (*auto simp add*: *th*)
next
case (*Suc k*)
let *?f* = λ(*y*::*nat*,*g*) *i*. *if i* = *Suc k then y else g i*
let *?S* = *?f* ‘ (*S* × {*f*. (∀ *i*∈{*1*..*k*}. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ {*1*..*k*} ⟶ *f i* = *i*)})
have *?S* = {*f*. (∀ *i*∈{*1*.. *Suc k*}. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ {*1*.. *Suc k*} ⟶ *f i* = *i*)}
apply (*auto simp add*: *image-iff*)
apply (*rule-tac x*=*x* (*Suc k*) **in** *bexI*)
apply (*rule-tac x* = λ*i*. *if i* = *Suc k then i else x i* **in** *exI*)
apply (*auto intro*: *ext*)
done
with *finite-imageI*[*OF finite-cartesian-product*[*OF fS Suc.hyps*(*1*)], *of ?f*]
show *?case* **by** *metis*
qed

lemma *eq-id-iff*[*simp*]: (∀ *x*. *f x* = *x*) = (*f* = *id*) **by** (*auto intro*: *ext*)

lemma *det-linear-rows-setsum-lemma*:
assumes *fS*: *finite S* **and** *fT*: *finite T*
shows *det*((χ *i*. *if i* ∈ *T then setsum* (*a i*) *S else c i*):: ′*a*::*comm-ring-1*^′*n*^′*n*)
=
setsum (λ*f*. *det*((χ *i*. *if i* ∈ *T then a i* (*f i*) *else c i*)::′*a*^′*n*^′*n*))
{*f*. (∀ *i* ∈ *T*. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ *T* ⟶ *f i* = *i*)}
using *fT*
proof(*induct T arbitrary*: *a c set*: *finite*)
case *empty*
have *th0*: ⋀*x y*. (χ *i*. *if i* ∈ {} *then x i else y i*) = (χ *i*. *y i*) **by** *vector*
from *empty.prems* **show** *?case* **unfolding** *th0* **by** *simp*
next

case (*insert z T a c*)
let *?F* = $\lambda T.\ \{f.\ (\forall i \in T.\ f\ i \in S) \wedge (\forall i.\ i \notin T \longrightarrow f\ i = i)\}$
let *?h* = $\lambda(y,g)\ i.$ *if i = z then y else g i*
let *?k* = $\lambda h.$ (*h*(*z*),($\lambda i.$ *if i = z then i else h i*))
let *?s* = $\lambda\ k\ a\ c\ f.$ *det*(($\chi\ i.$ *if i ∈ T then a i* (*f i*) *else c i*)::*'a^'n^'n*)
let *?c* = $\lambda i.$ *if i = z then a i j else c i*
have *thif*: $\bigwedge a\ b\ c\ d.$ (*if a ∨ b then c else d*) = (*if a then c else if b then c else d*) **by** *simp*
have *thif2*: $\bigwedge a\ b\ c\ d\ e.$ (*if a then b else if c then d else e*) =
(*if c then* (*if a then b else d*) *else* (*if a then b else e*)) **by** *simp*
from ‹*z* ∉ *T*› **have** *nz*: $\bigwedge i.\ i \in T \Longrightarrow i = z \longleftrightarrow$ *False* **by** *auto*
have *det* ($\chi\ i.$ *if i ∈ insert z T then setsum* (*a i*) *S else c i*) =
det ($\chi\ i.$ *if i = z then setsum* (*a i*) *S*
else if i ∈ T then setsum (*a i*) *S else c i*)
unfolding *insert-iff thif* **..**
also have ... = ($\sum j{\in}S.$ *det* ($\chi\ i.$ *if i ∈ T then setsum* (*a i*) *S*
else if i = z then a i j else c i))
unfolding *det-linear-row-setsum*[*OF fS*]
apply (*subst thif2*)
using *nz* **by** (*simp cong del*: *if-weak-cong cong add*: *if-cong*)
finally have *tha*:
det ($\chi\ i.$ *if i ∈ insert z T then setsum* (*a i*) *S else c i*) =
($\sum (j, f){\in}S \times$ *?F T*. *det* ($\chi\ i.$ *if i ∈ T then a i* (*f i*)
else if i = z then a i j
else c i))
unfolding *insert.hyps* **unfolding** *setsum-cartesian-product* **by** *blast*
show *?case* **unfolding** *tha*
apply(*rule setsum-eq-general-reverses*[**where** *h*= *?h* **and** *k*= *?k*],
blast intro: *finite-cartesian-product fS finite*,
blast intro: *finite-cartesian-product fS finite*)
using ‹*z* ∉ *T*›
apply (*auto intro*: *ext*)
apply (*rule cong*[*OF refl*[*of det*]])
by *vector*
qed

lemma *det-linear-rows-setsum*:
assumes *fS*: *finite* (*S*::*'n*::*finite set*)
shows *det* ($\chi\ i.$ *setsum* (*a i*) *S*) = *setsum* ($\lambda f.$ *det* ($\chi\ i.$ *a i* (*f i*) :: *'a*::*comm-ring-1 ^ 'n^'n*)) {*f*. $\forall i.$ *f i ∈ S*}
proof−
have *th0*: $\bigwedge x\ y.$ (($\chi\ i.$ *if i ∈* (*UNIV*:: *'n set*) *then x i else y i*) :: *'a^'n^'n*) = ($\chi\ i.$ *x i*) **by** *vector*

from *det-linear-rows-setsum-lemma*[*OF fS*, *of UNIV* :: *'n set a*, *unfolded th0*, *OF finite*] **show** *?thesis* **by** *simp*
qed

lemma *matrix-mul-setsum-alt*:

fixes *A B* :: *'a::comm-ring-1^'n^'n*
shows *A ** B = (χ i. setsum (λk. Aik *s B $ k) (UNIV :: 'n set))*
by (*vector matrix-matrix-mult-def setsum-component*)

lemma *det-rows-mul*:
*det((χ i. c i *s a i)::'a::comm-ring-1^'n^'n) =*
*setprod (λi. c i) (UNIV:: 'n set) * det((χ i. a i)::'a^'n^'n)*
proof (*simp add*: *det-def setsum-right-distrib cong add*: *setprod-cong*, *rule setsum-cong2*)
let *?U = UNIV :: 'n set*
let *?PU = {p. p permutes ?U}*
fix *p* **assume** *pU*: *p ∈ ?PU*
let *?s = of-int (sign p)*
from *pU* **have** *p*: *p permutes ?U* **by** *blast*
have *setprod (λi. c i * a i $ p i) ?U = setprod c ?U * setprod (λi. a i $ p i) ?U*
unfolding *setprod-timesf* **..**
then show *?s * (∏ xa∈?U. c xa * a xa $ p xa) =*
*setprod c ?U * (?s* (∏ xa∈?U. a xa $ p xa))* **by** (*simp add*: *field-simps*)
qed

lemma *det-mul*:
fixes *A B* :: *'a::linordered-idom^'n^'n*
shows *det (A ** B) = det A * det B*
proof−
let *?U = UNIV :: 'n set*
let *?F = {f. (∀ i∈ ?U. f i ∈ ?U) ∧ (∀ i. i ∉ ?U ⟶ f i = i)}*
let *?PU = {p. p permutes ?U}*
have *fU*: *finite ?U* **by** *simp*
have *fF*: *finite ?F* **by** (*rule finite*)
{fix *p* **assume** *p*: *p permutes ?U*

have *p ∈ ?F* **unfolding** *mem-Collect-eq permutes-in-image[OF p]*
using *p[unfolded permutes-def]* **by** *simp***}**
then have *PUF*: *?PU ⊆ ?F* **by** *blast*
{fix *f* **assume** *fPU*: *f ∈ ?F − ?PU*
have *fUU*: *f ' ?U ⊆ ?U* **using** *fPU* **by** *auto*
from *fPU* **have** *f*: *∀ i ∈ ?U. f i ∈ ?U*
∀ i. i ∉ ?U ⟶ f i = i ¬(∀ y. ∃!x. f x = y) **unfolding** *permutes-def*
by *auto*

let *?A = (χ i. Aif i *s B$f i) :: 'a^'n^'n*
let *?B = (χ i. B$f i) :: 'a^'n^'n*
{assume *fni*: *¬ inj-on f ?U*
then obtain *i j* **where** *ij*: *f i = f j i ≠ j*
unfolding *inj-on-def* **by** *blast*
from *ij*
have *rth*: *row i ?B = row j ?B* **by** (*vector row-def*)
from *det-identical-rows[OF ij(2) rth]*
have *det (χ i. Aif i *s B$f i) = 0*
unfolding *det-rows-mul* **by** *simp***}**

moreover
{assume *fi*: *inj-on f ?U*
from *f fi* **have** *fith*: $\bigwedge i\ j.\ f\ i = f\ j \Longrightarrow i = j$
unfolding *inj-on-def* **by** *metis*
note *fs* = *fi*[*unfolded surjective-iff-injective-gen*[*OF fU fU refl fUU*, *symmetric*]]

{fix *y*
from *fs f* **have** $\exists\, x.\ f\ x = y$ **by** *blast*
then obtain *x* **where** *x*: $f\ x = y$ **by** *blast*
{fix *z* **assume** *z*: $f\ z = y$ **from** *fith x z* **have** $z = x$ **by** *metis***}**
with *x* **have** $\exists!x.\ f\ x = y$ **by** *blast***}**
with *f*(*3*) **have** *det* $(\chi\ i.\ A\$i\$f\ i *s B\$f\ i) = 0$ **by** *blast***}**
ultimately have *det* $(\chi\ i.\ A\$i\$f\ i *s B\$f\ i) = 0$ **by** *blast***}**
hence *zth*: $\forall\ f \in ?F - ?PU.\ det\ (\chi\ i.\ A\$i\$f\ i *s B\$f\ i) = 0$ **by** *simp*
{fix *p* **assume** *pU*: $p \in ?PU$
from *pU* **have** *p*: *p permutes ?U* **by** *blast*
let *?s* = $\lambda p.$ *of-int* (*sign p*)
let *?f* = $\lambda q.$ *?s p* $*$ ($\prod i \in ?U.\ A\ \$\ i\ \$\ p\ i$) $*$
(*?s q* $*$ ($\prod i \in ?U.\ B\ \$\ i\ \$\ q\ i$))
have (*setsum* ($\lambda q.$ *?s q* $*$
($\prod i \in ?U.$ ($\chi\ i.\ A\ \$\ i\ \$\ p\ i *s B\ \$\ p\ i :: 'a\hat{}'n\hat{}'n$) $\$\ i\ \$\ q\ i$)) *?PU*) =
(*setsum* ($\lambda q.$ *?s p* $*$ ($\prod i \in ?U.\ A\ \$\ i\ \$\ p\ i$) $*$
(*?s q* $*$ ($\prod i \in ?U.\ B\ \$\ i\ \$\ q\ i$))) *?PU*)
unfolding *sum-permutations-compose-right*[*OF permutes-inv*[*OF p*], *of ?f*]
proof(*rule setsum-cong2*)
fix *q* **assume** *qU*: $q \in ?PU$
hence *q*: *q permutes ?U* **by** *blast*
from *p q* **have** *pp*: *permutation p* **and** *pq*: *permutation q*
unfolding *permutation-permutes* **by** *auto*
have *th00*: *of-int* (*sign p*) $*$ *of-int* (*sign p*) = (*1*::'*a*)
$\bigwedge a.$ *of-int* (*sign p*) $*$ (*of-int* (*sign p*) $*$ *a*) = *a*
unfolding *mult-assoc*[*symmetric*] **unfolding** *of-int-mult*[*symmetric*]
by (*simp-all add*: *sign-idempotent*)
have *ths*: *?s q* = *?s p* $*$ *?s* (*q o inv p*)
using *pp pq permutation-inverse*[*OF pp*] *sign-inverse*[*OF pp*]
by (*simp add*: *th00 mult-ac sign-idempotent sign-compose*)
have *th001*: *setprod* ($\lambda i.$ *B*$*i*$* *q* (*inv p i*)) *?U* = *setprod* (($\lambda i.$ *B*$*i*$* *q* (*inv p i*)) *o p*) *?U*
by (*rule setprod-permute*[*OF p*])
have *thp*: *setprod* ($\lambda i.$ ($\chi\ i.\ A\$i\$p\ i *s B\$p\ i :: 'a\hat{}'n\hat{}'n$) $\$i\ \$\ q\ i$) *?U* = *setprod* ($\lambda i.$ *A*$*i*$*p i*) *?U* $*$ *setprod* ($\lambda i.$ *B*$*i*$* *q* (*inv p i*)) *?U*
unfolding *th001 setprod-timesf*[*symmetric*] *o-def permutes-inverses*[*OF p*]
apply (*rule setprod-cong*[*OF refl*])
using *permutes-in-image*[*OF q*] **by** *vector*
show *?s q* $*$ *setprod* ($\lambda i.$ (($\chi\ i.\ A\$i\$p\ i *s B\$p\ i$) :: $'a\hat{}'n\hat{}'n$)$\$i\$q\ i$)) *?U* = *?s p* $*$ (*setprod* ($\lambda i.$ *A*$*i*$*p i*) *?U*) $*$ (*?s* (*q o inv p*) $*$ *setprod* ($\lambda i.$ *B*$*i*$*(q o inv p*) *i*) *?U*)
using *ths thp pp pq permutation-inverse*[*OF pp*] *sign-inverse*[*OF pp*]
by (*simp add*: *sign-nz th00 field-simps sign-idempotent sign-compose*)

```
    qed
  }
  then have th2: setsum (λf. det (χ i. A$i$f i *s B$f i)) ?PU = det A * det B
    unfolding det-def setsum-product
    by (rule setsum-cong2)
  have det (A**B) = setsum (λf.  det (χ i. A $ i $ f i *s B $ f i)) ?F
    unfolding matrix-mul-setsum-alt det-linear-rows-setsum[OF fU] by simp
  also have ... = setsum (λf. det (χ i. A$i$f i *s B$f i)) ?PU
    using setsum-mono-zero-cong-left[OF fF PUF zth, symmetric]
    unfolding det-rows-mul by auto
  finally show ?thesis unfolding th2 .
qed

lemma invertible-left-inverse:
  fixes A :: real^'n^'n
  shows invertible A ⟷ (∃(B::real^'n^'n). B** A = mat 1)
  by (metis invertible-def matrix-left-right-inverse)

lemma invertible-righ-inverse:
  fixes A :: real^'n^'n
  shows invertible A ⟷ (∃(B::real^'n^'n). A** B = mat 1)
  by (metis invertible-def matrix-left-right-inverse)

lemma invertible-det-nz:
  fixes A::real ^'n^'n
  shows invertible A ⟷ det A ≠ 0
proof-
  {assume invertible A
    then obtain B :: real ^'n^'n where B: A ** B = mat 1
      unfolding invertible-righ-inverse by blast
    hence det (A ** B) = det (mat 1 :: real ^'n^'n) by simp
    hence det A ≠ 0
      apply (simp add: det-mul det-I) by algebra }
  moreover
  {assume H: ¬ invertible A
    let ?U = UNIV :: 'n set
    have fU: finite ?U by simp
    from H obtain c i where c: setsum (λi. c i *s row i A) ?U = 0
      and iU: i ∈ ?U and ci: c i ≠ 0
      unfolding invertible-righ-inverse
      unfolding matrix-right-invertible-independent-rows by blast
    have stupid: ⋀(a::real^'n) b. a + b = 0 ⟹ -a = b
      apply (drule-tac f=op + (- a) in cong[OF refl])
      apply (simp only: ab-left-minus add-assoc[symmetric])
      apply simp
```

```
    done
  from c ci
  have thr0: − row i A = setsum (λj. (1/ c i) *s (c j *s row j A)) (?U − {i})
    unfolding setsum-diff1'[OF fU iU] setsum-cmul
    apply −
    apply (rule vector-mul-lcancel-imp[OF ci])
    apply (auto simp add: vector-smult-assoc vector-smult-rneg field-simps)
    unfolding stupid ..
  have thr: − row i A ∈ span {row j A| j. j ≠ i}
    unfolding thr0
    apply (rule span-setsum)
    apply simp
    apply (rule ballI)
    apply (rule span-mul [where 'a=real^'n, folded smult-conv-scaleR])+
    apply (rule span-superset)
    apply auto
    done
  let ?B = (χ k. if k = i then 0 else row k A) :: real ^'n^'n
  have thrb: row i ?B = 0 using iU by (vector row-def)
  have det A = 0
    unfolding det-row-span[OF thr, symmetric] right-minus
    unfolding  det-zero-row[OF thrb]  ..}
 ultimately show ?thesis by blast
qed
```

```
lemma cramer-lemma-transpose:
 fixes A:: real^'n^'n and x :: real^'n
 shows det ((χ i. if i = k then setsum (λi. x$i *s row i A) (UNIV::'n set)
                       else row i A)::real^'n^'n) = x$k * det A
 (is ?lhs = ?rhs)
proof−
 let ?U = UNIV :: 'n set
 let ?Uk = ?U − {k}
 have U: ?U = insert k ?Uk by blast
 have fUk: finite ?Uk by simp
 have kUk: k ∉ ?Uk by simp
 have th00: ⋀k s. x$k *s row k A + s = (x$k − 1) *s row k A + row k A + s
   by (vector field-simps)
 have th001: ⋀f k . (λx. if x = k then f k else f x) = f by (auto intro: ext)
 have (χ i. row i A) = A by (vector row-def)
 then have thd1: det (χ i. row i A) = det A  by simp
 have thd0: det (χ i. if i = k then row k A + (∑ i ∈ ?Uk. x $ i *s row i A) else
row i A) = det A
   apply (rule det-row-span)
   apply (rule span-setsum[OF fUk])
```

```
    apply (rule ballI)
    apply (rule span-mul [where 'a=real^'n, folded smult-conv-scaleR])+
    apply (rule span-superset)
    apply auto
    done
  show ?lhs = x$k * det A
    apply (subst U)
    unfolding setsum-insert[OF fUk kUk]
    apply (subst th00)
    unfolding add-assoc
    apply (subst det-row-add)
    unfolding thd0
    unfolding det-row-mul
    unfolding th001[of k λi. row i A]
    unfolding thd1  by (simp add: field-simps)
qed

lemma cramer-lemma:
  fixes A :: real^'n^'n
  shows det((χ i j. if j = k then (A *v x)$i else A$i$j):: real^'n^'n) = x$k * det
A
proof−
  let ?U = UNIV :: 'n set
  have stupid: ⋀c. setsum (λi. c i *s row i (transpose A)) ?U = setsum (λi. c i
*s column i A) ?U
    by (auto simp add: row-transpose intro: setsum-cong2)
  show ?thesis  unfolding matrix-mult-vsum
   unfolding cramer-lemma-transpose[of k x transpose A, unfolded det-transpose,
symmetric]
  unfolding stupid[of λi. x$i]
  apply (subst det-transpose[symmetric])
  apply (rule cong[OF refl[of det]]) by (vector transpose-def column-def row-def)
qed

lemma cramer:
  fixes A ::real^'n^'n
  assumes d0: det A ≠ 0
  shows A *v x = b ⟷ x = (χ k. det(χ i j. if j=k then b$i else A$i$j) / det A)
proof−
  from d0 obtain B where B: A ** B = mat 1 B ** A = mat 1
    unfolding invertible-det-nz[symmetric] invertible-def by blast
  have (A ** B) *v b = b by (simp add: B matrix-vector-mul-lid)
  hence A *v (B *v b) = b by (simp add: matrix-vector-mul-assoc)
  then have xe: ∃x. A*v x = b by blast
  {fix x assume x: A *v x = b
  have x = (χ k. det(χ i j. if j=k then b$i else A$i$j) / det A)
    unfolding x[symmetric]
    using d0 by (simp add: Cart-eq cramer-lemma field-simps)}
  with xe show ?thesis by auto
```

qed

definition *orthogonal-transformation f* $\longleftrightarrow$ *linear f* $\wedge$ $(\forall\, v\ w.\ f\ v \cdot f\ w = v \cdot w)$

lemma *orthogonal-transformation*: *orthogonal-transformation f* $\longleftrightarrow$ *linear f* $\wedge$ $(\forall\,(v{::}real\ \hat{}\text{-}).\ norm\ (f\ v) = norm\ v)$
unfolding *orthogonal-transformation-def*
apply *auto*
apply (*erule-tac x=v* **in** *allE*)+
apply (*simp add*: *norm-eq-sqrt-inner*)
by (*simp add*: *dot-norm linear-add*[*symmetric*])

definition *orthogonal-matrix* $(Q{::}'a{::}semiring\text{-}1\hat{}'n\hat{}'n) \longleftrightarrow$ *transpose Q* $**$ *Q* $=$ *mat 1* $\wedge$ *Q* $**$ *transpose Q* $=$ *mat 1*

lemma *orthogonal-matrix*: *orthogonal-matrix* $(Q{::}\ real\ \hat{}'n\hat{}'n) \longleftrightarrow$ *transpose Q* $**$ *Q* $=$ *mat 1*
by (*metis matrix-left-right-inverse orthogonal-matrix-def*)

lemma *orthogonal-matrix-id*: *orthogonal-matrix* $(mat\ 1 :: \text{-}\hat{}'n\hat{}'n)$
by (*simp add*: *orthogonal-matrix-def transpose-mat matrix-mul-lid*)

lemma *orthogonal-matrix-mul*:
fixes *A* :: *real* $\hat{}'n\hat{}'n$
assumes *oA* : *orthogonal-matrix A*
and *oB*: *orthogonal-matrix B*
shows *orthogonal-matrix*(*A* $**$ *B*)
using *oA oB*
unfolding *orthogonal-matrix matrix-transpose-mul*
apply (*subst matrix-mul-assoc*)
apply (*subst matrix-mul-assoc*[*symmetric*])
by (*simp add*: *matrix-mul-rid*)

lemma *orthogonal-transformation-matrix*:
fixes *f*:: *real*$\hat{}'n \Rightarrow real\hat{}'n$
shows *orthogonal-transformation f* $\longleftrightarrow$ *linear f* $\wedge$ *orthogonal-matrix*(*matrix f*)
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof−
let *?mf* $=$ *matrix f*
let *?ot* $=$ *orthogonal-transformation f*
let *?U* $=$ *UNIV* :: *'n set*
have *fU*: *finite ?U* **by** *simp*
let *?m1* $=$ *mat 1* :: *real* $\hat{}'n\hat{}'n$
{assume *ot*: *?ot*
from *ot* **have** *lf*: *linear f* **and** *fd*: $\forall\, v\ w.\ f\ v \cdot f\ w = v \cdot w$

```
    unfolding orthogonal-transformation-def orthogonal-matrix by blast+
   {fix i j
    let ?A = transpose ?mf ** ?mf
    have th0: ⋀b (x::'a::comm-ring-1). (if b then 1 else 0)*x = (if b then x else
0)
      ⋀b (x::'a::comm-ring-1). x*(if b then 1 else 0) = (if b then x else 0)
      by simp-all
   from fd[rule-format, of basis i basis j, unfolded matrix-works[OF lf, symmetric]
dot-matrix-vector-mul]
    have ?A$i$j = ?m1 $ i $ j
      by (simp add: inner-vector-def matrix-matrix-mult-def columnvector-def
rowvector-def basis-def th0 setsum-delta[OF fU] mat-def)}
   hence orthogonal-matrix ?mf unfolding orthogonal-matrix by vector
   with lf have ?rhs by blast}
 moreover
 {assume lf: linear f and om: orthogonal-matrix ?mf
  from lf om have ?lhs
   unfolding orthogonal-matrix-def norm-eq orthogonal-transformation
   unfolding matrix-works[OF lf, symmetric]
   apply (subst dot-matrix-vector-mul)
   by (simp add: dot-matrix-product matrix-mul-lid)}
 ultimately show ?thesis by blast
qed

lemma det-orthogonal-matrix:
 fixes Q:: 'a::linordered-idom^'n^'n
 assumes oQ: orthogonal-matrix Q
 shows det Q = 1 ∨ det Q = - 1
proof-

 have th: ⋀x::'a. x = 1 ∨ x = - 1 ⟷ x*x = 1 (is ⋀x::'a. ?ths x)
 proof-
  fix x:: 'a
  have th0: x*x - 1 = (x - 1)*(x + 1) by (simp add: field-simps)
  have th1: ⋀(x::'a) y. x = - y ⟷ x + y = 0
   apply (subst eq-iff-diff-eq-0) by simp
  have x*x = 1 ⟷ x*x - 1 = 0 by simp
  also have ... ⟷ x = 1 ∨ x = - 1 unfolding th0 th1 by simp
  finally show ?ths x ..
 qed
 from oQ have Q ** transpose Q = mat 1 by (metis orthogonal-matrix-def)
 hence det (Q ** transpose Q) = det (mat 1:: 'a^'n^'n) by simp
 hence det Q * det Q = 1 by (simp add: det-mul det-I det-transpose)
 then show ?thesis unfolding th .
qed
```

lemma *scaling-linear*:
fixes *f* :: *real ^'n ⇒ real ^'n*
assumes *f0*: *f 0 = 0* **and** *fd*: ∀ *x y*. *dist* (*f x*) (*f y*) = *c* ∗ *dist x y*
shows *linear f*
proof−
{fix *v w*
{fix *x* **note** *fd*[*rule-format*, *of x 0*, *unfolded dist-norm f0 diff-0-right*] **}**
note *th0* = *this*
have *f v* · *f w* = *c^2* ∗ (*v* · *w*)
unfolding *dot-norm-neg dist-norm*[*symmetric*]
unfolding *th0 fd*[*rule-format*] **by** (*simp add*: *power2-eq-square field-simps*)**}**
note *fc* = *this*
show *?thesis* **unfolding** *linear-def vector-eq*[**where** *'a=real^'n*] *smult-conv-scaleR*
by (*simp add*: *inner-simps fc field-simps*)
qed

lemma *isometry-linear*:
f (*0*:: *real^'n*) = (*0*:: *real^'n*) ⟹ ∀ *x y*. *dist*(*f x*) (*f y*) = *dist x y*
⟹ *linear f*
by (*rule scaling-linear*[**where** *c=1*]) *simp-all*

lemma *orthogonal-transformation-isometry*:
orthogonal-transformation f ⟷ *f*(*0*::*real^'n*) = (*0*::*real^'n*) ∧ (∀ *x y*. *dist*(*f x*) (*f y*) = *dist x y*)
unfolding *orthogonal-transformation*
apply (*rule iffI*)
apply *clarify*
apply (*clarsimp simp add*: *linear-0 linear-sub*[*symmetric*] *dist-norm*)
apply (*rule conjI*)
apply (*rule isometry-linear*)
apply *simp*
apply *simp*
apply *clarify*
apply (*erule-tac x=v* **in** *allE*)
apply (*erule-tac x=0* **in** *allE*)
by (*simp add*: *dist-norm*)

lemma *isometry-sphere-extend*:
fixes *f*:: *real ^'n ⇒ real ^'n*
assumes *f1*: ∀ *x*. *norm x = 1* ⟶ *norm* (*f x*) = *1*
and *fd1*: ∀ *x y*. *norm x = 1* ⟶ *norm y = 1* ⟶ *dist* (*f x*) (*f y*) = *dist x y*

shows $\exists g.$ *orthogonal-transformation* $g \wedge (\forall x.$ *norm* $x = 1 \longrightarrow g\ x = f\ x)$
proof−
{fix $x\ y\ x'\ y'\ x0\ y0\ x0'\ y0'$ :: *real* ^'*n*
assume H: $x =$ *norm* x *∗s* $x0\ y =$ *norm* y *∗s* $y0$
$x' =$ *norm* x *∗s* $x0'\ y' =$ *norm* y *∗s* $y0'$
norm $x0 = 1$ *norm* $x0' = 1$ *norm* $y0 = 1$ *norm* $y0' = 1$
norm$(x0' - y0') =$ *norm*$(x0 - y0)$
hence ∗:$x0 \cdot y0 = x0' \cdot y0' + y0' \cdot x0' - y0 \cdot x0$ **by**(*simp add*: *norm-eq norm-eq-1 inner-simps*)
have *norm*$(x' - y') =$ *norm*$(x - y)$
apply (*subst* $H(1)$)
apply (*subst* $H(2)$)
apply (*subst* $H(3)$)
apply (*subst* $H(4)$)
using $H(5-9)$
apply (*simp add*: *norm-eq norm-eq-1*)
apply (*simp add*: *inner-simps smult-conv-scaleR*) **unfolding** ∗
by (*simp add*: *field-simps*) **}**
note *th0* = *this*
let *?g* = $\lambda x.$ *if* $x = 0$ *then* 0 *else norm* x *∗s* f (*inverse* (*norm* x) *∗s* x)
{fix x:: *real* ^'*n* **assume** *nx*: *norm* $x = 1$
have *?g* $x = f\ x$ **using** *nx* **by** *auto***}**
hence *thfg*: $\forall x.$ *norm* $x = 1 \longrightarrow$ *?g* $x = f\ x$ **by** *blast*
have *g0*: *?g* $0 = 0$ **by** *simp*
{fix $x\ y$ :: *real* ^'*n*
{assume $x = 0\ y = 0$
then have *dist* (*?g* x) (*?g* y) = *dist* $x\ y$ **by** *simp* **}**
moreover
{assume $x = 0\ y \neq 0$
then have *dist* (*?g* x) (*?g* y) = *dist* $x\ y$
apply (*simp add*: *dist-norm*)
apply (*rule f1*[*rule-format*])
by(*simp add*: *field-simps*)**}**
moreover
{assume $x \neq 0\ y = 0$
then have *dist* (*?g* x) (*?g* y) = *dist* $x\ y$
apply (*simp add*: *dist-norm*)
apply (*rule f1*[*rule-format*])
by(*simp add*: *field-simps*)**}**
moreover
{assume z: $x \neq 0\ y \neq 0$
have *th00*: $x =$ *norm* x *∗s* (*inverse* (*norm* x) *∗s* x) $y =$ *norm* y *∗s* (*inverse* (*norm* y) *∗s* y) *norm* x *∗s* f ((*inverse* (*norm* x) *∗s* x)) = *norm* x *∗s* f (*inverse* (*norm* x) *∗s* x)
norm y *∗s* f (*inverse* (*norm* y) *∗s* y) = *norm* y *∗s* f (*inverse* (*norm* y) *∗s* y)
norm (*inverse* (*norm* x) *∗s* x) = 1
norm (f (*inverse* (*norm* x) *∗s* x)) = 1
norm (*inverse* (*norm* y) *∗s* y) = 1
norm (f (*inverse* (*norm* y) *∗s* y)) = 1

```
      norm (f (inverse (norm x) *s x) - f (inverse (norm y) *s y)) =
      norm (inverse (norm x) *s x - inverse (norm y) *s y)
      using z
        by (auto simp add: vector-smult-assoc field-simps intro: f1[rule-format]
fd1[rule-format, unfolded dist-norm])
    from z th0[OF th00] have dist (?g x) (?g y) = dist x y
      by (simp add: dist-norm)}
  ultimately have dist (?g x) (?g y) = dist x y by blast}
 note thd = this
  show ?thesis
  apply (rule exI[where x= ?g])
  unfolding orthogonal-transformation-isometry
    using  g0 thfg thd by metis
qed
```

```
definition rotation-matrix Q ⟷ orthogonal-matrix Q ∧ det Q = 1
definition rotoinversion-matrix Q ⟷ orthogonal-matrix Q ∧ det Q = - 1

lemma orthogonal-rotation-or-rotoinversion:
 fixes Q :: 'a::linordered-idom^'n^'n
 shows  orthogonal-matrix Q ⟷ rotation-matrix Q ∨ rotoinversion-matrix Q
 by (metis rotoinversion-matrix-def rotation-matrix-def det-orthogonal-matrix)
```

```
lemma setprod-1: setprod f {(1::nat)..1} = f 1 by simp

lemma setprod-2: setprod f {(1::nat)..2} = f 1 * f 2
 by (simp add: nat-number setprod-numseg mult-commute)
lemma setprod-3: setprod f {(1::nat)..3} = f 1 * f 2 * f 3
 by (simp add: nat-number setprod-numseg mult-commute)

lemma det-1: det (A::'a::comm-ring-1^1^1) = A$1$1
 by (simp add: det-def sign-id UNIV-1)

lemma det-2: det (A::'a::comm-ring-1^2^2) = A$1$1 * A$2$2 - A$1$2 * A$2$1
proof-
 have f12: finite {2::2} 1 ∉ {2::2} by auto
 show ?thesis
 unfolding det-def UNIV-2
 unfolding setsum-over-permutations-insert[OF f12]
 unfolding permutes-sing
 apply (simp add: sign-swap-id sign-id swap-id-eq)
 by (simp add: arith-simps(31)[symmetric] del: arith-simps(31))
```

qed

lemma *det-3*: *det* (*A*::*'a*::*comm-ring-1^3^3*) =
*A$1$1 * A$2$2 * A$3$3 +*
*A$1$2 * A$2$3 * A$3$1 +*
*A$1$3 * A$2$1 * A$3$2 −*
*A$1$1 * A$2$3 * A$3$2 −*
*A$1$2 * A$2$1 * A$3$3 −*
*A$1$3 * A$2$2 * A$3$1*
proof−
have *f123*: *finite* {*2*::*3*, *3*} *1* ∉ {*2*::*3*, *3*} **by** *auto*
have *f23*: *finite* {*3*::*3*} *2* ∉ {*3*::*3*} **by** *auto*

show *?thesis*
unfolding *det-def UNIV-3*
unfolding *setsum-over-permutations-insert*[*OF f123*]
unfolding *setsum-over-permutations-insert*[*OF f23*]

unfolding *permutes-sing*
apply (*simp add*: *sign-swap-id permutation-swap-id sign-compose sign-id swap-id-eq*)
apply (*simp add*: *arith-simps*(*31*)[*symmetric*] *del*: *arith-simps*(*31*))
by (*simp add*: *field-simps*)
qed

end

19 Convex-Euclidean-Space: Convex sets, functions and related things.

theory *Convex-Euclidean-Space*
imports *Topology-Euclidean-Space Convex*
begin

declare *vector-add-ldistrib*[*simp*] *vector-ssub-ldistrib*[*simp*] *vector-smult-assoc*[*simp*]
vector-smult-rneg[*simp*]
declare *vector-sadd-rdistrib*[*simp*] *vector-sub-rdistrib*[*simp*]

lemmas *vector-component-simps* = *vector-minus-component vector-smult-component*
vector-add-component vector-le-def Cart-lambda-beta basis-component vector-uminus-component

lemma *norm-not-0*:$(x::real\hat{}'n)\neq 0 \Longrightarrow norm\ x \neq 0$ **by** *auto*

lemma *setsum-delta-notmem*: **assumes** $x\notin s$
shows $setsum\ (\lambda y.\ if\ (y = x)\ then\ P\ x\ else\ Q\ y)\ s = setsum\ Q\ s$
$setsum\ (\lambda y.\ if\ (x = y)\ then\ P\ x\ else\ Q\ y)\ s = setsum\ Q\ s$
$setsum\ (\lambda y.\ if\ (y = x)\ then\ P\ y\ else\ Q\ y)\ s = setsum\ Q\ s$
$setsum\ (\lambda y.\ if\ (x = y)\ then\ P\ y\ else\ Q\ y)\ s = setsum\ Q\ s$
apply(*rule-tac* [!] *setsum-cong2*) **using** *assms* **by** *auto*

lemma *setsum-delta''*:
fixes $s::'a::real\text{-}vector\ set$ **assumes** *finite s*
shows $(\sum x\in s.\ (if\ y = x\ then\ f\ x\ else\ 0) *_R x) = (if\ y\in s\ then\ (f\ y) *_R y\ else\ 0)$
proof−
have $*:\bigwedge x\ y.\ (if\ y = x\ then\ f\ x\ else\ (0::real)) *_R x = (if\ x=y\ then\ (f\ x) *_R x\ else\ 0)$ **by** *auto*
show *?thesis* **unfolding** $*$ **using** *setsum-delta*[*OF assms*, *of y* $\lambda x.\ f\ x *_R x$] **by** *auto*
qed

lemma *not-disjointI*:$x\in A \Longrightarrow x\in B \Longrightarrow A \cap B \neq \{\}$ **by** *blast*

lemma *if-smult*:$(if\ P\ then\ x\ else\ (y::real)) *_R v = (if\ P\ then\ x *_R v\ else\ y *_R v)$ **by** *auto*

lemma *image-smult-interval*:$(\lambda x.\ m *_R (x::real\hat{}'n))\ `\ \{a..b\} =$
$(if\ \{a..b\} = \{\}\ then\ \{\}\ else\ if\ 0 \leq m\ then\ \{m *_R a..m *_R b\}\ else\ \{m *_R b..m *_R a\})$
using *image-affinity-interval*[*of m 0 a b*] **by** *auto*

lemma *dist-triangle-eq*:
fixes *x y z* :: *real* ^ -
shows $dist\ x\ z = dist\ x\ y + dist\ y\ z \longleftrightarrow norm\ (x - y) *_R (y - z) = norm\ (y - z) *_R (x - y)$
proof− **have** $*:x - y + (y - z) = x - z$ **by** *auto*
show *?thesis* **unfolding** *dist-norm norm-triangle-eq*[*of x − y y − z*, *unfolded smult-conv-scaleR* $*$]
by(*auto simp add*:*norm-minus-commute*) **qed**

lemma *norm-eqI*:$x = y \Longrightarrow norm\ x = norm\ y$ **by** *auto*
lemma *norm-minus-eqI*:$(x::real\hat{}'n) = - y \Longrightarrow norm\ x = norm\ y$ **by** *auto*

lemma *Min-grI*: **assumes** *finite A* $A \neq \{\}\ \forall a\in A.\ x < a$ **shows** $x < Min\ A$
unfolding *Min-gr-iff*[*OF assms*(*1*,*2*)] **using** *assms*(*3*) **by** *auto*

lemma *dimindex-ge-1*:$CARD(-::finite) \geq 1$
using *one-le-card-finite* **by** *auto*

lemma *real-dimindex-ge-1*:$real\ (CARD('n::finite)) \geq 1$
by(*metis dimindex-ge-1 real-eq-of-nat real-of-nat-1 real-of-nat-le-iff*)

lemma *real-dimindex-gt-0*:*real* $(CARD('n::finite)) > 0$ **apply**(*rule less-le-trans*[*OF - real-dimindex-ge-1*]) **by** *auto*

19.1 Affine set and affine hull.

definition
affine :: $'a::$*real-vector set* $\Rightarrow$ *bool* **where**
affine $s \longleftrightarrow (\forall x{\in}s.\ \forall y{\in}s.\ \forall u\ v.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in s)$

lemma *affine-alt*: *affine* $s \longleftrightarrow (\forall x{\in}s.\ \forall y{\in}s.\ \forall u::real.\ (1 - u) *_R x + u *_R y \in s)$
unfolding *affine-def* **by**(*metis eq-diff-eq'*)

lemma *affine-empty*[*intro*]: *affine* $\{\}$
unfolding *affine-def* **by** *auto*

lemma *affine-sing*[*intro*]: *affine* $\{x\}$
unfolding *affine-alt* **by** (*auto simp add*: *scaleR-left-distrib* [*symmetric*])

lemma *affine-UNIV*[*intro*]: *affine UNIV*
unfolding *affine-def* **by** *auto*

lemma *affine-Inter*: $(\forall s{\in}f.\ affine\ s) \Longrightarrow affine\ (\bigcap f)$
unfolding *affine-def* **by** *auto*

lemma *affine-Int*: *affine* $s \Longrightarrow$ *affine* $t \Longrightarrow$ *affine* $(s \cap t)$
unfolding *affine-def* **by** *auto*

lemma *affine-affine-hull*: *affine*(*affine hull s*)
unfolding *hull-def* **using** *affine-Inter*[*of* $\{t \in affine.\ s \subseteq t\}$]
unfolding *mem-def* **by** *auto*

lemma *affine-hull-eq*[*simp*]: (*affine hull* $s = s$) $\longleftrightarrow$ *affine* s
by (*metis affine-affine-hull hull-same mem-def*)

lemma *setsum-restrict-set''*: **assumes** *finite A*
shows *setsum f* $\{x \in A.\ P\ x\} = (\sum x{\in}A.\ if\ P\ x\ then\ f\ x\ else\ 0)$
unfolding *mem-def*[*of - P*, *symmetric*] **unfolding** *setsum-restrict-set'*[*OF assms*]
..

19.2 Some explicit formulations (from Lars Schewe).

lemma *affine*: **fixes** $V::'a::$*real-vector set*
shows *affine* $V \longleftrightarrow (\forall s\ u.\ finite\ s \wedge s \neq \{\} \wedge s \subseteq V \wedge setsum\ u\ s = 1 \longrightarrow (setsum\ (\lambda x.\ (u\ x) *_R x))\ s \in V)$
unfolding *affine-def* **apply** *rule* **apply**(*rule*, *rule*, *rule*) **apply**(*erule conjE*)+
defer apply(*rule*, *rule*, *rule*, *rule*, *rule*) **proof**−
fix *x y u v* **assume** *as*:$x \in V\ y \in V\ u + v = (1::real)$
$\forall s\ u.\ finite\ s \wedge s \neq \{\} \wedge s \subseteq V \wedge setsum\ u\ s = 1 \longrightarrow (\sum x{\in}s.\ u\ x *_R x) \in V$

thus $u *_R x + v *_R y \in V$ **apply**(*cases* $x=y$)
using *as*(*4*)[*THEN spec*[**where** $x=\{x,y\}$], *THEN spec*[**where** $x=\lambda w.$ *if* $w = x$ *then u else v*]] **and** *as*(*1−3*)
by(*auto simp add*: *scaleR-left-distrib*[*THEN sym*])
next
fix *s u* **assume** *as*:$\forall x\in V.\ \forall y\in V.\ \forall u\ v.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in V$
finite s $s \neq \{\}$ $s \subseteq V$ *setsum u s* $= (1::real)$
def $n \equiv$ *card s*
have *card s* $= 0 \vee$ *card s* $= 1 \vee$ *card s* $= 2 \vee$ *card s* > 2 **by** *auto*
thus $(\sum x\in s.\ u\ x *_R x) \in V$ **proof**(*auto simp only*: *disjE*)
assume *card s* $= 2$ **hence** *card s* $=$ *Suc* (*Suc 0*) **by** *auto*
then obtain *a b* **where** $s = \{a, b\}$ **unfolding** *card-Suc-eq* **by** *auto*
thus *?thesis* **using** *as*(*1*)[*THEN bspec*[**where** $x=a$], *THEN bspec*[**where** $x=b$]] **using** *as*(*4*,*5*)
by(*auto simp add*: *setsum-clauses*(*2*))
next assume *card s* > 2 **thus** *?thesis* **using** *as* **and** *n-def* **proof**(*induct n arbitrary*: *u s*)
case (*Suc n*) **fix** *s*::$'a$ *set* **and** *u*::$'a \Rightarrow$ *real*
assume *IA*:$\bigwedge u\ s.\ [\![2 <$ *card s*; $\forall x\in V.\ \forall y\in V.\ \forall u\ v.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in V$; *finite s*;
$s \neq \{\}$; $s \subseteq V$; *setsum u s* $= 1$; $n =$ *card s* $]\!] \Longrightarrow (\sum x\in s.\ u\ x *_R x) \in V$ **and**
as:*Suc n* $=$ *card s* $2 <$ *card s* $\forall x\in V.\ \forall y\in V.\ \forall u\ v.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in V$
finite s $s \neq \{\}$ $s \subseteq V$ *setsum u s* $= 1$
have $\exists x\in s.\ u\ x \neq 1$ **proof**(*rule-tac ccontr*)
assume $\neg\ (\exists x\in s.\ u\ x \neq 1)$ **hence** *setsum u s* $=$ *real-of-nat* (*card s*) **unfolding** *card-eq-setsum* **by** *auto*
thus *False* **using** *as*(*7*) **and** ‹*card s* > 2› **by** (*metis Numeral1-eq1-nat less-0-number-of less-int-code*(*15*) *less-nat-number-of not-less-iff-gr-or-eq of-nat-1 of-nat-eq-iff pos2 rel-simps*(*4*))
qed
then obtain *x* **where** *x*:$x\in s$ *u x* $\neq 1$ **by** *auto*

have *c*:*card* $(s - \{x\}) =$ *card s* $- 1$ **apply**(*rule card-Diff-singleton*) **using** ‹$x\in s$› *as*(*4*) **by** *auto*
have $*$:$s =$ *insert x* $(s - \{x\})$ *finite* $(s - \{x\})$ **using** ‹$x\in s$› **and** *as*(*4*) **by** *auto*
have $**$:*setsum u* $(s - \{x\}) = 1 - u\ x$
using *setsum-clauses*(*2*)[*OF* $*$(*2*), *of u x*, *unfolded* $*$(*1*)[*THEN sym*] *as*(*7*)] **by** *auto*
have $***$:*inverse* $(1 - u\ x) *$ *setsum u* $(s - \{x\}) = 1$ **unfolding** $**$ **using** ‹*u x* $\neq 1$› **by** *auto*
have $(\sum xa\in s - \{x\}.\ (inverse\ (1 - u\ x) * u\ xa) *_R xa) \in V$ **proof**(*cases card* $(s - \{x\}) > 2$)
case *True* **hence** $s - \{x\} \neq \{\}$ *card* $(s - \{x\}) = n$ **unfolding** *c* **and** *as*(*1*)[*symmetric*] **proof**(*rule-tac ccontr*)
assume $\neg\ s - \{x\} \neq \{\}$ **hence** *card* $(s - \{x\}) = 0$ **unfolding** *card-0-eq*[*OF* $*$(*2*)] **by** *simp*

thus *False* **using** *True* **by** *auto* **qed** *auto*
thus *?thesis* **apply**(*rule-tac IA*[*of* $s - \{x\}$ $\lambda y.$ (*inverse* (1 − *u x*) ∗ *u y*)])
unfolding *setsum-right-distrib*[*THEN sym*] **using** *as* **and** ∗∗∗ **and** *True* **by** *auto*
next case *False* **hence** *card* $(s - \{x\}) = Suc\ (Suc\ 0)$ **using** *as*(*2*) **and** *c* **by** *auto*
then obtain *a b* **where** $(s - \{x\}) = \{a, b\}$ $a \neq b$ **unfolding** *card-Suc-eq* **by** *auto*
thus *?thesis* **using** *as*(*3*)[*THEN bspec*[**where** *x*=*a*], *THEN bspec*[**where** *x*=*b*]]
using ∗∗∗ ∗(*2*) **and** ‹$s \subseteq V$› **unfolding** *setsum-right-distrib* **by**(*auto simp add*: *setsum-clauses*(*2*)) **qed**
thus $(\sum x{\in}s.\ u\ x\ *_R\ x) \in V$ **unfolding** *scaleR-scaleR*[*THEN sym*] **and** *scaleR-right.setsum* [*symmetric*]
apply(*subst* ∗) **unfolding** *setsum-clauses*(*2*)[*OF* ∗(*2*)]
using *as*(*3*)[*THEN bspec*[**where** *x*=*x*], *THEN bspec*[**where** *x*=(*inverse* (*1* − *u x*)) $*_R$ $(\sum xa{\in}s - \{x\}.\ u\ xa\ *_R\ xa)$],
THEN spec[**where** *x*=*u x*], *THEN spec*[**where** *x*=*1* − *u x*]] **and** *rev-subsetD*[*OF* ‹*x*∈*s*› ‹*s*⊆*V*›] **and** ‹*u x* ≠ *1*› **by** *auto*
qed *auto*
next assume *card s* = *1* **then obtain** *a* **where** *s*={*a*} **by**(*auto simp add*: *card-Suc-eq*)
thus *?thesis* **using** *as*(*4*,*5*) **by** *simp*
qed(*insert* ‹*s*≠{}› ‹*finite s*›, *auto*)
qed

lemma *affine-hull-explicit*:
affine hull p = {*y*. ∃ *s u*. *finite s* ∧ *s* ≠ {} ∧ *s* ⊆ *p* ∧ *setsum u s* = *1* ∧ *setsum* (λ*v*. (*u v*) $*_R$ *v*) *s* = *y*}
apply(*rule hull-unique*) **apply**(*subst subset-eq*) **prefer** *3* **apply** *rule* **unfolding** *mem-Collect-eq* **and** *mem-def*[*of* - *affine*]
apply (*erule exE*)+ **apply**(*erule conjE*)+ **prefer** *2* **apply** *rule* **proof**−
fix *x* **assume** *x*∈*p* **thus** ∃ *s u*. *finite s* ∧ *s* ≠ {} ∧ *s* ⊆ *p* ∧ *setsum u s* = *1* ∧ $(\sum v{\in}s.\ u\ v\ *_R\ v) = x$
apply(*rule-tac x*={*x*} **in** *exI*, *rule-tac x*=λ*x*. *1* **in** *exI*) **by** *auto*
next
fix *t x s u* **assume** *as*:*p* ⊆ *t affine t finite s s* ≠ {} *s* ⊆ *p setsum u s* = *1* $(\sum v{\in}s.$ *u v* $*_R$ *v*) = *x*
thus *x* ∈ *t* **using** *as*(*2*)[*unfolded affine*, *THEN spec*[**where** *x*=*s*], *THEN spec*[**where** *x*=*u*]] **by** *auto*
next
show *affine* {*y*. ∃ *s u*. *finite s* ∧ *s* ≠ {} ∧ *s* ⊆ *p* ∧ *setsum u s* = *1* ∧ $(\sum v{\in}s.$ *u v* $*_R$ *v*) = *y*} **unfolding** *affine-def*
apply(*rule*,*rule*,*rule*,*rule*,*rule*) **unfolding** *mem-Collect-eq* **proof**−
fix *u v* ::*real* **assume** *uv*:*u* + *v* = *1*
fix *x* **assume** ∃ *s u*. *finite s* ∧ *s* ≠ {} ∧ *s* ⊆ *p* ∧ *setsum u s* = *1* ∧ $(\sum v{\in}s.$ *u v* $*_R$ *v*) = *x*
then obtain *sx ux* **where** *x*:*finite sx sx* ≠ {} *sx* ⊆ *p setsum ux sx* = *1* $(\sum v{\in}sx.$ *ux v* $*_R$ *v*) = *x* **by** *auto*

fix *y* **assume** $\exists s\ u.$ *finite* $s \wedge s \neq \{\} \wedge s \subseteq p \wedge$ *setsum* $u\ s = 1 \wedge (\sum v{\in}s.\ u\ v *_R v) = y$
then obtain *sy uy* **where** *y*:*finite sy* $sy \neq \{\}$ $sy \subseteq p$ *setsum uy sy* $= 1$ $(\sum v{\in}sy.\ uy\ v *_R v) = y$ **by** *auto*
have *xy*:*finite* $(sx \cup sy)$ **using** $x(1)$ $y(1)$ **by** *auto*
have $**$:$(sx \cup sy) \cap sx = sx$ $(sx \cup sy) \cap sy = sy$ **by** *auto*
show $\exists s\ ua.$ *finite* $s \wedge s \neq \{\} \wedge s \subseteq p \wedge$ *setsum ua* $s = 1 \wedge (\sum v{\in}s.\ ua\ v *_R v) = u *_R x + v *_R y$
apply(*rule-tac x*=*sx* $\cup$ *sy* **in** *exI*)
apply(*rule-tac x*=$\lambda a.$ (*if a*$\in$*sx then u* $*$ *ux a else 0*) + (*if a*$\in$*sy then v* $*$ *uy a else 0*) **in** *exI*)
unfolding *scaleR-left-distrib setsum-addf if-smult scaleR-zero-left* $**$ *setsum-restrict-set*[*OF xy*, *THEN sym*]
unfolding *scaleR-scaleR*[*THEN sym*] *scaleR-right.setsum* [*symmetric*] **and** *setsum-right-distrib*[*THEN sym*]
unfolding *x y* **using** $x(1{-}3)$ $y(1{-}3)$ *uv* **by** *simp* **qed qed**

lemma *affine-hull-finite*:
assumes *finite s*
shows *affine hull s* $= \{y.\ \exists u.$ *setsum u s* $= 1 \wedge$ *setsum* $(\lambda v.\ u\ v *_R v)$ *s* $= y\}$
unfolding *affine-hull-explicit* **and** *expand-set-eq* **and** *mem-Collect-eq* **apply**(*rule*,*rule*)
apply(*erule exE*)+ **apply**(*erule conjE*)+ **defer apply**(*erule exE*) **apply**(*erule conjE*) **proof**−
fix *x u* **assume** *setsum u s* $= 1$ $(\sum v{\in}s.\ u\ v *_R v) = x$
thus $\exists$ *sa u. finite sa* $\wedge \neg\ (\forall x.\ (x \in sa) = (x \in \{\})) \wedge sa \subseteq s \wedge$ *setsum u sa* $= 1 \wedge (\sum v{\in}sa.\ u\ v *_R v) = x$
apply(*rule-tac x*=*s* **in** *exI*, *rule-tac x*=*u* **in** *exI*) **using** *assms* **by** *auto*
next
fix *x t u* **assume** $t \subseteq s$ **hence** $*$:$s \cap t = t$ **by** *auto*
assume *finite t* $\neg\ (\forall x.\ (x \in t) = (x \in \{\}))$ *setsum u t* $= 1$ $(\sum v{\in}t.\ u\ v *_R v) = x$
thus $\exists u.$ *setsum u s* $= 1 \wedge (\sum v{\in}s.\ u\ v *_R v) = x$ **apply**(*rule-tac x*=$\lambda x.$ *if x*$\in$*t then u x else 0* **in** *exI*)
unfolding *if-smult scaleR-zero-left* **and** *setsum-restrict-set*[*OF assms*, *THEN sym*] **and** $*$ **by** *auto* **qed**

19.3 Stepping theorems and hence small special cases.

lemma *affine-hull-empty*[*simp*]: *affine hull* $\{\} = \{\}$
apply(*rule hull-unique*) **unfolding** *mem-def* **by** *auto*

lemma *affine-hull-finite-step*:
fixes *y* :: *'a*::*real-vector*
shows $(\exists u.$ *setsum u* $\{\} = w \wedge$ *setsum* $(\lambda x.\ u\ x *_R x)\ \{\} = y) \longleftrightarrow w = 0 \wedge y = 0$ (**is** *?th1*)
finite s $\Longrightarrow$ $(\exists u.$ *setsum u* (*insert a s*) $= w \wedge$ *setsum* $(\lambda x.\ u\ x *_R x)$ (*insert a s*) $= y) \longleftrightarrow$
$(\exists v\ u.$ *setsum u s* $= w - v \wedge$ *setsum* $(\lambda x.\ u\ x *_R x)\ s = y - v *_R$

a) (**is** *?as* $\Longrightarrow$ (*?lhs* = *?rhs*))
proof−
show *?th1* **by** *simp*
assume *?as*
{ assume *?lhs*
then obtain *u* **where** *u*:*setsum u* (*insert a s*) = *w* ∧ ($\sum$ *x*∈*insert a s*. *u x* $*_R$ *x*) = *y* **by** *auto*
have *?rhs* **proof**(*cases a*∈*s*)
case *True* **hence** ∗:*insert a s* = *s* **by** *auto*
show *?thesis* **using** *u*[*unfolded* ∗] **apply**(*rule-tac x*=*0* **in** *exI*) **by** *auto*
next
case *False* **thus** *?thesis* **apply**(*rule-tac x*=*u a* **in** *exI*) **using** *u* **and** ‹*?as*› **by** *auto*
qed **} moreover**
{ assume *?rhs*
then obtain *v u* **where** *vu*:*setsum u s* = *w* − *v* ($\sum$ *x*∈*s*. *u x* $*_R$ *x*) = *y* − *v* $*_R$ *a* **by** *auto*
have ∗:$\bigwedge$*x M*. (*if x* = *a then v else M*) $*_R$ *x* = (*if x* = *a then v* $*_R$ *x else M* $*_R$ *x*) **by** *auto*
have *?lhs* **proof**(*cases a*∈*s*)
case *True* **thus** *?thesis*
apply(*rule-tac x*=λ*x*. (*if x*=*a then v else 0*) + *u x* **in** *exI*)
unfolding *setsum-clauses*(*2*)[*OF* ‹*?as*›] **apply** *simp*
unfolding *scaleR-left-distrib* **and** *setsum-addf*
unfolding *vu* **and** ∗ **and** *scaleR-zero-left*
by (*auto simp add*: *setsum-delta*[*OF* ‹*?as*›])
next
case *False*
hence ∗∗:$\bigwedge$*x*. *x* ∈ *s* $\Longrightarrow$ *u x* = (*if x* = *a then v else u x*)
$\bigwedge$*x*. *x* ∈ *s* $\Longrightarrow$ *u x* $*_R$ *x* = (*if x* = *a then v* $*_R$ *x else u x* $*_R$ *x*) **by** *auto*
from *False* **show** *?thesis*
apply(*rule-tac x*=λ*x*. *if x*=*a then v else u x* **in** *exI*)
unfolding *setsum-clauses*(*2*)[*OF* ‹*?as*›] **and** ∗ **using** *vu*
using *setsum-cong2*[*of s* λ*x*. *u x* $*_R$ *x* λ*x*. *if x* = *a then v* $*_R$ *x else u x* $*_R$ *x*, *OF* ∗∗(*2*)]
using *setsum-cong2*[*of s u* λ*x*. *if x* = *a then v else u x*, *OF* ∗∗(*1*)] **by** *auto*
qed **}**
ultimately show *?lhs* = *?rhs* **by** *blast*
qed

lemma *affine-hull-2*:
fixes *a b* :: *'a*::*real-vector*
shows *affine hull* {*a*,*b*} = {*u* $*_R$ *a* + *v* $*_R$ *b*| *u v*. (*u* + *v* = *1*)} (**is** *?lhs* = *?rhs*)
proof−
have ∗:$\bigwedge$*x y z*. *z* = *x* − *y* ⟷ *y* + *z* = (*x*::*real*)
$\bigwedge$*x y z*. *z* = *x* − *y* ⟷ *y* + *z* = (*x*::*'a*) **by** *auto*
have *?lhs* = {*y*. ∃ *u*. *setsum u* {*a*, *b*} = *1* ∧ ($\sum$ *v*∈{*a*, *b*}. *u v* $*_R$ *v*) = *y*}
using *affine-hull-finite*[*of* {*a*,*b*}] **by** *auto*

also have ... = $\{y.\ \exists v\ u.\ u\ b = 1 - v \wedge u\ b *_R b = y - v *_R a\}$
by(*simp add*: *affine-hull-finite-step*(*2*)[*of* $\{b\}$ *a*])
also have ... = *?rhs* **unfolding** $*$ **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *affine-hull-3*:
fixes *a b c* :: *'a*::*real-vector*
shows *affine hull* $\{a,b,c\} = \{ u *_R a + v *_R b + w *_R c |\ u\ v\ w.\ u + v + w = 1\}$ (**is** *?lhs* = *?rhs*)
proof−
have $*$:$\bigwedge x\ y\ z.\ z = x - y \longleftrightarrow y + z = (x::real)$
$\bigwedge x\ y\ z.\ z = x - y \longleftrightarrow y + z = (x::'a)$ **by** *auto*
show *?thesis* **apply**(*simp add*: *affine-hull-finite affine-hull-finite-step*)
unfolding $*$ **apply** *auto*
apply(*rule-tac x*=*v* **in** *exI*) **apply**(*rule-tac x*=*va* **in** *exI*) **apply** *auto*
apply(*rule-tac x*=*u* **in** *exI*) **by**(*auto intro*!: *exI*)
qed

19.4 Some relations between affine hull and subspaces.

lemma *affine-hull-insert-subset-span*:
fixes *a* :: *real* ^ -
shows *affine hull* (*insert a s*) $\subseteq \{a + v |\ v\ .\ v \in span\ \{x - a \mid x\ .\ x \in s\}\}$
unfolding *subset-eq Ball-def* **unfolding** *affine-hull-explicit span-explicit mem-Collect-eq smult-conv-scaleR*
apply(*rule*,*rule*) **apply**(*erule exE*)+ **apply**(*erule conjE*)+ **proof**−
fix *x t u* **assume** *as*:*finite t* $t \neq \{\}$ $t \subseteq$ *insert a s* $\sum v \in t.\ u\ v *_R v) = x$
have $(\lambda x.\ x - a)$ ' $(t - \{a\}) \subseteq \{x - a\ |x.\ x \in s\}$ **using** *as*(*3*) **by** *auto*
thus $\exists v.\ x = a + v \wedge (\exists S\ u.$ *finite* $S \wedge S \subseteq \{x - a\ |x.\ x \in s\} \wedge (\sum v \in S.\ u\ v *_R v) = v)$
apply(*rule-tac x*=*x* − *a* **in** *exI*)
apply (*rule conjI*, *simp*)
apply(*rule-tac x*=$(\lambda x.\ x - a)$ ' $(t - \{a\})$ **in** *exI*)
apply(*rule-tac x*=$\lambda x.\ u\ (x + a)$ **in** *exI*)
apply (*rule conjI*) **using** *as*(*1*) **apply** *simp*
apply (*erule conjI*)
using *as*(*1*)
apply (*simp add*: *setsum-reindex*[*unfolded inj-on-def*] *scaleR-right-diff-distrib setsum-subtractf scaleR-left*.*setsum*[*THEN sym*] *setsum-diff1 scaleR-left-diff-distrib*)
unfolding *as* **by** *simp* **qed**

lemma *affine-hull-insert-span*:
fixes *a* :: *real* ^ -
assumes $a \notin s$
shows *affine hull* (*insert a s*) =
$\{a + v \mid v\ .\ v \in span\ \{x - a \mid x.\ \ x \in s\}\}$
apply(*rule*, *rule affine-hull-insert-subset-span*) **unfolding** *subset-eq Ball-def*

unfolding *affine-hull-explicit* **and** *mem-Collect-eq* **proof**(*rule*,*rule*,*erule exE*,*erule conjE*)
fix *y v* **assume** $y = a + v\ v \in span\ \{x - a\ |x.\ x \in s\}$
then obtain *t u* **where** *obt*:*finite t* $t \subseteq \{x - a\ |x.\ x \in s\}\ a + (\sum v{\in}t.\ u\ v *_R v) = y$ **unfolding** *span-explicit smult-conv-scaleR* **by** *auto*
def $f \equiv (\lambda x.\ x + a)\ `\ t$
have *f*:*finite f* $f \subseteq s\ (\sum v{\in}f.\ u\ (v - a) *_R (v - a)) = y - a$ **unfolding** *f-def* **using** *obt*
by(*auto simp add*: *setsum-reindex*[*unfolded inj-on-def*])
have $*$:$f \cap \{a\} = \{\}\ f \cap - \{a\} = f$ **using** *f*(*2*) *assms* **by** *auto*
show $\exists sa\ u.$ *finite sa* $\wedge$ *sa* $\neq \{\} \wedge$ *sa* $\subseteq$ *insert a s* $\wedge$ *setsum u sa* $= 1 \wedge (\sum v{\in}sa.\ u\ v *_R v) = y$
apply(*rule-tac x*=*insert a f* **in** *exI*)
apply(*rule-tac x*=$\lambda x.$ *if x*=*a then 1* $-$ *setsum* $(\lambda x.\ u\ (x - a))$ *f else u* $(x - a)$ **in** *exI*)
using *assms* **and** *f* **unfolding** *setsum-clauses*(*2*)[*OF f*(*1*)] **and** *if-smult*
unfolding *setsum-cases*[*OF f*(*1*), *of* $\lambda x.\ x = a$]
by (*auto simp add*: *setsum-subtractf scaleR-left.setsum algebra-simps* $*$) **qed**

lemma *affine-hull-span*:
fixes *a* :: *real* ^ -
assumes $a \in s$
shows *affine hull s* $= \{a + v \mid v.\ v \in span\ \{x - a \mid x.\ x \in s - \{a\}\}\}$
using *affine-hull-insert-span*[*of a s* $-$ $\{a\}$, *unfolded insert-Diff*[*OF assms*]] **by** *auto*

19.5 Cones.

definition
cone :: *'a*::*real-vector set* $\Rightarrow$ *bool* **where**
cone s $\longleftrightarrow$ $(\forall x{\in}s.\ \forall c{\geq}0.\ (c *_R x) \in s)$

lemma *cone-empty*[*intro*, *simp*]: *cone* $\{\}$
unfolding *cone-def* **by** *auto*

lemma *cone-univ*[*intro*, *simp*]: *cone UNIV*
unfolding *cone-def* **by** *auto*

lemma *cone-Inter*[*intro*]: $(\forall s{\in}f.\ cone\ s) \Longrightarrow cone(\bigcap f)$
unfolding *cone-def* **by** *auto*

19.6 Conic hull.

lemma *cone-cone-hull*: *cone* (*cone hull s*)
unfolding *hull-def* **using** *cone-Inter*[*of* $\{t \in conic.\ s \subseteq t\}$]
by (*auto simp add*: *mem-def*)

lemma *cone-hull-eq*: (*cone hull s* $= s$) $\longleftrightarrow$ *cone s*
apply(*rule hull-eq*[*unfolded mem-def*])
using *cone-Inter* **unfolding** *subset-eq* **by** (*auto simp add*: *mem-def*)

19.7 Affine dependence and consequential theorems (from Lars Schewe).

definition
affine-dependent :: *'a*::*real-vector set* ⇒ *bool* **where**
affine-dependent s ⟷ (∃ *x*∈*s*. *x* ∈ (*affine hull* (*s* − {*x*})))

lemma *affine-dependent-explicit*:
affine-dependent p ⟷
(∃ *s u*. *finite s* ∧ *s* ⊆ *p* ∧ *setsum u s* = *0* ∧
(∃ *v*∈*s*. *u v* ≠ *0*) ∧ *setsum* (λ*v*. *u v* ∗*R v*) *s* = *0*)
unfolding *affine-dependent-def affine-hull-explicit mem-Collect-eq* **apply**(*rule*)
apply(*erule bexE*,*erule exE*,*erule exE*) **apply**(*erule conjE*)+ **defer apply**(*erule exE*,*erule exE*) **apply**(*erule conjE*)+ **apply**(*erule bexE*)
proof−
fix *x s u* **assume** *as*:*x* ∈ *p finite s s* ≠ {} *s* ⊆ *p* − {*x*} *setsum u s* = *1* (∑ *v*∈*s*. *u v* ∗*R v*) = *x*
have *x*∉*s* **using** *as*(*1*,*4*) **by** *auto*
show ∃ *s u*. *finite s* ∧ *s* ⊆ *p* ∧ *setsum u s* = *0* ∧ (∃ *v*∈*s*. *u v* ≠ *0*) ∧ (∑ *v*∈*s*. *u v* ∗*R v*) = *0*
apply(*rule-tac x*=*insert x s* **in** *exI*, *rule-tac x*=λ*v*. *if v* = *x then* − *1 else u v* **in** *exI*)
unfolding *if-smult* **and** *setsum-clauses*(*2*)[*OF as*(*2*)] **and** *setsum-delta-notmem*[*OF* ‹*x*∉*s*›] **and** *as* **using** *as* **by** *auto*
next
fix *s u v* **assume** *as*:*finite s s* ⊆ *p setsum u s* = *0* (∑ *v*∈*s*. *u v* ∗*R v*) = *0 v* ∈ *s u v* ≠ *0*
have *s* ≠ {*v*} **using** *as*(*3*,*6*) **by** *auto*
thus ∃ *x*∈*p*. ∃ *s u*. *finite s* ∧ *s* ≠ {} ∧ *s* ⊆ *p* − {*x*} ∧ *setsum u s* = *1* ∧ (∑ *v*∈*s*. *u v* ∗*R v*) = *x*
apply(*rule-tac x*=*v* **in** *bexI*, *rule-tac x*=*s* − {*v*} **in** *exI*, *rule-tac x*=λ*x*. − (*1* / *u v*) ∗ *u x* **in** *exI*)
unfolding *scaleR-scaleR*[*THEN sym*] **and** *scaleR-right.setsum* [*symmetric*] **unfolding** *setsum-right-distrib*[*THEN sym*] **and** *setsum-diff1*[*OF as*(*1*)] **using** *as* **by** *auto*
qed

lemma *affine-dependent-explicit-finite*:
fixes *s* :: *'a*::*real-vector set* **assumes** *finite s*
shows *affine-dependent s* ⟷ (∃ *u*. *setsum u s* = *0* ∧ (∃ *v*∈*s*. *u v* ≠ *0*) ∧ *setsum* (λ*v*. *u v* ∗*R v*) *s* = *0*)
(**is** *?lhs* = *?rhs*)
proof
have ∗:⋀*vt u v*. (*if vt then u v else 0*) ∗*R v* = (*if vt then* (*u v*) ∗*R v else* (*0*::*'a*)) **by** *auto*
assume *?lhs*
then obtain *t u v* **where** *finite t t* ⊆ *s setsum u t* = *0 v*∈*t u v* ≠ *0* (∑ *v*∈*t*. *u v* ∗*R v*) = *0*
unfolding *affine-dependent-explicit* **by** *auto*
thus *?rhs* **apply**(*rule-tac x*=λ*x*. *if x*∈*t then u x else 0* **in** *exI*)

apply *auto* **unfolding** $*$ **and** *setsum-restrict-set*[*OF assms*, *THEN sym*]
unfolding *Int-absorb1*[*OF* ‹$t \subseteq s$›] **by** *auto*
next
assume *?rhs*
then obtain $u\ v$ **where** *setsum* $u\ s = 0$ $v \in s\ u\ v \neq 0$ $(\sum v \in s.\ u\ v *_R v) = 0$
by *auto*
thus *?lhs* **unfolding** *affine-dependent-explicit* **using** *assms* **by** *auto*
qed

19.8 A general lemma.

lemma *convex-connected*:
fixes s :: *'a*::*real-normed-vector set*
assumes *convex s* **shows** *connected s*
proof−
{ fix *e1 e2* **assume** *as*:*open e1 open e2 e1* ∩ *e2* ∩ $s = \{\}$ $s \subseteq e1 \cup e2$
assume $e1 \cap s \neq \{\}$ $e2 \cap s \neq \{\}$
then obtain *x1 x2* **where** *x1*:*x1*∈*e1 x1*∈*s* **and** *x2*:*x2*∈*e2 x2*∈*s* **by** *auto*
hence *n*:*norm* $(x1 - x2) > 0$ **unfolding** *zero-less-norm-iff* **using** *as*(*3*) **by**
auto

{ fix *x e*::*real* **assume** *as*:$0 \leq x\ x \leq 1\ 0 < e$
{ fix *y* **have** $*$:$(1 - x) *_R x1 + x *_R x2 - ((1 - y) *_R x1 + y *_R x2) =$
$(y - x) *_R x1 - (y - x) *_R x2$
by (*simp add*: *algebra-simps*)
assume $|y - x| < e\ /\ norm\ (x1 - x2)$
hence $norm\ ((1 - x) *_R x1 + x *_R x2 - ((1 - y) *_R x1 + y *_R x2)) < e$
unfolding $*$ **and** *scaleR-right-diff-distrib*[*THEN sym*]
unfolding *less-divide-eq* **using** *n* **by** *auto* **}**
hence $\exists d{>}0.\ \forall y.\ |y - x| < d \longrightarrow norm\ ((1 - x) *_R x1 + x *_R x2 - ((1$
$- y) *_R x1 + y *_R x2)) < e$
apply(*rule-tac x*=*e* / *norm* (*x1* − *x2*) **in** *exI*) **using** *as*
apply *auto* **unfolding** *zero-less-divide-iff* **using** *n* **by** *simp* **}** **note** $*$ =
this

have $\exists x{\geq}0.\ x \leq 1 \wedge (1 - x) *_R x1 + x *_R x2 \notin e1 \wedge (1 - x) *_R x1 + x$
$*_R x2 \notin e2$
apply(*rule connected-real-lemma*) **apply** (*simp add*: ‹*x1*∈*e1*› ‹*x2*∈*e2*› *dist-commute*)+
using $*$ **apply**(*simp add*: *dist-norm*)
using *as*(*1*,*2*)[*unfolded open-dist*] **apply** *simp*
using *as*(*1*,*2*)[*unfolded open-dist*] **apply** *simp*
using *assms*[*unfolded convex-alt*, *THEN bspec*[**where** *x*=*x1*], *THEN bspec*[**where**
x=*x2*]] **using** *x1 x2*
using *as*(*3*) **by** *auto*
then obtain *x* **where** $x{\geq}0\ x{\leq}1\ (1 - x) *_R x1 + x *_R x2 \notin e1\ \ (1 - x) *_R$
$x1 + x *_R x2 \notin e2$ **by** *auto*
hence *False* **using** *as*(*4*)
using *assms*[*unfolded convex-alt*, *THEN bspec*[**where** *x*=*x1*], *THEN bspec*[**where**
x=*x2*]]

using *x1(2) x2(2)* **by** *auto* **}**
thus *?thesis* **unfolding** *connected-def* **by** *auto*
qed

19.9 One rather trivial consequence.

lemma *connected-UNIV*[*intro*]: *connected* (*UNIV* :: *'a*::*real-normed-vector set*)
by(*simp add*: *convex-connected convex-UNIV*)

19.10 Balls, being convex, are connected.

lemma *convex-box*:
assumes $\bigwedge i$. *convex* {*x*. *P i x*}
shows *convex* {*x*. $\forall i$. *P i* (*x*$*i*)}
using *assms* **unfolding** *convex-def* **by** *auto*

lemma *convex-positive-orthant*: *convex* {*x*::*real^'n*. ($\forall i$. *0* $\leq$ *x*$*i*)}
by (*rule convex-box*) (*simp add*: *atLeast-def*[*symmetric*] *convex-real-interval*)

lemma *convex-local-global-minimum*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *0*<*e convex-on s f ball x e* $\subseteq$ *s* $\forall$ *y*$\in$*ball x e*. *f x* $\leq$ *f y*
shows $\forall$ *y*$\in$*s*. *f x* $\leq$ *f y*
proof(*rule ccontr*)
have *x*$\in$*s* **using** *assms*(*1*,*3*) **by** *auto*
assume $\neg$ ($\forall$ *y*$\in$*s*. *f x* $\leq$ *f y*)
then obtain *y* **where** *y*$\in$*s* **and** *y*:*f x* > *f y* **by** *auto*
hence *xy*:*0* < *dist x y* **by** (*auto simp add*: *dist-nz*[*THEN sym*])

then obtain *u* **where** *0* < *u u* $\leq$ *1* **and** *u*:*u* < *e* / *dist x y*
using *real-lbound-gt-zero*[*of 1 e* / *dist x y*] **using** *xy* ‹*e*>*0*› **and** *divide-pos-pos*[*of e dist x y*] **by** *auto*
hence *f* ((*1*−*u*) $*_R$ *x* + *u* $*_R$ *y*) $\leq$ (*1*−*u*) * *f x* + *u* * *f y* **using** ‹*x*$\in$*s*› ‹*y*$\in$*s*›
using *assms*(*2*)[*unfolded convex-on-def*, *THEN bspec*[**where** *x*=*x*], *THEN bspec*[**where** *x*=*y*], *THEN spec*[**where** *x*=*1*−*u*]] **by** *auto*
moreover
have *:*x* − ((*1* − *u*) $*_R$ *x* + *u* $*_R$ *y*) = *u* $*_R$ (*x* − *y*) **by** (*simp add*: *algebra-simps*)
have (*1* − *u*) $*_R$ *x* + *u* $*_R$ *y* $\in$ *ball x e* **unfolding** *mem-ball dist-norm* **unfolding** * **and** *norm-scaleR* **and** *abs-of-pos*[*OF* ‹*0*<*u*›] **unfolding** *dist-norm*[*THEN sym*]
using *u* **unfolding** *pos-less-divide-eq*[*OF xy*] **by** *auto*
hence *f x* $\leq$ *f* ((*1* − *u*) $*_R$ *x* + *u* $*_R$ *y*) **using** *assms*(*4*) **by** *auto*
ultimately show *False* **using** *mult-strict-left-mono*[*OF y* ‹*u*>*0*›] **unfolding** *left-diff-distrib* **by** *auto*
qed

lemma *convex-ball*:
fixes *x* :: *'a*::*real-normed-vector*
shows *convex* (*ball x e*)
proof(*auto simp add*: *convex-def*)
fix *y z* **assume** *yz*:*dist x y* < *e dist x z* < *e*

fix *u v* ::*real* **assume** *uv*:*0* ≤ *u 0* ≤ *v u* + *v* = *1*
have *dist x* (*u* ∗$_R$ *y* + *v* ∗$_R$ *z*) ≤ *u* ∗ *dist x y* + *v* ∗ *dist x z* **using** *uv yz*
using *convex-distance*[*of ball x e x*, *unfolded convex-on-def*, *THEN bspec*[**where** *x*=*y*], *THEN bspec*[**where** *x*=*z*]] **by** *auto*
thus *dist x* (*u* ∗$_R$ *y* + *v* ∗$_R$ *z*) < *e* **using** *convex-bound-lt*[*OF yz uv*] **by** *auto*
qed

lemma *convex-cball*:
fixes *x* :: *'a*::*real-normed-vector*
shows *convex*(*cball x e*)
proof(*auto simp add*: *convex-def Ball-def*)
fix *y z* **assume** *yz*:*dist x y* ≤ *e dist x z* ≤ *e*
fix *u v* ::*real* **assume** *uv*: *0* ≤ *u 0* ≤ *v u* + *v* = *1*
have *dist x* (*u* ∗$_R$ *y* + *v* ∗$_R$ *z*) ≤ *u* ∗ *dist x y* + *v* ∗ *dist x z* **using** *uv yz*
using *convex-distance*[*of cball x e x*, *unfolded convex-on-def*, *THEN bspec*[**where** *x*=*y*], *THEN bspec*[**where** *x*=*z*]] **by** *auto*
thus *dist x* (*u* ∗$_R$ *y* + *v* ∗$_R$ *z*) ≤ *e* **using** *convex-bound-le*[*OF yz uv*] **by** *auto*
qed

lemma *connected-ball*:
fixes *x* :: *'a*::*real-normed-vector*
shows *connected* (*ball x e*)
using *convex-connected convex-ball* **by** *auto*

lemma *connected-cball*:
fixes *x* :: *'a*::*real-normed-vector*
shows *connected*(*cball x e*)
using *convex-connected convex-cball* **by** *auto*

19.11 Convex hull.

lemma *convex-convex-hull*: *convex*(*convex hull s*)
unfolding *hull-def* **using** *convex-Inter*[*of* {*t*∈*convex*. *s*⊆*t*}]
unfolding *mem-def* **by** *auto*

lemma *convex-hull-eq*: *convex hull s* = *s* ⟷ *convex s*
by (*metis convex-convex-hull hull-same mem-def*)

lemma *bounded-convex-hull*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *bounded s* **shows** *bounded*(*convex hull s*)
proof− **from** *assms* **obtain** *B* **where** *B*:∀ *x*∈*s*. *norm x* ≤ *B* **unfolding** *bounded-iff*
by *auto*
show *?thesis* **apply**(*rule bounded-subset*[*OF bounded-cball*, *of - 0 B*])
unfolding *subset-hull*[*unfolded mem-def*, *of convex*, *OF convex-cball*]
unfolding *subset-eq mem-cball dist-norm* **using** *B* **by** *auto* **qed**

lemma *finite-imp-bounded-convex-hull*:
fixes *s* :: *'a*::*real-normed-vector set*

shows *finite s* $\Longrightarrow$ *bounded*(*convex hull s*)
using *bounded-convex-hull finite-imp-bounded* **by** *auto*

19.12 Stepping theorems for convex hulls of finite sets.

lemma *convex-hull-empty*[*simp*]: *convex hull* {} = {}
apply(*rule hull-unique*) **unfolding** *mem-def* **by** *auto*

lemma *convex-hull-singleton*[*simp*]: *convex hull* {*a*} = {*a*}
apply(*rule hull-unique*) **unfolding** *mem-def* **by** *auto*

lemma *convex-hull-insert*:
fixes *s* :: *'a*::*real-vector set*
assumes *s* ≠ {}
shows *convex hull* (*insert a s*) = {*x*. ∃ *u*≥*0*. ∃ *v*≥*0*. ∃ *b*. (*u* + *v* = *1*) ∧
b ∈ (*convex hull s*) ∧ (*x* = *u* ∗*R* *a* + *v* ∗*R* *b*)} (**is** *?xyz* = *?hull*)
apply(*rule*,*rule hull-minimal*,*rule*) **unfolding** *mem-def*[*of - convex*] **and** *insert-iff*
prefer *3* **apply** *rule* **proof**−
fix *x* **assume** *x*:*x* = *a* ∨ *x* ∈ *s*
thus *x*∈*?hull* **apply** *rule* **unfolding** *mem-Collect-eq* **apply**(*rule-tac x*=*1* **in** *exI*)
defer
apply(*rule-tac x*=*0* **in** *exI*) **using** *assms hull-subset*[*of s convex*] **by** *auto*
next
fix *x* **assume** *x*∈*?hull*
then obtain *u v b* **where** *obt*:*u*≥*0 v*≥*0 u* + *v* = *1 b* ∈ *convex hull s x* = *u* ∗*R* *a* + *v* ∗*R* *b* **by** *auto*
have *a*∈*convex hull insert a s b*∈*convex hull insert a s*
using *hull-mono*[*of s insert a s convex*] *hull-mono*[*of* {*a*} *insert a s convex*] **and** *obt*(*4*) **by** *auto*
thus *x*∈ *convex hull insert a s* **unfolding** *obt*(*5*) **using** *convex-convex-hull*[*of insert a s*, *unfolded convex-def*]
apply(*erule-tac x*=*a* **in** *ballE*) **apply**(*erule-tac x*=*b* **in** *ballE*) **apply**(*erule-tac x*=*u* **in** *allE*) **using** *obt* **by** *auto*
next
show *convex ?hull* **unfolding** *convex-def* **apply**(*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*rule*)
proof−
fix *x y u v* **assume** *as*:(*0*::*real*) ≤ *u 0* ≤ *v u* + *v* = *1 x*∈*?hull y*∈*?hull*
from *as*(*4*) **obtain** *u1 v1 b1* **where** *obt1*:*u1*≥*0 v1*≥*0 u1* + *v1* = *1 b1* ∈ *convex hull s x* = *u1* ∗*R* *a* + *v1* ∗*R* *b1* **by** *auto*
from *as*(*5*) **obtain** *u2 v2 b2* **where** *obt2*:*u2*≥*0 v2*≥*0 u2* + *v2* = *1 b2* ∈ *convex hull s y* = *u2* ∗*R* *a* + *v2* ∗*R* *b2* **by** *auto*
have ∗:⋀(*x*::*'a*) *s1 s2*. *x* − *s1* ∗*R* *x* − *s2* ∗*R* *x* = ((*1*::*real*) − (*s1* + *s2*)) ∗*R* *x* **by** (*auto simp add*: *algebra-simps*)
have ∃ *b* ∈ *convex hull s*. *u* ∗*R* *x* + *v* ∗*R* *y* = (*u* ∗ *u1*) ∗*R* *a* + (*v* ∗ *u2*) ∗*R* *a* + (*b* − (*u* ∗ *u1*) ∗*R* *b* − (*v* ∗ *u2*) ∗*R* *b*)
proof(*cases u* ∗ *v1* + *v* ∗ *v2* = *0*)
have ∗:⋀(*x*::*'a*) *s1 s2*. *x* − *s1* ∗*R* *x* − *s2* ∗*R* *x* = ((*1*::*real*) − (*s1* + *s2*)) ∗*R* *x* **by** (*auto simp add*: *algebra-simps*)

case *True* **hence** $**$:$u * v1 = 0$ $v * v2 = 0$
using *mult-nonneg-nonneg*[*OF* ‹$u \geq 0$› ‹$v1 \geq 0$›] *mult-nonneg-nonneg*[*OF* ‹$v \geq 0$› ‹$v2 \geq 0$›] **by** *arith*+
hence $u * u1 + v * u2 = 1$ **using** *as*(3) *obt1*(3) *obt2*(3) **by** *auto*
thus *?thesis* **unfolding** *obt1*(5) *obt2*(5) $*$ **using** *assms hull-subset*[*of s convex*] **by**(*auto simp add*: $**$ *scaleR-right-distrib*)
next
have $1 - (u * u1 + v * u2) = (u + v) - (u * u1 + v * u2)$ **using** *as*(3) *obt1*(3) *obt2*(3) **by** (*auto simp add*: *field-simps*)
also have $\ldots = u * (v1 + u1 - u1) + v * (v2 + u2 - u2)$ **using** *as*(3) *obt1*(3) *obt2*(3) **by** (*auto simp add*: *field-simps*)
also have $\ldots = u * v1 + v * v2$ **by** *simp* **finally have** $**$:$1 - (u * u1 + v * u2) = u * v1 + v * v2$ **by** *auto*
case *False* **have** $0 \leq u * v1 + v * v2$ $0 \leq u * v1$ $0 \leq u * v1 + v * v2$ $0 \leq v * v2$ **apply** $-$
apply(*rule add-nonneg-nonneg*) **prefer** 4 **apply**(*rule add-nonneg-nonneg*) **apply**(*rule-tac* [!] *mult-nonneg-nonneg*)
using *as*(1,2) *obt1*(1,2) *obt2*(1,2) **by** *auto*
thus *?thesis* **unfolding** *obt1*(5) *obt2*(5) **unfolding** $*$ **and** $**$ **using** *False*
apply(*rule-tac* x=$((u * v1) / (u * v1 + v * v2)) *_R$ $b1 + ((v * v2) / (u * v1 + v * v2)) *_R$ $b2$ **in** *bexI*) **defer**
apply(*rule convex-convex-hull*[*of s*, *unfolded convex-def*, *rule-format*]) **using** *obt1*(4) *obt2*(4)
unfolding *add-divide-distrib*[*THEN sym*] **and** *real-0-le-divide-iff*
by (*auto simp add*: *scaleR-left-distrib scaleR-right-distrib*)
qed note $* = this$
have *u1*:$u1 \leq 1$ **unfolding** *obt1*(3)[*THEN sym*] **and** *not-le* **using** *obt1*(2) **by** *auto*
have *u2*:$u2 \leq 1$ **unfolding** *obt2*(3)[*THEN sym*] **and** *not-le* **using** *obt2*(2) **by** *auto*
have $u1 * u + u2 * v \leq (max\ u1\ u2) * u + (max\ u1\ u2) * v$ **apply**(*rule add-mono*)
apply(*rule-tac* [!] *mult-right-mono*) **using** *as*(1,2) *obt1*(1,2) *obt2*(1,2) **by** *auto*
also have $\ldots \leq 1$ **unfolding** *mult.add-right*[*THEN sym*] **and** *as*(3) **using** *u1 u2* **by** *auto*
finally
show $u *_R x + v *_R y \in$ *?hull* **unfolding** *mem-Collect-eq* **apply**(*rule-tac* x=$u * u1 + v * u2$ **in** *exI*)
apply(*rule conjI*) **defer apply**(*rule-tac* x=$1 - u * u1 - v * u2$ **in** *exI*) **unfolding** *Bex-def*
using *as*(1,2) *obt1*(1,2) *obt2*(1,2) $*$ **by**(*auto intro*!: *mult-nonneg-nonneg add-nonneg-nonneg simp add*: *algebra-simps*)
qed
qed

19.13 Explicit expression for convex hull.

lemma *convex-hull-indexed*:

fixes $s :: 'a$::*real-vector set*
shows *convex hull* $s = \{y.\ \exists k\ u\ x.\ (\forall i{\in}\{1{::}nat\ ..\ k\}.\ 0 \le u\ i \wedge x\ i \in s) \wedge$
$(setsum\ u\ \{1..k\} = 1) \wedge$
$(setsum\ (\lambda i.\ u\ i *_R x\ i)\ \{1..k\} = y)\}$ (**is** *?xyz = ?hull*)
apply(*rule hull-unique*) **unfolding** *mem-def*[*of - convex*] **apply**(*rule*) **defer**
apply(*subst convex-def*) **apply**(*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*rule*)
proof−
fix *x* **assume** *x*∈*s*
thus *x* ∈ *?hull* **unfolding** *mem-Collect-eq* **apply**(*rule-tac x=1* **in** *exI*, *rule-tac x=*λ*x. 1* **in** *exI*) **by** *auto*
next
fix *t* **assume** *as*:*s* ⊆ *t convex t*
show *?hull* ⊆ *t* **apply**(*rule*) **unfolding** *mem-Collect-eq* **apply**(*erule exE* | *erule conjE*)+ **proof**−
fix *x k u y* **assume** *assm*:∀ *i*∈{*1*::*nat..k*}. *0* ≤ *u i* ∧ *y i* ∈ *s setsum u* {*1..k*} = *1* (∑ *i* = *1..k*. *u i* $*_R$ *y i*) = *x*
show *x*∈*t* **unfolding** *assm*(*3*)[*THEN sym*] **apply**(*rule as*(*2*)[*unfolded convex*, *rule-format*])
using *assm*(*1*,*2*) *as*(*1*) **by** *auto* **qed**
next
fix *x y u v* **assume** *uv*:*0*≤*u 0*≤*v u*+*v*=(*1*::*real*) **and** *xy*:*x*∈*?hull y*∈*?hull*
from *xy* **obtain** *k1 u1 x1* **where** *x*:∀ *i*∈{*1*::*nat..k1*}. *0*≤*u1 i* ∧ *x1 i* ∈ *s setsum u1* {*Suc 0..k1*} = *1* (∑ *i* = *Suc 0..k1*. *u1 i* $*_R$ *x1 i*) = *x* **by** *auto*
from *xy* **obtain** *k2 u2 x2* **where** *y*:∀ *i*∈{*1*::*nat..k2*}. *0*≤*u2 i* ∧ *x2 i* ∈ *s setsum u2* {*Suc 0..k2*} = *1* (∑ *i* = *Suc 0..k2*. *u2 i* $*_R$ *x2 i*) = *y* **by** *auto*
have ∗:⋀*P* (*x1*::*'a*) *x2 s1 s2 i*.(*if P i then s1 else s2*) $*_R$ (*if P i then x1 else x2*) = (*if P i then s1* $*_R$ *x1 else s2* $*_R$ *x2*)
{*1..k1* + *k2*} ∩ {*1..k1*} = {*1..k1*} {*1..k1* + *k2*} ∩ − {*1..k1*} = (λ*i*. *i* + *k1*) ‘ {*1..k2*}
prefer *3* **apply**(*rule*,*rule*) **unfolding** *image-iff* **apply**(*rule-tac x=x* − *k1* **in** *bexI*) **by**(*auto simp add*: *not-le*)
have *inj*:*inj-on* (λ*i*. *i* + *k1*) {*1..k2*} **unfolding** *inj-on-def* **by** *auto*
show *u* $*_R$ *x* + *v* $*_R$ *y* ∈ *?hull* **apply**(*rule*)
apply(*rule-tac x=k1* + *k2* **in** *exI*, *rule-tac x=*λ*i. if i* ∈ {*1..k1*} *then u* ∗ *u1 i else v* ∗ *u2* (*i* − *k1*) **in** *exI*)
apply(*rule-tac x=*λ*i. if i* ∈ {*1..k1*} *then x1 i else x2* (*i* − *k1*) **in** *exI*) **apply**(*rule*,*rule*) **defer apply**(*rule*)
unfolding ∗ **and** *setsum-cases*[*OF finite-atLeastAtMost*[*of 1 k1* + *k2*]] **and** *setsum-reindex*[*OF inj*] **and** *o-def Collect-mem-eq*
unfolding *scaleR-scaleR*[*THEN sym*] *scaleR-right.setsum* [*symmetric*] *setsum-right-distrib*[*THEN sym*] **proof**−
fix *i* **assume** *i*:*i* ∈ {*1..k1*+*k2*}
show *0* ≤ (*if i* ∈ {*1..k1*} *then u* ∗ *u1 i else v* ∗ *u2* (*i* − *k1*)) ∧ (*if i* ∈ {*1..k1*} *then x1 i else x2* (*i* − *k1*)) ∈ *s*
proof(*cases i*∈{*1..k1*})
case *True* **thus** *?thesis* **using** *mult-nonneg-nonneg*[*of u u1 i*] **and** *uv*(*1*) *x*(*1*)[*THEN bspec*[**where** *x=i*]] **by** *auto*
next def *j* ≡ *i* − *k1*
case *False* **with** *i* **have** *j* ∈ {*1..k2*} **unfolding** *j-def* **by** *auto*

thus *?thesis* **unfolding** *j-def*[*symmetric*] **using** *False*
using *mult-nonneg-nonneg*[*of v u2 j*] **and** *uv*(*2*) *y*(*1*)[*THEN bspec*[**where** *x*=*j*]] **by** *auto* **qed**
qed(*auto simp add*: *not-le x*(*2*,*3*) *y*(*2*,*3*) *uv*(*3*))
qed

lemma *convex-hull-finite*:
fixes *s* :: *'a*::*real-vector set*
assumes *finite s*
shows *convex hull s* = {*y*. ∃ *u*. (∀ *x*∈*s*. *0* ≤ *u x*) ∧
setsum u s = *1* ∧ *setsum* (λ*x*. *u x* ∗$_R$ *x*) *s* = *y*} (**is** *?HULL* = *?set*)
proof(*rule hull-unique*, *auto simp add*: *mem-def*[*of - convex*] *convex-def*[*of ?set*])
fix *x* **assume** *x*∈*s* **thus** ∃ *u*. (∀ *x*∈*s*. *0* ≤ *u x*) ∧ *setsum u s* = *1* ∧ (∑ *x*∈*s*. *u x* ∗$_R$ *x*) = *x*
apply(*rule-tac x*=λ*y*. *if x*=*y then 1 else 0* **in** *exI*) **apply** *auto*
unfolding *setsum-delta'*[*OF assms*] **and** *setsum-delta''*[*OF assms*] **by** *auto*
next
fix *u v* ::*real* **assume** *uv*:*0* ≤ *u 0* ≤ *v u* + *v* = *1*
fix *ux* **assume** *ux*:∀ *x*∈*s*. *0* ≤ *ux x setsum ux s* = (*1*::*real*)
fix *uy* **assume** *uy*:∀ *x*∈*s*. *0* ≤ *uy x setsum uy s* = (*1*::*real*)
{ fix *x* **assume** *x*∈*s*
hence *0* ≤ *u* ∗ *ux x* + *v* ∗ *uy x* **using** *ux*(*1*)[*THEN bspec*[**where** *x*=*x*]] *uy*(*1*)[*THEN bspec*[**where** *x*=*x*]] **and** *uv*(*1*,*2*)
by (*auto*, *metis add-nonneg-nonneg mult-nonneg-nonneg uv*(*1*) *uv*(*2*)) **}**
moreover have (∑ *x*∈*s*. *u* ∗ *ux x* + *v* ∗ *uy x*) = *1*
unfolding *setsum-addf* **and** *setsum-right-distrib*[*THEN sym*] **and** *ux*(*2*) *uy*(*2*) **using** *uv*(*3*) **by** *auto*
moreover have (∑ *x*∈*s*. (*u* ∗ *ux x* + *v* ∗ *uy x*) ∗$_R$ *x*) = *u* ∗$_R$ (∑ *x*∈*s*. *ux x* ∗$_R$ *x*) + *v* ∗$_R$ (∑ *x*∈*s*. *uy x* ∗$_R$ *x*)
unfolding *scaleR-left-distrib* **and** *setsum-addf* **and** *scaleR-scaleR*[*THEN sym*] **and** *scaleR-right.setsum* [*symmetric*] **by** *auto*
ultimately show ∃ *uc*. (∀ *x*∈*s*. *0* ≤ *uc x*) ∧ *setsum uc s* = *1* ∧ (∑ *x*∈*s*. *uc x* ∗$_R$ *x*) = *u* ∗$_R$ (∑ *x*∈*s*. *ux x* ∗$_R$ *x*) + *v* ∗$_R$ (∑ *x*∈*s*. *uy x* ∗$_R$ *x*)
apply(*rule-tac x*=λ*x*. *u* ∗ *ux x* + *v* ∗ *uy x* **in** *exI*) **by** *auto*
next
fix *t* **assume** *t*:*s* ⊆ *t convex t*
fix *u* **assume** *u*:∀ *x*∈*s*. *0* ≤ *u x setsum u s* = (*1*::*real*)
thus (∑ *x*∈*s*. *u x* ∗$_R$ *x*) ∈ *t* **using** *t*(*2*)[*unfolded convex-explicit*, *THEN spec*[**where** *x*=*s*], *THEN spec*[**where** *x*=*u*]]
using *assms* **and** *t*(*1*) **by** *auto*
qed

19.14 Another formulation from Lars Schewe.

lemma *setsum-constant-scaleR*:
fixes *y* :: *'a*::*real-vector*
shows (∑ *x*∈*A*. *y*) = *of-nat* (*card A*) ∗$_R$ *y*
apply (*cases finite A*)
apply (*induct set*: *finite*)

apply (*simp-all add*: *algebra-simps*)
done

lemma *convex-hull-explicit*:
fixes $p :: 'a$::*real-vector set*
shows *convex hull* $p = \{y.\ \exists s\ u.$ *finite* $s \wedge s \subseteq p \wedge$
$(\forall x{\in}s.\ 0 \leq u\ x) \wedge$ *setsum* $u\ s = 1 \wedge$ *setsum* $(\lambda v.\ u\ v *_R v)\ s = y\}$ (**is** *?lhs* = *?rhs*)
proof−
{ fix x **assume** $x{\in}$*?lhs*
then obtain $k\ u\ y$ **where** *obt*:$\forall i{\in}\{1::nat..k\}.\ 0 \leq u\ i \wedge y\ i \in p$ *setsum* u $\{1..k\} = 1$ $(\sum i = 1..k.\ u\ i *_R y\ i) = x$
unfolding *convex-hull-indexed* **by** *auto*

have *fin*:*finite* $\{1..k\}$ **by** *auto*
have *fin'*:$\bigwedge v.$ *finite* $\{i \in \{1..k\}.\ y\ i = v\}$ **by** *auto*
{ fix j **assume** $j{\in}\{1..k\}$
hence $y\ j \in p\ 0 \leq$ *setsum* u $\{i.$ *Suc* $0 \leq i \wedge i \leq k \wedge y\ i = y\ j\}$
using *obt*(*1*)[*THEN bspec*[**where** $x{=}j$]] **and** *obt*(*2*) **apply** *simp*
apply(*rule setsum-nonneg*) **using** *obt*(*1*) **by** *auto* **}**
moreover
have $(\sum v{\in}y\ `\ \{1..k\}.$ *setsum* u $\{i \in \{1..k\}.\ y\ i = v\}) = 1$
unfolding *setsum-image-gen*[*OF fin*, *THEN sym*] **using** *obt*(*2*) **by** *auto*
moreover have $(\sum v{\in}y\ `\ \{1..k\}.$ *setsum* u $\{i \in \{1..k\}.\ y\ i = v\} *_R v) = x$
using *setsum-image-gen*[*OF fin*, *of* $\lambda i.\ u\ i *_R y\ i\ y$, *THEN sym*]
unfolding *scaleR-left.setsum* **using** *obt*(*3*) **by** *auto*
ultimately have $\exists s\ u.$ *finite* $s \wedge s \subseteq p \wedge (\forall x{\in}s.\ 0 \leq u\ x) \wedge$ *setsum* $u\ s = 1 \wedge (\sum v{\in}s.\ u\ v *_R v) = x$
apply(*rule-tac* $x{=}y\ `\ \{1..k\}$ **in** *exI*)
apply(*rule-tac* $x{=}\lambda v.$ *setsum* u $\{i{\in}\{1..k\}.\ y\ i = v\}$ **in** *exI*) **by** *auto*
hence *x*∈*?rhs* **by** *auto* **}**
moreover
{ fix y **assume** $y{\in}$*?rhs*
then obtain $s\ u$ **where** *obt*:*finite* $s\ s \subseteq p\ \forall x{\in}s.\ 0 \leq u\ x$ *setsum* $u\ s = 1$ $(\sum v{\in}s.\ u\ v *_R v) = y$ **by** *auto*

obtain f **where** f:*inj-on* f $\{1..card\ s\}$ $f\ `\ \{1..card\ s\} = s$ **using** *ex-bij-betw-nat-finite-1*[*OF obt*(*1*)] **unfolding** *bij-betw-def* **by** *auto*

{ fix i::*nat* **assume** $i{\in}\{1..card\ s\}$
hence $f\ i \in s$ **apply**(*subst* f(*2*)[*THEN sym*]) **by** *auto*
hence $0 \leq u\ (f\ i)\ f\ i \in p$ **using** *obt*(*2*,*3*) **by** *auto* **}**
moreover have $*$:*finite* $\{1..card\ s\}$ **by** *auto*
{ fix y **assume** $y{\in}s$
then obtain i **where** $i{\in}\{1..card\ s\}$ $f\ i = y$ **using** f **using** *image-iff*[*of y f* $\{1..card\ s\}$] **by** *auto*
hence $\{x.$ *Suc* $0 \leq x \wedge x \leq card\ s \wedge f\ x = y\} = \{i\}$ **apply** *auto* **using** *f*(*1*)[*unfolded inj-on-def*] **apply**(*erule-tac* $x{=}x$ **in** *ballE*) **by** *auto*
hence *card* $\{x.$ *Suc* $0 \leq x \wedge x \leq card\ s \wedge f\ x = y\} = 1$ **by** *auto*

hence $(\sum x{\in}\{x \in \{1..card\ s\}.\ f\ x = y\}.\ u\ (f\ x)) = u\ y$
$(\sum x{\in}\{x \in \{1..card\ s\}.\ f\ x = y\}.\ u\ (f\ x) *_R f\ x) = u\ y *_R y$
by (*auto simp add*: *setsum-constant-scaleR*) **}**

hence $(\sum x = 1..card\ s.\ u\ (f\ x)) = 1\ (\sum i = 1..card\ s.\ u\ (f\ i) *_R f\ i) = y$
unfolding *setsum-image-gen*[*OF* $*(1)$, *of* $\lambda x.\ u\ (f\ x) *_R f\ x\ f$] **and** *setsum-image-gen*[*OF* $*(1)$, *of* $\lambda x.\ u\ (f\ x)\ f$]
unfolding *f* **using** *setsum-cong2*[*of s* $\lambda y.\ (\sum x{\in}\{x \in \{1..card\ s\}.\ f\ x = y\}.\ u\ (f\ x) *_R f\ x)\ \lambda v.\ u\ v *_R v$]
using *setsum-cong2* [*of s* $\lambda y.\ (\sum x{\in}\{x \in \{1..card\ s\}.\ f\ x = y\}.\ u\ (f\ x))\ u$]
unfolding *obt*(*4*,*5*) **by** *auto*

ultimately have $\exists k\ u\ x.\ (\forall i{\in}\{1..k\}.\ 0 \le u\ i \wedge x\ i \in p) \wedge setsum\ u\ \{1..k\} = 1 \wedge (\sum i{::}nat = 1..k.\ u\ i *_R x\ i) = y$
apply(*rule-tac x=card s* **in** *exI*) **apply**(*rule-tac x=u* $\circ$ *f* **in** *exI*) **apply**(*rule-tac x=f* **in** *exI*) **by** *fastsimp*
hence *y* ∈ *?lhs* **unfolding** *convex-hull-indexed* **by** *auto* **}**
ultimately show *?thesis* **unfolding** *expand-set-eq* **by** *blast*
qed

19.15 A stepping theorem for that expansion.

lemma *convex-hull-finite-step*:
fixes *s* :: *'a*::*real-vector set* **assumes** *finite s*
shows $(\exists u.\ (\forall x{\in}insert\ a\ s.\ 0 \le u\ x) \wedge setsum\ u\ (insert\ a\ s) = w \wedge setsum\ (\lambda x.\ u\ x *_R x)\ (insert\ a\ s) = y)$
$\longleftrightarrow (\exists v{\ge}0.\ \exists u.\ (\forall x{\in}s.\ 0 \le u\ x) \wedge setsum\ u\ s = w - v \wedge setsum\ (\lambda x.\ u\ x *_R x)\ s = y - v *_R a)$ (**is** *?lhs = ?rhs*)
proof(*rule*, *case-tac*[!] *a*∈*s*)
assume *a*∈*s* **hence** ∗:*insert a s = s* **by** *auto*
assume *?lhs* **thus** *?rhs* **unfolding** ∗ **apply**(*rule-tac x=0* **in** *exI*) **by** *auto*
next
assume *?lhs* **then obtain** *u* **where** *u*:$\forall x{\in}insert\ a\ s.\ 0 \le u\ x\ setsum\ u\ (insert\ a\ s) = w\ (\sum x{\in}insert\ a\ s.\ u\ x *_R x) = y$ **by** *auto*
assume *a*∉*s* **thus** *?rhs* **apply**(*rule-tac x=u a* **in** *exI*) **using** *u*(*1*)[*THEN bspec*[**where** *x=a*]] **apply** *simp*
apply(*rule-tac x=u* **in** *exI*) **using** *u*[*unfolded setsum-clauses*(*2*)[*OF assms*]] **and** ‹*a*∉*s*› **by** *auto*
next
assume *a*∈*s* **hence** ∗:*insert a s = s* **by** *auto*
have *fin*:*finite* (*insert a s*) **using** *assms* **by** *auto*
assume *?rhs* **then obtain** *v u* **where** *uv*:$v{\ge}0\ \forall x{\in}s.\ 0 \le u\ x\ setsum\ u\ s = w - v\ (\sum x{\in}s.\ u\ x *_R x) = y - v *_R a$ **by** *auto*
show *?lhs* **apply**(*rule-tac x=*$\lambda x.\ (if\ a = x\ then\ v\ else\ 0) + u\ x$ **in** *exI*) **unfolding** *scaleR-left-distrib* **and** *setsum-addf* **and** *setsum-delta''*[*OF fin*] **and** *setsum-delta'*[*OF fin*]
unfolding *setsum-clauses*(*2*)[*OF assms*] **using** *uv* **and** *uv*(*2*)[*THEN bspec*[**where** *x=a*]] **and** ‹*a*∈*s*› **by** *auto*
next

assume *?rhs* **then obtain** *v u* **where** *uv:v≥0* $\forall x{\in}s.\ 0 \le u\ x$ *setsum u s* $= w - v$ $(\sum x{\in}s.\ u\ x *_R x) = y - v *_R a$ **by** *auto*
moreover assume $a{\notin}s$ **moreover have** $(\sum x{\in}s.$ *if a = x then v else u x*$) =$ *setsum u s* $(\sum x{\in}s.$ (*if a = x then v else u x*$) *_R x) = (\sum x{\in}s.\ u\ x *_R x)$
apply(*rule-tac setsum-cong2*) **defer apply**(*rule-tac setsum-cong2*) **using** ‹$a{\notin}s$› **by** *auto*
ultimately show *?lhs* **apply**(*rule-tac* $x{=}\lambda x.$ *if a = x then v else u x* **in** *exI*) **unfolding** *setsum-clauses*(*2*)[*OF assms*] **by** *auto*
qed

19.16 Hence some special cases.

lemma *convex-hull-2*:
convex hull $\{a,b\} = \{u *_R a + v *_R b \mid u\ v.\ 0 \le u \wedge 0 \le v \wedge u + v = 1\}$
proof− **have** $*{:}\bigwedge u.\ (\forall x{\in}\{a,\ b\}.\ 0 \le u\ x) \longleftrightarrow 0 \le u\ a \wedge 0 \le u\ b$ **by** *auto* **have** $**{:}$*finite* $\{b\}$ **by** *auto*
show *?thesis* **apply**(*simp add*: *convex-hull-finite*) **unfolding** *convex-hull-finite-step*[*OF* $**$, *of a 1*, *unfolded* $*$ *conj-assoc*]
apply *auto* **apply**(*rule-tac x=v* **in** *exI*) **apply**(*rule-tac x=1 − v* **in** *exI*) **apply** *simp*
apply(*rule-tac x=u* **in** *exI*) **apply** *simp* **apply**(*rule-tac* $x{=}\lambda x.$ *v* **in** *exI*) **by** *simp* **qed**

lemma *convex-hull-2-alt*: *convex hull* $\{a,b\} = \{a + u *_R (b - a) \mid u.\ \ 0 \le u \wedge u \le 1\}$
unfolding *convex-hull-2* **unfolding** *Collect-def*
proof(*rule ext*) **have** $*{:}\bigwedge x\ y$ ::*real*. $x + y = 1 \longleftrightarrow x = 1 - y$ **by** *auto*
fix *x* **show** $(\exists v\ u.\ x = v *_R a + u *_R b \wedge 0 \le v \wedge 0 \le u \wedge v + u = 1) = (\exists u.\ x = a + u *_R (b - a) \wedge 0 \le u \wedge u \le 1)$
unfolding $*$ **apply** *auto* **apply**(*rule-tac*[!] *x=u* **in** *exI*) **by** (*auto simp add*: *algebra-simps*) **qed**

lemma *convex-hull-3*:
convex hull $\{a,b,c\} = \{\ u *_R a + v *_R b + w *_R c \mid u\ v\ w.\ 0 \le u \wedge 0 \le v \wedge 0 \le w \wedge u + v + w = 1\}$
proof−
have *fin*:*finite* $\{a,b,c\}$ *finite* $\{b,c\}$ *finite* $\{c\}$ **by** *auto*
have $*{:}\bigwedge x\ y\ z$ ::*real*. $x + y + z = 1 \longleftrightarrow x = 1 - y - z$
$\bigwedge x\ y\ z$ ::*real*^-. $x + y + z = 1 \longleftrightarrow x = 1 - y - z$ **by** (*auto simp add*: *field-simps*)
show *?thesis* **unfolding** *convex-hull-finite*[*OF fin*(*1*)] **and** *Collect-def* **and** *convex-hull-finite-step*[*OF fin*(*2*)] **and** $*$
unfolding *convex-hull-finite-step*[*OF fin*(*3*)] **apply**(*rule ext*) **apply** *simp* **apply** *auto*
apply(*rule-tac x=va* **in** *exI*) **apply** (*rule-tac x=u c* **in** *exI*) **apply** *simp*
apply(*rule-tac x=1 − v − w* **in** *exI*) **apply** *simp* **apply**(*rule-tac x=v* **in** *exI*) **apply** *simp* **apply**(*rule-tac* $x{=}\lambda x.$ *w* **in** *exI*) **by** *simp* **qed**

lemma *convex-hull-3-alt*:

convex hull {*a*,*b*,*c*} = {*a* + *u* ∗*R* (*b* − *a*) + *v* ∗*R* (*c* − *a*) | *u v*. *0* ≤ *u* ∧ *0* ≤ *v* ∧ *u* + *v* ≤ *1*}
proof− **have** ∗:⋀*x y z* ::*real*. *x* + *y* + *z* = *1* ⟷ *x* = *1* − *y* − *z* **by** *auto*
show *?thesis* **unfolding** *convex-hull-3* **apply** (*auto simp add*: ∗) **apply**(*rule-tac x*=*v* **in** *exI*) **apply**(*rule-tac x*=*w* **in** *exI*) **apply** (*simp add*: *algebra-simps*)
apply(*rule-tac x*=*u* **in** *exI*) **apply**(*rule-tac x*=*v* **in** *exI*) **by** (*simp add*: *algebra-simps*)
qed

19.17 Relations among closure notions and corresponding hulls.

TODO: Generalize linear algebra concepts defined in *Euclidean-Space.thy* so that we can generalize these lemmas.

lemma *subspace-imp-affine*:
fixes *s* :: (*real* ^ -) *set* **shows** *subspace s* ⟹ *affine s*
unfolding *subspace-def affine-def smult-conv-scaleR* **by** *auto*

lemma *affine-imp-convex*: *affine s* ⟹ *convex s*
unfolding *affine-def convex-def* **by** *auto*

lemma *subspace-imp-convex*:
fixes *s* :: (*real* ^ -) *set* **shows** *subspace s* ⟹ *convex s*
using *subspace-imp-affine affine-imp-convex* **by** *auto*

lemma *affine-hull-subset-span*:
fixes *s* :: (*real* ^ -) *set* **shows** (*affine hull s*) ⊆ (*span s*)
by (*metis hull-minimal mem-def span-inc subspace-imp-affine subspace-span*)

lemma *convex-hull-subset-span*:
fixes *s* :: (*real* ^ -) *set* **shows** (*convex hull s*) ⊆ (*span s*)
by (*metis hull-minimal mem-def span-inc subspace-imp-convex subspace-span*)

lemma *convex-hull-subset-affine-hull*: (*convex hull s*) ⊆ (*affine hull s*)
by (*metis affine-affine-hull affine-imp-convex hull-minimal hull-subset mem-def*)

lemma *affine-dependent-imp-dependent*:
fixes *s* :: (*real* ^ -) *set* **shows** *affine-dependent s* ⟹ *dependent s*
unfolding *affine-dependent-def dependent-def*
using *affine-hull-subset-span* **by** *auto*

lemma *dependent-imp-affine-dependent*:
fixes *s* :: (*real* ^ -) *set*
assumes *dependent* {*x* − *a*| *x* . *x* ∈ *s*} *a* ∉ *s*
shows *affine-dependent* (*insert a s*)
proof−
from *assms*(*1*)[*unfolded dependent-explicit smult-conv-scaleR*] **obtain** *S u v*
where *obt*:*finite S S* ⊆ {*x* − *a* |*x*. *x* ∈ *s*} *v*∈*S u v* ≠ *0* (∑ *v*∈*S*. *u v* ∗*R* *v*) =

0 **by** *auto*
def $t \equiv (\lambda x.\ x + a)\ `\ S$

have *inj*:*inj-on* $(\lambda x.\ x + a)\ S$ **unfolding** *inj-on-def* **by** *auto*
have $0 \notin S$ **using** *obt*(*2*) *assms*(*2*) **unfolding** *subset-eq* **by** *auto*
have *fin*:*finite t* **and** $t \subseteq s$ **unfolding** *t-def* **using** *obt*(*1*,*2*) **by** *auto*

hence *finite* (*insert a t*) **and** *insert a t* $\subseteq$ *insert a s* **by** *auto*
moreover have $*$:$\bigwedge P\ Q.\ (\sum x \in t.\ (\textit{if } x = a \textit{ then } P\ x \textit{ else } Q\ x)) = (\sum x \in t.\ Q\ x)$
apply(*rule setsum-cong2*) **using** ‹$a \notin s$› ‹$t \subseteq s$› **by** *auto*
have $(\sum x \in \textit{insert } a\ t.\ \textit{if } x = a \textit{ then } -(\sum x \in t.\ u\ (x - a)) \textit{ else } u\ (x - a)) = 0$
unfolding *setsum-clauses*(*2*)[*OF fin*] **using** ‹$a \notin s$› ‹$t \subseteq s$› **apply** *auto* **unfolding** $*$ **by** *auto*
moreover have $\exists v \in \textit{insert } a\ t.\ (\textit{if } v = a \textit{ then } -(\sum x \in t.\ u\ (x - a)) \textit{ else } u\ (v - a)) \neq 0$
apply(*rule-tac x*=$v + a$ **in** *bexI*) **using** *obt*(*3*,*4*) **and** ‹$0 \notin S$› **unfolding** *t-def* **by** *auto*
moreover have $*$:$\bigwedge P\ Q.\ (\sum x \in t.\ (\textit{if } x = a \textit{ then } P\ x \textit{ else } Q\ x) *_R x) = (\sum x \in t.\ Q\ x *_R x)$
apply(*rule setsum-cong2*) **using** ‹$a \notin s$› ‹$t \subseteq s$› **by** *auto*
have $(\sum x \in t.\ u\ (x - a)) *_R a = (\sum v \in t.\ u\ (v - a) *_R v)$
unfolding *scaleR-left.setsum* **unfolding** *t-def* **and** *setsum-reindex*[*OF inj*] **and** *o-def*
using *obt*(*5*) **by** (*auto simp add*: *setsum-addf scaleR-right-distrib*)
hence $(\sum v \in \textit{insert } a\ t.\ (\textit{if } v = a \textit{ then } -(\sum x \in t.\ u\ (x - a)) \textit{ else } u\ (v - a)) *_R v) = 0$
unfolding *setsum-clauses*(*2*)[*OF fin*] **using** ‹$a \notin s$› ‹$t \subseteq s$› **by** (*auto simp add*: $*$ *vector-smult-lneg*)
ultimately show *?thesis* **unfolding** *affine-dependent-explicit*
apply(*rule-tac x*=*insert a t* **in** *exI*) **by** *auto*
qed

lemma *convex-cone*:
$\textit{convex } s \wedge \textit{cone } s \longleftrightarrow (\forall x \in s.\ \forall y \in s.\ (x + y) \in s) \wedge (\forall x \in s.\ \forall c \geq 0.\ (c *_R x) \in s)$ (**is** *?lhs* = *?rhs*)
proof−
{ fix *x y* **assume** $x \in s$ $y \in s$ **and** *?lhs*
hence $2 *_R x \in s$ $2 *_R y \in s$ **unfolding** *cone-def* **by** *auto*
hence $x + y \in s$ **using** ‹*?lhs*›[*unfolded convex-def*, *THEN conjunct1*]
apply(*erule-tac x*=$2 *_R x$ **in** *ballE*) **apply**(*erule-tac x*=$2 *_R y$ **in** *ballE*)
apply(*erule-tac x*=*1/2* **in** *allE*) **apply** *simp* **apply**(*erule-tac x*=*1/2* **in** *allE*)
by *auto* **}**
thus *?thesis* **unfolding** *convex-def cone-def* **by** *blast*
qed

lemma *affine-dependent-biggerset*: **fixes** *s*::(*real^'n*) *set*
assumes *finite s card s* $\geq$ *CARD*(*'n*) + *2*
shows *affine-dependent s*

proof−
have $s\neq\{\}$ **using** *assms* **by** *auto* **then obtain** a **where** $a\in s$ **by** *auto*
have $*$:$\{x - a \mid x.\ x \in s - \{a\}\} = (\lambda x.\ x - a)$ ` $(s - \{a\})$ **by** *auto*
have *card* $\{x - a \mid x.\ x \in s - \{a\}\} =$ *card* $(s - \{a\})$ **unfolding** $*$
apply(*rule card-image*) **unfolding** *inj-on-def* **by** *auto*
also have $\ldots > CARD('n)$ **using** *assms*(*2*)
unfolding *card-Diff-singleton*[*OF assms*(*1*) ‹$a\in s$›] **by** *auto*
finally show *?thesis* **apply**(*subst insert-Diff*[*OF* ‹$a\in s$›, *THEN sym*])
apply(*rule dependent-imp-affine-dependent*)
apply(*rule dependent-biggerset*) **by** *auto* **qed**

lemma *affine-dependent-biggerset-general*:
assumes *finite* $(s::(real\hat{}'n)\ set)$ *card* $s \geq$ *dim* $s + 2$
shows *affine-dependent* s
proof−
from *assms*(*2*) **have** $s \neq \{\}$ **by** *auto*
then obtain a **where** $a\in s$ **by** *auto*
have $*$:$\{x - a \mid x.\ x \in s - \{a\}\} = (\lambda x.\ x - a)$ ` $(s - \{a\})$ **by** *auto*
have $**$:*card* $\{x - a \mid x.\ x \in s - \{a\}\} =$ *card* $(s - \{a\})$ **unfolding** $*$
apply(*rule card-image*) **unfolding** *inj-on-def* **by** *auto*
have *dim* $\{x - a \mid x.\ x \in s - \{a\}\} \leq$ *dim* s
apply(*rule subset-le-dim*) **unfolding** *subset-eq*
using ‹$a\in s$› **by** (*auto simp add*:*span-superset span-sub*)
also have $\ldots <$ *dim* $s + 1$ **by** *auto*
also have $\ldots \leq$ *card* $(s - \{a\})$ **using** *assms*
using *card-Diff-singleton*[*OF assms*(*1*) ‹$a\in s$›] **by** *auto*
finally show *?thesis* **apply**(*subst insert-Diff*[*OF* ‹$a\in s$›, *THEN sym*])
apply(*rule dependent-imp-affine-dependent*) **apply**(*rule dependent-biggerset-general*)
unfolding $**$ **by** *auto* **qed**

19.18 Caratheodory's theorem.

lemma *convex-hull-caratheodory*: **fixes** $p::(real\hat{}'n)\ set$
shows *convex hull* $p = \{y.\ \exists s\ u.$ *finite* $s \wedge s \subseteq p \wedge$ *card* $s \leq CARD('n) + 1 \wedge$
$(\forall x\in s.\ 0 \leq u\ x) \wedge$ *setsum* $u\ s = 1 \wedge$ *setsum* $(\lambda v.\ u\ v *_R v)\ s = y\}$
unfolding *convex-hull-explicit expand-set-eq mem-Collect-eq*
proof(*rule*,*rule*)
fix y **let** *?P* $= \lambda n.\ \exists s\ u.$ *finite* $s \wedge$ *card* $s = n \wedge s \subseteq p \wedge (\forall x\in s.\ 0 \leq u\ x) \wedge$
setsum $u\ s = 1 \wedge (\sum v\in s.\ u\ v *_R v) = y$
assume $\exists s\ u.$ *finite* $s \wedge s \subseteq p \wedge (\forall x\in s.\ 0 \leq u\ x) \wedge$ *setsum* $u\ s = 1 \wedge (\sum v\in s.$
$u\ v *_R v) = y$
then obtain N **where** *?P N* **by** *auto*
hence $\exists n\leq N.\ (\forall k<n.\ \neg$ *?P k*$) \wedge$ *?P n* **apply**(*rule-tac ex-least-nat-le*) **by** *auto*
then obtain n **where** *?P n* **and** *smallest*:$\forall k<n.\ \neg$ *?P k* **by** *blast*
then obtain $s\ u$ **where** *obt*:*finite s card s* $= n$ $s\subseteq p$ $\forall x\in s.\ 0 \leq u\ x$ *setsum u s*
$= 1$ $(\sum v\in s.\ u\ v *_R v) = y$ **by** *auto*

have *card* $s \leq CARD('n) + 1$ **proof**(*rule ccontr*, *simp only*: *not-le*)
assume $CARD('n) + 1 <$ *card* s

hence *affine-dependent s* **using** *affine-dependent-biggerset*[*OF obt*(*1*)] **by** *auto*
then obtain *w v* **where** *wv*:*setsum w s* = *0 v*∈*s w v* ≠ *0* (∑ *v*∈*s*. *w v* ∗$_R$ *v*) = *0*
using *affine-dependent-explicit-finite*[*OF obt*(*1*)] **by** *auto*
def *i* ≡ (λ*v*. (*u v*) / (− *w v*)) ‘ {*v*∈*s*. *w v* < *0*} **def** *t* ≡ *Min i*
have ∃ *x*∈*s*. *w x* < *0* **proof**(*rule ccontr*, *simp add*: *not-less*)
assume *as*:∀ *x*∈*s*. *0* ≤ *w x*
hence *setsum w* (*s* − {*v*}) ≥ *0* **apply**(*rule-tac setsum-nonneg*) **by** *auto*
hence *setsum w s* > *0* **unfolding** *setsum-diff1'*[*OF obt*(*1*) ‹*v*∈*s*›]
using *as*[*THEN bspec*[**where** *x*=*v*]] **and** ‹*v*∈*s*› **using** ‹*w v* ≠ *0*› **by** *auto*
thus *False* **using** *wv*(*1*) **by** *auto*
qed hence *i*≠{} **unfolding** *i-def* **by** *auto*

hence *t* ≥ *0* **using** *Min-ge-iff*[*of i 0*] **and** *obt*(*1*) **unfolding** *t-def i-def*
using *obt*(*4*)[*unfolded le-less*] **apply** *auto* **unfolding** *divide-le-0-iff* **by** *auto*
have *t*:∀ *v*∈*s*. *u v* + *t* ∗ *w v* ≥ *0* **proof**
fix *v* **assume** *v*∈*s* **hence** *v*:*0*≤*u v* **using** *obt*(*4*)[*THEN bspec*[**where** *x*=*v*]] **by** *auto*
show*0* ≤ *u v* + *t* ∗ *w v* **proof**(*cases w v* < *0*)
case *False* **thus** *?thesis* **apply**(*rule-tac add-nonneg-nonneg*)
using *v* **apply** *simp* **apply**(*rule mult-nonneg-nonneg*) **using** ‹*t*≥*0*› **by** *auto* **next**
case *True* **hence** *t* ≤ *u v* / (− *w v*) **using** ‹*v*∈*s*›
unfolding *t-def i-def* **apply**(*rule-tac Min-le*) **using** *obt*(*1*) **by** *auto*
thus *?thesis* **unfolding** *real-0-le-add-iff*
using *pos-le-divide-eq*[*OF True*[*unfolded neg-0-less-iff-less*[*THEN sym*]]] **by** *auto*
qed qed

obtain *a* **where** *a*∈*s* **and** *t* = (λ*v*. (*u v*) / (− *w v*)) *a* **and** *w a* < *0*
using *Min-in*[*OF* - ‹*i*≠{}›] **and** *obt*(*1*) **unfolding** *i-def t-def* **by** *auto*
hence *a*:*a*∈*s u a* + *t* ∗ *w a* = *0* **by** *auto*
have ∗:⋀*f*. *setsum f* (*s* − {*a*}) = *setsum f s* − ((*f a*)::'*a*::*ring*) **unfolding** *setsum-diff1'*[*OF obt*(*1*) ‹*a*∈*s*›] **by** *auto*
have (∑ *v*∈*s*. *u v* + *t* ∗ *w v*) = *1*
unfolding *setsum-addf wv*(*1*) *setsum-right-distrib*[*THEN sym*] *obt*(*5*) **by** *auto*
moreover have (∑ *v*∈*s*. *u v* ∗$_R$ *v* + (*t* ∗ *w v*) ∗$_R$ *v*) − (*u a* ∗$_R$ *a* + (*t* ∗ *w a*) ∗$_R$ *a*) = *y*
unfolding *setsum-addf obt*(*6*) *scaleR-scaleR*[*THEN sym*] *scaleR-right.setsum* [*symmetric*] *wv*(*4*)
using *a*(*2*) [*THEN eq-neg-iff-add-eq-0* [*THEN iffD2*]]
by (*simp add*: *vector-smult-lneg*)
ultimately have *?P* (*n* − *1*) **apply**(*rule-tac x*=(*s* − {*a*}) **in** *exI*)
apply(*rule-tac x*=λ*v*. *u v* + *t* ∗ *w v* **in** *exI*) **using** *obt*(*1*−*3*) **and** *t* **and** *a* **by** (*auto simp add*: ∗ *scaleR-left-distrib*)
thus *False* **using** *smallest*[*THEN spec*[**where** *x*=*n* − *1*]] **by** *auto* **qed**
thus ∃ *s u*. *finite s* ∧ *s* ⊆ *p* ∧ *card s* ≤ *CARD*('*n*) + *1*
∧ (∀ *x*∈*s*. *0* ≤ *u x*) ∧ *setsum u s* = *1* ∧ (∑ *v*∈*s*. *u v* ∗$_R$ *v*) = *y* **using** *obt* **by** *auto*

qed *auto*

lemma *caratheodory*:
convex hull p = {*x*::*real*^'*n*. ∃ *s*. *finite s* ∧ *s* ⊆ *p* ∧
card s ≤ *CARD*('*n*) + *1* ∧ *x* ∈ *convex hull s*}
unfolding *expand-set-eq* **apply**(*rule*, *rule*) **unfolding** *mem-Collect-eq* **proof**−
fix *x* **assume** *x* ∈ *convex hull p*
then obtain *s u* **where** *finite s s* ⊆ *p card s* ≤ *CARD*('*n*) + *1*
∀ *x*∈*s*. *0* ≤ *u x setsum u s* = *1* (∑ *v*∈*s*. *u v* ∗*R v*) = *x***unfolding** *convex-hull-caratheodory*
by *auto*
thus ∃ *s*. *finite s* ∧ *s* ⊆ *p* ∧ *card s* ≤ *CARD*('*n*) + *1* ∧ *x* ∈ *convex hull s*
apply(*rule-tac x*=*s* **in** *exI*) **using** *hull-subset*[*of s convex*]
using *convex-convex-hull*[*unfolded convex-explicit*, *of s*, *THEN spec*[**where** *x*=*s*],
THEN spec[**where** *x*=*u*]] **by** *auto*
next
fix *x* **assume** ∃ *s*. *finite s* ∧ *s* ⊆ *p* ∧ *card s* ≤ *CARD*('*n*) + *1* ∧ *x* ∈ *convex hull*
s
then obtain *s* **where** *finite s s* ⊆ *p card s* ≤ *CARD*('*n*) + *1 x* ∈ *convex hull s*
by *auto*
thus *x* ∈ *convex hull p* **using** *hull-mono*[*OF* ‹*s*⊆*p*›] **by** *auto*
qed

19.19 Openness and compactness are preserved by convex hull operation.

lemma *open-convex-hull*[*intro*]:
fixes *s* :: '*a*::*real-normed-vector set*
assumes *open s*
shows *open*(*convex hull s*)
unfolding *open-contains-cball convex-hull-explicit* **unfolding** *mem-Collect-eq*
ball-simps(*10*)
proof(*rule*, *rule*) **fix** *a*
assume ∃ *sa u*. *finite sa* ∧ *sa* ⊆ *s* ∧ (∀ *x*∈*sa*. *0* ≤ *u x*) ∧ *setsum u sa* = *1* ∧
(∑ *v*∈*sa*. *u v* ∗*R v*) = *a*
then obtain *t u* **where** *obt*:*finite t t*⊆*s* ∀ *x*∈*t*. *0* ≤ *u x setsum u t* = *1* (∑ *v*∈*t*.
u v ∗*R v*) = *a* **by** *auto*

from *assms*[*unfolded open-contains-cball*] **obtain** *b* **where** *b*:∀ *x*∈*s*. *0* < *b x* ∧
cball x (*b x*) ⊆ *s*
using *bchoice*[*of s* λ*x e*. *e*>*0* ∧ *cball x e* ⊆ *s*] **by** *auto*
have *b* ‘ *t*≠{} **unfolding** *i-def* **using** *obt* **by** *auto* **def** *i* ≡ *b* ‘ *t*

show ∃ *e*>*0*. *cball a e* ⊆ {*y*. ∃ *sa u*. *finite sa* ∧ *sa* ⊆ *s* ∧ (∀ *x*∈*sa*. *0* ≤ *u x*) ∧
setsum u sa = *1* ∧ (∑ *v*∈*sa*. *u v* ∗*R v*) = *y*}
apply(*rule-tac x*=*Min i* **in** *exI*) **unfolding** *subset-eq* **apply** *rule* **defer apply**
rule **unfolding** *mem-Collect-eq*
proof−
show *0* < *Min i* **unfolding** *i-def* **and** *Min-gr-iff*[*OF finite-imageI*[*OF obt*(*1*)]
‹*b* ‘ *t*≠{}›]

using *b* **apply** *simp* **apply** *rule* **apply**(*erule-tac x=x* **in** *ballE*) **using** ‹*t*⊆*s*› **by** *auto*
next **fix** *y* **assume** *y* ∈ *cball a* (*Min i*)
hence *y*:*norm* (*a* − *y*) ≤ *Min i* **unfolding** *dist-norm*[*THEN sym*] **by** *auto*
{ fix *x* **assume** *x*∈*t*
hence *Min i* ≤ *b x* **unfolding** *i-def* **apply**(*rule-tac Min-le*) **using** *obt*(*1*) **by** *auto*
hence *x* + (*y* − *a*) ∈ *cball x* (*b x*) **using** *y* **unfolding** *mem-cball dist-norm* **by** *auto*
moreover from ‹*x*∈*t*› **have** *x*∈*s* **using** *obt*(*2*) **by** *auto*
ultimately have *x* + (*y* − *a*) ∈ *s* **using** *y* **and** *b*[*THEN bspec*[**where** *x*=*x*]] **unfolding** *subset-eq* **by** *fast* **}**
moreover
have ∗:*inj-on* (λ*v*. *v* + (*y* − *a*)) *t* **unfolding** *inj-on-def* **by** *auto*
have (∑ *v*∈(λ*v*. *v* + (*y* − *a*)) ‘ *t*. *u* (*v* − (*y* − *a*))) = *1*
unfolding *setsum-reindex*[*OF* ∗] *o-def* **using** *obt*(*4*) **by** *auto*
moreover have (∑ *v*∈(λ*v*. *v* + (*y* − *a*)) ‘ *t*. *u* (*v* − (*y* − *a*)) ∗*R* *v*) = *y*
unfolding *setsum-reindex*[*OF* ∗] *o-def* **using** *obt*(*4*,*5*)
by (*simp add*: *setsum-addf setsum-subtractf scaleR-left.setsum*[*THEN sym*] *scaleR-right-distrib*)
ultimately show ∃ *sa u*. *finite sa* ∧ (∀ *x*∈*sa*. *x* ∈ *s*) ∧ (∀ *x*∈*sa*. *0* ≤ *u x*) ∧ *setsum u sa* = *1* ∧ (∑ *v*∈*sa*. *u v* ∗*R* *v*) = *y*
apply(*rule-tac x*=(λ*v*. *v* + (*y* − *a*)) ‘ *t* **in** *exI*) **apply**(*rule-tac x*=λ*v*. *u* (*v* − (*y* − *a*)) **in** *exI*)
using *obt*(*1*, *3*) **by** *auto*
qed
qed

lemma *compact-real-interval*:
fixes *a b* :: *real* **shows** *compact* {*a*..*b*}
proof (*rule bounded-closed-imp-compact*)
have ∀ *y*∈{*a*..*b*}. *dist a y* ≤ *dist a b*
unfolding *dist-real-def* **by** *auto*
thus *bounded* {*a*..*b*} **unfolding** *bounded-def* **by** *fast*
show *closed* {*a*..*b*} **by** (*rule closed-real-atLeastAtMost*)
qed

lemma *compact-convex-combinations*:
fixes *s t* :: *'a*::*real-normed-vector set*
assumes *compact s compact t*
shows *compact* { (*1* − *u*) ∗*R* *x* + *u* ∗*R* *y* | *x y u*. *0* ≤ *u* ∧ *u* ≤ *1* ∧ *x* ∈ *s* ∧ *y* ∈ *t*}
proof−
let *?X* = {*0*..*1*} × *s* × *t*
let *?h* = (λ*z*. (*1* − *fst z*) ∗*R* *fst* (*snd z*) + *fst z* ∗*R* *snd* (*snd z*))
have ∗:{ (*1* − *u*) ∗*R* *x* + *u* ∗*R* *y* | *x y u*. *0* ≤ *u* ∧ *u* ≤ *1* ∧ *x* ∈ *s* ∧ *y* ∈ *t*} = *?h* ‘ *?X*
apply(*rule set-ext*) **unfolding** *image-iff mem-Collect-eq*

```
    apply rule apply auto
    apply (rule-tac x=u in rev-bexI, simp)
    apply (erule rev-bexI, erule rev-bexI, simp)
    by auto
  have continuous-on ({0..1} × s × t)
     (λz. (1 − fst z) *R fst (snd z) + fst z *R snd (snd z))
    unfolding continuous-on by (rule ballI) (intro tendsto-intros)
  thus ?thesis unfolding *
    apply (rule compact-continuous-image)
    apply (intro compact-Times compact-real-interval assms)
    done
qed

lemma compact-convex-hull: fixes s::(real^'n) set
  assumes compact s  shows compact(convex hull s)
proof(cases s={})
  case True thus ?thesis using compact-empty by simp
next
  case False then obtain w where w∈s by auto
  show ?thesis unfolding caratheodory[of s]
  proof(induct (CARD('n) + 1))
    have *:{x.∃ sa. finite sa ∧ sa ⊆ s ∧ card sa ≤ 0 ∧ x ∈ convex hull sa} = {}
      using compact-empty by auto
    case 0 thus ?case unfolding * by simp
  next
    case (Suc n)
    show ?case proof(cases n=0)
      case True have {x. ∃ t. finite t ∧ t ⊆ s ∧ card t ≤ Suc n ∧ x ∈ convex hull
t} = s
        unfolding expand-set-eq and mem-Collect-eq proof(rule, rule)
        fix x assume ∃ t. finite t ∧ t ⊆ s ∧ card t ≤ Suc n ∧ x ∈ convex hull t
        then obtain t where t:finite t t ⊆ s card t ≤ Suc n x ∈ convex hull t by
auto
        show x∈s proof(cases card t = 0)
         case True thus ?thesis using t(4) unfolding card-0-eq[OF t(1)] by simp
        next
          case False hence card t = Suc 0 using t(3) ‹n=0› by auto
          then obtain a where t = {a} unfolding card-Suc-eq by auto
          thus ?thesis using t(2,4) by simp
        qed
      next
        fix x assume x∈s
        thus ∃ t. finite t ∧ t ⊆ s ∧ card t ≤ Suc n ∧ x ∈ convex hull t
          apply(rule-tac x={x} in exI) unfolding convex-hull-singleton by auto
      qed thus ?thesis using assms by simp
    next
      case False have {x. ∃ t. finite t ∧ t ⊆ s ∧ card t ≤ Suc n ∧ x ∈ convex hull
t} =
        { (1 − u) *R x + u *R y | x y u.
```

0 ≤ u ∧ u ≤ 1 ∧ x ∈ s ∧ y ∈ {x. ∃ t. finite t ∧ t ⊆ s ∧ card t ≤ n ∧ x ∈ convex hull t}}
unfolding *expand-set-eq* **and** *mem-Collect-eq* **proof**(*rule*,*rule*)
fix *x* **assume** *∃ u v c. x = (1 − c) ∗R u + c ∗R v ∧*
0 ≤ c ∧ c ≤ 1 ∧ u ∈ s ∧ (∃ t. finite t ∧ t ⊆ s ∧ card t ≤ n ∧ v ∈ convex hull t)
then obtain *u v c t* **where** *obt*:*x = (1 − c) ∗R u + c ∗R v*
0 ≤ c ∧ c ≤ 1 u ∈ s finite t t ⊆ s card t ≤ n v ∈ convex hull t **by** *auto*
moreover have *(1 − c) ∗R u + c ∗R v ∈ convex hull insert u t*
apply(*rule mem-convex*) **using** *obt*(*2*) **and** *convex-convex-hull* **and** *hull-subset*[*of insert u t convex*]
using *obt*(*7*) **and** *hull-mono*[*of t insert u t*] **by** *auto*
ultimately show *∃ t. finite t ∧ t ⊆ s ∧ card t ≤ Suc n ∧ x ∈ convex hull t*
apply(*rule-tac x=insert u t* **in** *exI*) **by** (*auto simp add*: *card-insert-if*)
next
fix *x* **assume** *∃ t. finite t ∧ t ⊆ s ∧ card t ≤ Suc n ∧ x ∈ convex hull t*
then obtain *t* **where** *t*:*finite t t ⊆ s card t ≤ Suc n x ∈ convex hull t* **by** *auto*
let *?P = ∃ u v c. x = (1 − c) ∗R u + c ∗R v ∧*
0 ≤ c ∧ c ≤ 1 ∧ u ∈ s ∧ (∃ t. finite t ∧ t ⊆ s ∧ card t ≤ n ∧ v ∈ convex hull t)
show *?P* **proof**(*cases card t = Suc n*)
case *False* **hence** *card t ≤ n* **using** *t*(*3*) **by** *auto*
thus *?P* **apply**(*rule-tac x=w* **in** *exI*, *rule-tac x=x* **in** *exI*, *rule-tac x=1* **in** *exI*) **using** ‹*w∈s*› **and** *t*
by(*auto intro*!: *exI*[**where** *x=t*])
next
case *True* **then obtain** *a u* **where** *au*:*t = insert a u a∉u* **apply**(*drule-tac card-eq-SucD*) **by** *auto*
show *?P* **proof**(*cases u={}*)
case *True* **hence** *x=a* **using** *t*(*4*)[*unfolded au*] **by** *auto*
show *?P* **unfolding** ‹*x=a*› **apply**(*rule-tac x=a* **in** *exI*, *rule-tac x=a* **in** *exI*, *rule-tac x=1* **in** *exI*)
using *t* **and** ‹*n≠0*› **unfolding** *au* **by**(*auto intro*!: *exI*[**where** *x={a}*])
next
case *False* **obtain** *ux vx b* **where** *obt*:*ux≥0 vx≥0 ux + vx = 1 b ∈ convex hull u x = ux ∗R a + vx ∗R b*
using *t*(*4*)[*unfolded au convex-hull-insert*[*OF False*]] **by** *auto*
have *∗:1 − vx = ux* **using** *obt*(*3*) **by** *auto*
show *?P* **apply**(*rule-tac x=a* **in** *exI*, *rule-tac x=b* **in** *exI*, *rule-tac x=vx* **in** *exI*)
using *obt* **and** *t*(*1−3*) **unfolding** *au* **and** *∗* **using** *card-insert-disjoint*[*OF - au*(*2*)]
by(*auto intro*!: *exI*[**where** *x=u*])
qed
qed
qed
thus *?thesis* **using** *compact-convex-combinations*[*OF assms Suc*] **by** *simp*
qed

qed
qed

lemma *finite-imp-compact-convex-hull*:
fixes *s* :: (*real* ^ -) *set*
shows *finite s* ⟹ *compact*(*convex hull s*)
by (*metis compact-convex-hull finite-imp-compact*)

19.20 Extremal points of a simplex are some vertices.

lemma *dist-increases-online*:
fixes *a b d* :: *'a*::*real-inner*
assumes *d* ≠ *0*
shows *dist a* (*b* + *d*) > *dist a b* ∨ *dist a* (*b* − *d*) > *dist a b*
proof(*cases inner a d* − *inner b d* > *0*)
case *True* **hence** *0* < *inner d d* + (*inner a d* ∗ *2* − *inner b d* ∗ *2*)
apply(*rule-tac add-pos-pos*) **using** *assms* **by** *auto*
thus *?thesis* **apply**(*rule-tac disjI2*) **unfolding** *dist-norm* **and** *norm-eq-sqrt-inner* **and** *real-sqrt-less-iff*
by (*simp add*: *algebra-simps inner-commute*)
next
case *False* **hence** *0* < *inner d d* + (*inner b d* ∗ *2* − *inner a d* ∗ *2*)
apply(*rule-tac add-pos-nonneg*) **using** *assms* **by** *auto*
thus *?thesis* **apply**(*rule-tac disjI1*) **unfolding** *dist-norm* **and** *norm-eq-sqrt-inner* **and** *real-sqrt-less-iff*
by (*simp add*: *algebra-simps inner-commute*)
qed

lemma *norm-increases-online*:
fixes *d* :: *'a*::*real-inner*
shows *d* ≠ *0* ⟹ *norm*(*a* + *d*) > *norm a* ∨ *norm*(*a* − *d*) > *norm a*
using *dist-increases-online*[*of d a 0*] **unfolding** *dist-norm* **by** *auto*

lemma *simplex-furthest-lt*:
fixes *s*::*'a*::*real-inner set* **assumes** *finite s*
shows ∀ *x* ∈ (*convex hull s*). *x* ∉ *s* ⟶ (∃ *y*∈(*convex hull s*). *norm*(*x* − *a*) < *norm*(*y* − *a*))
proof(*induct-tac rule*: *finite-induct*[*of s*])
fix *x s* **assume** *as*:*finite s x*∉*s* ∀ *x*∈*convex hull s*. *x* ∉ *s* ⟶ (∃ *y*∈*convex hull s*. *norm* (*x* − *a*) < *norm* (*y* − *a*))
show ∀ *xa*∈*convex hull insert x s*. *xa* ∉ *insert x s* ⟶ (∃ *y*∈*convex hull insert x s*. *norm* (*xa* − *a*) < *norm* (*y* − *a*))
proof(*rule*,*rule*,*cases s* = {})
case *False* **fix** *y* **assume** *y*:*y* ∈ *convex hull insert x s y* ∉ *insert x s*
obtain *u v b* **where** *obt*:*u*≥*0 v*≥*0 u* + *v* = *1 b* ∈ *convex hull s y* = *u* ∗*R x* + *v* ∗*R b*
using *y*(*1*)[*unfolded convex-hull-insert*[*OF False*]] **by** *auto*
show ∃ *z*∈*convex hull insert x s*. *norm* (*y* − *a*) < *norm* (*z* − *a*)
proof(*cases y*∈*convex hull s*)

```
      case True then obtain z where z∈convex hull s norm (y − a) < norm (z
− a)
        using as(3)[THEN bspec[where x=y]] and y(2) by auto
       thus ?thesis apply(rule-tac x=z in bexI) unfolding convex-hull-insert[OF
False] by auto
    next
      case False show ?thesis  using obt(3) proof(cases u=0, case-tac[!] v=0)
        assume u=0 v≠0 hence y = b using obt by auto
        thus ?thesis using False and obt(4) by auto
      next
        assume u≠0 v=0 hence y = x using obt by auto
        thus ?thesis using y(2) by auto
      next
        assume u≠0 v≠0
        then obtain w where w:w>0 w<u w<v using real-lbound-gt-zero[of u v]
and obt(1,2) by auto
        have x≠b proof(rule ccontr)
          assume ¬ x≠b hence y=b unfolding obt(5)
            using obt(3) by(auto simp add: scaleR-left-distrib[THEN sym])
          thus False using obt(4) and False by simp qed
        hence ∗:w ∗R (x − b) ≠ 0 using w(1) by auto
        show ?thesis using dist-increases-online[OF ∗, of a y]
        proof(erule-tac disjE)
          assume dist a y < dist a (y + w ∗R (x − b))
          hence norm (y − a) < norm ((u + w) ∗R x + (v − w) ∗R b − a)
              unfolding dist-commute[of a] unfolding dist-norm obt(5) by (simp
add: algebra-simps)
          moreover have (u + w) ∗R x + (v − w) ∗R b ∈ convex hull insert x s
            unfolding convex-hull-insert[OF ‹s≠{}›] and mem-Collect-eq
            apply(rule-tac x=u + w in exI) apply rule defer
            apply(rule-tac x=v − w in exI) using ‹u≥0› and w and obt(3,4) by
auto
          ultimately show ?thesis by auto
        next
          assume dist a y < dist a (y − w ∗R (x − b))
          hence norm (y − a) < norm ((u − w) ∗R x + (v + w) ∗R b − a)
              unfolding dist-commute[of a] unfolding dist-norm obt(5) by (simp
add: algebra-simps)
          moreover have (u − w) ∗R x + (v + w) ∗R b ∈ convex hull insert x s
            unfolding convex-hull-insert[OF ‹s≠{}›] and mem-Collect-eq
            apply(rule-tac x=u − w in exI) apply rule defer
            apply(rule-tac x=v + w in exI) using ‹u≥0› and w and obt(3,4) by
auto
          ultimately show ?thesis by auto
        qed
      qed auto
    qed
  qed auto
qed (auto simp add: assms)
```

lemma *simplex-furthest-le*:
fixes $s :: (real\ \hat{}\ -)\ set$
assumes *finite s* $s \neq \{\}$
shows $\exists y{\in}s.\ \forall x{\in}(convex\ hull\ s).\ norm(x - a) \leq norm(y - a)$
proof−
have *convex hull s* $\neq \{\}$ **using** *hull-subset*[*of s convex*] **and** *assms*(*2*) **by** *auto*
then obtain *x* **where** *x*:$x{\in}convex\ hull\ s\ \forall y{\in}convex\ hull\ s.\ norm\ (y - a) \leq norm\ (x - a)$
using *distance-attains-sup*[*OF finite-imp-compact-convex-hull*[*OF assms*(*1*)], *of a*]
unfolding *dist-commute*[*of a*] **unfolding** *dist-norm* **by** *auto*
thus *?thesis* **proof**(*cases* $x{\in}s$)
case *False* **then obtain** *y* **where** $y{\in}convex\ hull\ s\ norm\ (x - a) < norm\ (y - a)$
using *simplex-furthest-lt*[*OF assms*(*1*), *THEN bspec*[**where** *x*=*x*]] **and** *x*(*1*) **by** *auto*
thus *?thesis* **using** *x*(*2*)[*THEN bspec*[**where** *x*=*y*]] **by** *auto*
qed *auto*
qed

lemma *simplex-furthest-le-exists*:
fixes $s :: (real\ \hat{}\ -)\ set$
shows *finite s* $\Longrightarrow (\forall x{\in}(convex\ hull\ s).\ \exists y{\in}s.\ norm(x - a) \leq norm(y - a))$
using *simplex-furthest-le*[*of s*] **by** (*cases s*={})*auto*

lemma *simplex-extremal-le*:
fixes $s :: (real\ \hat{}\ -)\ set$
assumes *finite s* $s \neq \{\}$
shows $\exists u{\in}s.\ \exists v{\in}s.\ \forall x{\in}convex\ hull\ s.\ \forall y \in convex\ hull\ s.\ norm(x - y) \leq norm(u - v)$
proof−
have *convex hull s* $\neq \{\}$ **using** *hull-subset*[*of s convex*] **and** *assms*(*2*) **by** *auto*
then obtain *u v* **where** *obt*:$u{\in}convex\ hull\ s\ v{\in}convex\ hull\ s$
$\forall x{\in}convex\ hull\ s.\ \forall y{\in}convex\ hull\ s.\ norm\ (x - y) \leq norm\ (u - v)$
using *compact-sup-maxdistance*[*OF finite-imp-compact-convex-hull*[*OF assms*(*1*)]]
by *auto*
thus *?thesis* **proof**(*cases* $u{\notin}s \vee v{\notin}s$, *erule-tac disjE*)
assume $u{\notin}s$ **then obtain** *y* **where** $y{\in}convex\ hull\ s\ norm\ (u - v) < norm\ (y - v)$
using *simplex-furthest-lt*[*OF assms*(*1*), *THEN bspec*[**where** *x*=*u*]] **and** *obt*(*1*) **by** *auto*
thus *?thesis* **using** *obt*(*3*)[*THEN bspec*[**where** *x*=*y*], *THEN bspec*[**where** *x*=*v*]] **and** *obt*(*2*) **by** *auto*
next
assume $v{\notin}s$ **then obtain** *y* **where** $y{\in}convex\ hull\ s\ norm\ (v - u) < norm\ (y - u)$
using *simplex-furthest-lt*[*OF assms*(*1*), *THEN bspec*[**where** *x*=*v*]] **and** *obt*(*2*) **by** *auto*

thus *?thesis* **using** *obt*(*3*)[*THEN bspec*[**where** *x*=*u*], *THEN bspec*[**where** *x*=*y*]]
and *obt*(*1*)
by (*auto simp add*: *norm-minus-commute*)
qed *auto*
qed

lemma *simplex-extremal-le-exists*:
fixes *s* :: (*real* ^ -) *set*
shows *finite s* $\Longrightarrow$ *x* $\in$ *convex hull s* $\Longrightarrow$ *y* $\in$ *convex hull s*
$\Longrightarrow$ ($\exists$ *u*$\in$*s*. $\exists$ *v*$\in$*s*. *norm*(*x* $-$ *y*) $\leq$ *norm*(*u* $-$ *v*))
using *convex-hull-empty simplex-extremal-le*[*of s*] **by**(*cases s*={})*auto*

19.21 Closest point of a convex set is unique, with a continuous projection.

definition
closest-point :: *'a*::{*real-inner*,*heine-borel*} *set* $\Rightarrow$ *'a* $\Rightarrow$ *'a* **where**
closest-point s a = (*SOME x*. *x* $\in$ *s* $\wedge$ ($\forall$ *y*$\in$*s*. *dist a x* $\leq$ *dist a y*))

lemma *closest-point-exists*:
assumes *closed s s* $\neq$ {}
shows *closest-point s a* $\in$ *s* $\forall$ *y*$\in$*s*. *dist a* (*closest-point s a*) $\leq$ *dist a y*
unfolding *closest-point-def* **apply**(*rule-tac*[!] *someI2-ex*)
using *distance-attains-inf*[*OF assms*(*1*,*2*), *of a*] **by** *auto*

lemma *closest-point-in-set*:
closed s $\Longrightarrow$ *s* $\neq$ {} $\Longrightarrow$ (*closest-point s a*) $\in$ *s*
by(*meson closest-point-exists*)

lemma *closest-point-le*:
closed s $\Longrightarrow$ *x* $\in$ *s* $\Longrightarrow$ *dist a* (*closest-point s a*) $\leq$ *dist a x*
using *closest-point-exists*[*of s*] **by** *auto*

lemma *closest-point-self*:
assumes *x* $\in$ *s* **shows** *closest-point s x* = *x*
unfolding *closest-point-def* **apply**(*rule some1-equality*, *rule ex1I*[*of* - *x*])
using *assms* **by** *auto*

lemma *closest-point-refl*:
closed s $\Longrightarrow$ *s* $\neq$ {} $\Longrightarrow$ (*closest-point s x* = *x* $\longleftrightarrow$ *x* $\in$ *s*)
using *closest-point-in-set*[*of s x*] *closest-point-self*[*of x s*] **by** *auto*

lemma *norm-lt*: *norm x* < *norm y* $\longleftrightarrow$ *inner x x* < *inner y y*
unfolding *norm-eq-sqrt-inner* **by** *simp*

lemma *norm-le*: *norm x* $\leq$ *norm y* $\longleftrightarrow$ *inner x x* $\leq$ *inner y y*
unfolding *norm-eq-sqrt-inner* **by** *simp*

lemma *closer-points-lemma*:
assumes *inner y z > 0*
shows $\exists u{>}0.\ \forall v{>}0.\ v \leq u \longrightarrow norm(v *_R z - y) < norm\ y$
proof− **have** *z*:*inner z z > 0* **unfolding** *inner-gt-zero-iff* **using** *assms* **by** *auto*
thus *?thesis* **using** *assms* **apply**(*rule-tac x*=*inner y z / inner z z* **in** *exI*) **apply**(*rule*) **defer proof**(*rule*+)
fix *v* **assume** *0*<*v v* ≤ *inner y z / inner z z*
thus *norm* (*v* *$_R$ *z* − *y*) < *norm y* **unfolding** *norm-lt* **using** *z* **and** *assms*
by (*simp add*: *field-simps inner-diff inner-commute mult-strict-left-mono*[*OF* - ‹*0*<*v*›])
qed(*rule divide-pos-pos*, *auto*) **qed**

lemma *closer-point-lemma*:
assumes *inner* (*y* − *x*) (*z* − *x*) > *0*
shows $\exists u{>}0.\ u \leq 1 \wedge dist\ (x + u *_R (z - x))\ y < dist\ x\ y$
proof− **obtain** *u* **where** *u*>*0* **and** *u*:∀ *v*>*0*. *v* ≤ *u* ⟶ *norm* (*v* *$_R$ (*z* − *x*) − (*y* − *x*)) < *norm* (*y* − *x*)
using *closer-points-lemma*[*OF assms*] **by** *auto*
show *?thesis* **apply**(*rule-tac x*=*min u 1* **in** *exI*) **using** *u*[*THEN spec*[**where** *x*=*min u 1*]] **and** ‹*u*>*0*›
unfolding *dist-norm* **by**(*auto simp add*: *norm-minus-commute field-simps*) **qed**

lemma *any-closest-point-dot*:
assumes *convex s closed s x* ∈ *s y* ∈ *s* ∀ *z*∈*s*. *dist a x* ≤ *dist a z*
shows *inner* (*a* − *x*) (*y* − *x*) ≤ *0*
proof(*rule ccontr*) **assume** ¬ *inner* (*a* − *x*) (*y* − *x*) ≤ *0*
then obtain *u* **where** *u*:*u*>*0* *u*≤*1 dist* (*x* + *u* *$_R$ (*y* − *x*)) *a* < *dist x a* **using** *closer-point-lemma*[*of a x y*] **by** *auto*
let *?z* = (*1* − *u*) *$_R$ *x* + *u* *$_R$ *y* **have** *?z* ∈ *s* **using** *mem-convex*[*OF assms*(*1*,*3*,*4*), *of u*] **using** *u* **by** *auto*
thus *False* **using** *assms*(*5*)[*THEN bspec*[**where** *x*=*?z*]] **and** *u*(*3*) **by** (*auto simp add*: *dist-commute algebra-simps*) **qed**

lemma *any-closest-point-unique*:
fixes *x* :: *'a*::*real-inner*
assumes *convex s closed s x* ∈ *s y* ∈ *s*
∀ *z*∈*s*. *dist a x* ≤ *dist a z* ∀ *z*∈*s*. *dist a y* ≤ *dist a z*
shows *x* = *y* **using** *any-closest-point-dot*[*OF assms*(*1*−*4*,*5*)] **and** *any-closest-point-dot*[*OF assms*(*1*−*2*,*4*,*3*,*6*)]
unfolding *norm-pths*(*1*) **and** *norm-le-square*
by (*auto simp add*: *algebra-simps*)

lemma *closest-point-unique*:
assumes *convex s closed s x* ∈ *s* ∀ *z*∈*s*. *dist a x* ≤ *dist a z*
shows *x* = *closest-point s a*
using *any-closest-point-unique*[*OF assms*(*1*−*3*) - *assms*(*4*), *of closest-point s a*]

using *closest-point-exists*[*OF assms*(*2*)] **and** *assms*(*3*) **by** *auto*

lemma *closest-point-dot*:
assumes *convex s closed s x* ∈ *s*
shows *inner* (*a* − *closest-point s a*) (*x* − *closest-point s a*) ≤ *0*
apply(*rule any-closest-point-dot*[*OF assms*(*1*,*2*) - *assms*(*3*)])
using *closest-point-exists*[*OF assms*(*2*)] **and** *assms*(*3*) **by** *auto*

lemma *closest-point-lt*:
assumes *convex s closed s x* ∈ *s x* ≠ *closest-point s a*
shows *dist a* (*closest-point s a*) < *dist a x*
apply(*rule ccontr*) **apply**(*rule-tac notE*[*OF assms*(*4*)])
apply(*rule closest-point-unique*[*OF assms*(*1*−*3*), *of a*])
using *closest-point-le*[*OF assms*(*2*), *of* - *a*] **by** *fastsimp*

lemma *closest-point-lipschitz*:
assumes *convex s closed s s* ≠ {}
shows *dist* (*closest-point s x*) (*closest-point s y*) ≤ *dist x y*
proof−
have *inner* (*x* − *closest-point s x*) (*closest-point s y* − *closest-point s x*) ≤ *0*
inner (*y* − *closest-point s y*) (*closest-point s x* − *closest-point s y*) ≤ *0*
apply(*rule-tac*[!] *any-closest-point-dot*[*OF assms*(*1*−*2*)])
using *closest-point-exists*[*OF assms*(*2*−*3*)] **by** *auto*
thus *?thesis* **unfolding** *dist-norm* **and** *norm-le*
using *inner-ge-zero*[*of* (*x* − *closest-point s x*) − (*y* − *closest-point s y*)]
by (*simp add*: *inner-add inner-diff inner-commute*) **qed**

lemma *continuous-at-closest-point*:
assumes *convex s closed s s* ≠ {}
shows *continuous* (*at x*) (*closest-point s*)
unfolding *continuous-at-eps-delta*
using *le-less-trans*[*OF closest-point-lipschitz*[*OF assms*]] **by** *auto*

lemma *continuous-on-closest-point*:
assumes *convex s closed s s* ≠ {}
shows *continuous-on t* (*closest-point s*)
by(*metis continuous-at-imp-continuous-on continuous-at-closest-point*[*OF assms*])

19.22 Various point-to-set separating/supporting hyperplane theorems.

lemma *supporting-hyperplane-closed-point*:
fixes *z* :: *'a*::{*real-inner*,*heine-borel*}
assumes *convex s closed s s* ≠ {} *z* ∉ *s*
shows ∃ *a b*. ∃ *y*∈*s*. *inner a z* < *b* ∧ (*inner a y* = *b*) ∧ (∀ *x*∈*s*. *inner a x* ≥ *b*)
proof−
from *distance-attains-inf*[*OF assms*(*2*−*3*)] **obtain** *y* **where** *y*∈*s* **and** *y*:∀ *x*∈*s*. *dist z y* ≤ *dist z x* **by** *auto*
show *?thesis* **apply**(*rule-tac x*=*y* − *z* **in** *exI*, *rule-tac x*=*inner* (*y* − *z*) *y* **in** *exI*, *rule-tac x*=*y* **in** *bexI*)

apply *rule* **defer apply** *rule* **defer apply**(*rule*, *rule ccontr*) **using** ‹*y*∈*s*›
proof−
show *inner* (*y* − *z*) *z* < *inner* (*y* − *z*) *y* **apply**(*subst diff-less-iff*(*1*)[*THEN sym*])
unfolding *inner-diff-right*[*THEN sym*] **and** *inner-gt-zero-iff* **using** ‹*y*∈*s*› ‹*z*∉*s*› **by** *auto*
next
fix *x* **assume** *x*∈*s* **have** ∗:∀ *u*. *0* ≤ *u* ∧ *u* ≤ *1* ⟶ *dist z y* ≤ *dist z* ((*1* − *u*) ∗*R y* + *u* ∗*R x*)
using *assms*(*1*)[*unfolded convex-alt*] **and** *y* **and** ‹*x*∈*s*› **and** ‹*y*∈*s*› **by** *auto*
assume ¬ *inner* (*y* − *z*) *y* ≤ *inner* (*y* − *z*) *x* **then obtain** *v* **where**
v>*0* *v*≤*1* *dist* (*y* + *v* ∗*R* (*x* − *y*)) *z* < *dist y z* **using** *closer-point-lemma*[*of z y x*] **apply** − **by** (*auto simp add*: *inner-diff*)
thus *False* **using** ∗[*THEN spec*[**where** *x*=*v*]] **by**(*auto simp add*: *dist-commute algebra-simps*)
qed *auto*
qed

lemma *separating-hyperplane-closed-point*:
fixes *z* :: *'a*::{*real-inner*,*heine-borel*}
assumes *convex s closed s z* ∉ *s*
shows ∃ *a b*. *inner a z* < *b* ∧ (∀ *x*∈*s*. *inner a x* > *b*)
proof(*cases s*={})
case *True* **thus** *?thesis* **apply**(*rule-tac x*=−*z* **in** *exI*, *rule-tac x*=*1* **in** *exI*)
using *less-le-trans*[*OF - inner-ge-zero*[*of z*]] **by** *auto*
next
case *False* **obtain** *y* **where** *y*∈*s* **and** *y*:∀ *x*∈*s*. *dist z y* ≤ *dist z x*
using *distance-attains-inf*[*OF assms*(*2*) *False*] **by** *auto*
show *?thesis* **apply**(*rule-tac x*=*y* − *z* **in** *exI*, *rule-tac x*=*inner* (*y* − *z*) *z* + (*norm*(*y* − *z*))^2 / *2* **in** *exI*)
apply *rule* **defer apply** *rule* **proof**−
fix *x* **assume** *x*∈*s*
have ¬ *0* < *inner* (*z* − *y*) (*x* − *y*) **apply**(*rule-tac notI*) **proof**(*drule closer-point-lemma*)
assume ∃ *u*>*0*. *u* ≤ *1* ∧ *dist* (*y* + *u* ∗*R* (*x* − *y*)) *z* < *dist y z*
then obtain *u* **where** *u*>*0* *u*≤*1* *dist* (*y* + *u* ∗*R* (*x* − *y*)) *z* < *dist y z* **by** *auto*
thus *False* **using** *y*[*THEN bspec*[**where** *x*=*y* + *u* ∗*R* (*x* − *y*)]]
using *assms*(*1*)[*unfolded convex-alt*, *THEN bspec*[**where** *x*=*y*]]
using ‹*x*∈*s*› ‹*y*∈*s*› **by** (*auto simp add*: *dist-commute algebra-simps*) **qed**
moreover have *0* < *norm* (*y* − *z*) ^ *2* **using** ‹*y*∈*s*› ‹*z*∉*s*› **by** *auto*
hence *0* < *inner* (*y* − *z*) (*y* − *z*) **unfolding** *power2-norm-eq-inner* **by** *simp*
ultimately show *inner* (*y* − *z*) *z* + (*norm* (*y* − *z*))^2 / *2* < *inner* (*y* − *z*) *x*
unfolding *power2-norm-eq-inner* **and** *not-less* **by** (*auto simp add*: *field-simps inner-commute inner-diff*)
qed(*insert* ‹*y*∈*s*› ‹*z*∉*s*›, *auto*)
qed

lemma *separating-hyperplane-closed-0*:
assumes *convex* (*s*::(*real*^*'n*) *set*) *closed s 0* ∉ *s*

shows $\exists a\ b.\ a \neq 0 \wedge 0 < b \wedge (\forall x{\in}s.\ inner\ a\ x > b)$
proof(*cases s*={}) **guess** *a* **using** *UNIV-witness*[**where** *'a*=*'n*] **..**
case *True* **have** *norm* ((*basis a*)::*real^'n*) = *1*
using *norm-basis* **and** *dimindex-ge-1* **by** *auto*
thus *?thesis* **apply**(*rule-tac x*=*basis a* **in** *exI*, *rule-tac x*=*1* **in** *exI*) **using** *True*
by *auto*
next case *False* **thus** *?thesis* **using** *False* **using** *separating-hyperplane-closed-point*[*OF assms*]
apply − **apply**(*erule exE*)+ **unfolding** *inner.zero-right* **apply**(*rule-tac x*=*a*
in *exI*, *rule-tac x*=*b* **in** *exI*) **by** *auto* **qed**

19.23 Now set-to-set for closed/compact sets.

lemma *separating-hyperplane-closed-compact*:
assumes *convex* (*s*::(*real^'n*) *set*) *closed s convex t compact t t* ≠ {} *s* ∩ *t* = {}
shows $\exists a\ b.\ (\forall x{\in}s.\ inner\ a\ x < b) \wedge (\forall x{\in}t.\ inner\ a\ x > b)$
proof(*cases s*={})
case *True*
obtain *b* **where** *b*:*b*>*0* ∀*x*∈*t*. *norm x* ≤ *b* **using** *compact-imp-bounded*[*OF assms*(*4*)] **unfolding** *bounded-pos* **by** *auto*
obtain *z*::*real^'n* **where** *z*:*norm z* = *b* + *1* **using** *vector-choose-size*[*of b* + *1*]
and *b*(*1*) **by** *auto*
hence *z*∉*t* **using** *b*(*2*)[*THEN bspec*[**where** *x*=*z*]] **by** *auto*
then obtain *a b* **where** *ab*:*inner a z* < *b* ∀*x*∈*t*. *b* < *inner a x*
using *separating-hyperplane-closed-point*[*OF assms*(*3*) *compact-imp-closed*[*OF assms*(*4*)], *of z*] **by** *auto*
thus *?thesis* **using** *True* **by** *auto*
next
case *False* **then obtain** *y* **where** *y*∈*s* **by** *auto*
obtain *a b* **where** *0* < *b* ∀*x*∈{*x* − *y* |*x y*. *x* ∈ *s* ∧ *y* ∈ *t*}. *b* < *inner a x*
using *separating-hyperplane-closed-point*[*OF convex-differences*[*OF assms*(*1*,*3*)], *of 0*]
using *closed-compact-differences*[*OF assms*(*2*,*4*)] **using** *assms*(*6*) **by**(*auto*, *blast*)
hence *ab*:∀*x*∈*s*. ∀*y*∈*t*. *b* + *inner a y* < *inner a x* **apply**− **apply**(*rule*,*rule*)
apply(*erule-tac x*=*x* − *y* **in** *ballE*) **by** (*auto simp add*: *inner-diff*)
def *k* ≡ *Sup* ((λ*x*. *inner a x*) ‘ *t*)
show *?thesis* **apply**(*rule-tac x*=−*a* **in** *exI*, *rule-tac x*=−(*k* + *b* / *2*) **in** *exI*)
apply(*rule*,*rule*) **defer apply**(*rule*) **unfolding** *inner-minus-left* **and** *neg-less-iff-less*
proof−
from *ab* **have** ((λ*x*. *inner a x*) ‘ *t*) *<= (*inner a y* − *b*)
apply(*erule-tac x*=*y* **in** *ballE*) **apply**(*rule setleI*) **using** ‹*y*∈*s*› **by** *auto*
hence *k*:*isLub UNIV* ((λ*x*. *inner a x*) ‘ *t*) *k* **unfolding** *k-def* **apply**(*rule-tac Sup*) **using** *assms*(*5*) **by** *auto*
fix *x* **assume** *x*∈*t* **thus** *inner a x* < (*k* + *b* / *2*) **using** ‹*0*<*b*› **and** *isLubD2*[*OF k*, *of inner a x*] **by** *auto*
next
fix *x* **assume** *x*∈*s*
hence *k* ≤ *inner a x* − *b* **unfolding** *k-def* **apply**(*rule-tac Sup-least*) **using**

assms(*5*)
using *ab*[*THEN bspec*[**where** *x*=*x*]] **by** *auto*
thus *k* + *b* / *2* < *inner a x* **using** ‹*0* < *b*› **by** *auto*
qed
qed

lemma *separating-hyperplane-compact-closed*:
fixes *s* :: (*real* ^ -) *set*
assumes *convex s compact s s* ≠ {} *convex t closed t s* ∩ *t* = {}
shows ∃ *a b*. (∀ *x*∈*s*. *inner a x* < *b*) ∧ (∀ *x*∈*t*. *inner a x* > *b*)
proof− **obtain** *a b* **where** (∀ *x*∈*t*. *inner a x* < *b*) ∧ (∀ *x*∈*s*. *b* < *inner a x*)
using *separating-hyperplane-closed-compact*[*OF assms*(*4*−*5*,*1*−*2*,*3*)] **and** *assms*(*6*)
by *auto*
thus *?thesis* **apply**(*rule-tac x*=−*a* **in** *exI*, *rule-tac x*=−*b* **in** *exI*) **by** *auto* **qed**

19.24 General case without assuming closure and getting non-strict separation.

lemma *separating-hyperplane-set-0*:
assumes *convex s* (*0*::*real*^*'n*) ∉ *s*
shows ∃ *a*. *a* ≠ *0* ∧ (∀ *x*∈*s*. *0* ≤ *inner a x*)
proof− **let** *?k* = λ*c*. {*x*::*real*^*'n*. *0* ≤ *inner c x*}
have *frontier* (*cball 0 1*) ∩ (⋂ (*?k* ‘ *s*)) ≠ {}
apply(*rule compact-imp-fip*) **apply**(*rule compact-frontier*[*OF compact-cball*])
defer apply(*rule*,*rule*,*erule conjE*) **proof**−
fix *f* **assume** *as*:*f* ⊆ *?k* ‘ *s finite f*
obtain *c* **where** *c*:*f* = *?k* ‘ *c c*⊆*s finite c* **using** *finite-subset-image*[*OF as*(*2*,*1*)]
by *auto*
then obtain *a b* **where** *ab*:*a* ≠ *0 0* < *b* ∀ *x*∈*convex hull c*. *b* < *inner a x*
using *separating-hyperplane-closed-0*[*OF convex-convex-hull*, *of c*]
using *finite-imp-compact-convex-hull*[*OF c*(*3*), *THEN compact-imp-closed*]
and *assms*(*2*)
using *subset-hull*[*unfolded mem-def*, *of convex*, *OF assms*(*1*), *THEN sym*, *of c*] **by** *auto*
hence ∃ *x*. *norm x* = *1* ∧ (∀ *y*∈*c*. *0* ≤ *inner y x*) **apply**(*rule-tac x*=*inverse*(*norm a*) *R a* **in** *exI*)
using *hull-subset*[*of c convex*] **unfolding** *subset-eq* **and** *inner-scaleR*
apply− **apply** *rule* **defer apply** *rule* **apply**(*rule mult-nonneg-nonneg*)
by(*auto simp add*: *inner-commute elim*!: *ballE*)
thus *frontier* (*cball 0 1*) ∩ ⋂*f* ≠ {} **unfolding** *c*(*1*) *frontier-cball dist-norm*
by *auto*
qed(*insert closed-halfspace-ge*, *auto*)
then obtain *x* **where** *norm x* = *1* ∀ *y*∈*s*. *x*∈*?k y* **unfolding** *frontier-cball dist-norm* **by** *auto*
thus *?thesis* **apply**(*rule-tac x*=*x* **in** *exI*) **by**(*auto simp add*: *inner-commute*)
qed

lemma *separating-hyperplane-sets*:
assumes *convex s convex* (*t*::(*real*^*'n*) *set*) *s* ≠ {} *t* ≠ {} *s* ∩ *t* = {}

shows $\exists a\ b.\ a \neq 0 \wedge (\forall x{\in}s.\ inner\ a\ x \leq b) \wedge (\forall x{\in}t.\ inner\ a\ x \geq b)$
proof− **from** *separating-hyperplane-set-0*[*OF convex-differences*[*OF assms*(*2*,*1*)]]
obtain *a* **where** $a{\neq}0\ \forall x{\in}\{x - y\ |x\ y.\ x \in t \wedge y \in s\}.\ 0 \leq inner\ a\ x$
using *assms*(*3*−*5*) **by** *auto*
hence $\forall x{\in}t.\ \forall y{\in}s.\ inner\ a\ y \leq inner\ a\ x$
by (*force simp add*: *inner-diff*)
thus *?thesis*
apply(*rule-tac x*=*a* **in** *exI*, *rule-tac x*=*Sup* ((λ*x*. *inner a x*) ' *s*) **in** *exI*) **using** ‹*a*≠*0*›
apply *auto*
apply (*rule Sup*[*THEN isLubD2*])
prefer *4*
apply (*rule Sup-least*)
using *assms*(*3*−*5*) **apply** (*auto simp add*: *setle-def*)
apply *metis*
done
qed

19.25 More convexity generalities.

lemma *convex-closure*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *convex s* **shows** *convex*(*closure s*)
unfolding *convex-def Ball-def closure-sequential*
apply(*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*rule*) **apply**(*erule-tac exE*)+
apply(*rule-tac x*=λ*n*. *u* $*_R$ *xb n* + *v* $*_R$ *xc n* **in** *exI*) **apply**(*rule*,*rule*)
apply(*rule assms*[*unfolded convex-def*, *rule-format*]) **prefer** *6*
apply(*rule Lim-add*) **apply**(*rule-tac* [*1*−*2*] *Lim-cmul*) **by** *auto*

lemma *convex-interior*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *convex s* **shows** *convex*(*interior s*)
unfolding *convex-alt Ball-def mem-interior* **apply**(*rule*,*rule*,*rule*,*rule*,*rule*,*rule*) **apply**(*erule exE* | *erule conjE*)+ **proof**−
fix *x y u* **assume** *u*:*0* ≤ *u u* ≤ (*1*::*real*)
fix *e d* **assume** *ed*:*ball x e* ⊆ *s ball y d* ⊆ *s* 0<*d* 0<*e*
show ∃ *e*>*0*. *ball* ((*1* − *u*) $*_R$ *x* + *u* $*_R$ *y*) *e* ⊆ *s* **apply**(*rule-tac x*=*min d e* **in** *exI*)
apply *rule* **unfolding** *subset-eq* **defer apply** *rule* **proof**−
fix *z* **assume** *z* ∈ *ball* ((*1* − *u*) $*_R$ *x* + *u* $*_R$ *y*) (*min d e*)
hence (*1*− *u*) $*_R$ (*z* − *u* $*_R$ (*y* − *x*)) + *u* $*_R$ (*z* + (*1* − *u*) $*_R$ (*y* − *x*)) ∈ *s*
apply(*rule-tac assms*[*unfolded convex-alt*, *rule-format*])
using *ed*(*1*,*2*) **and** *u* **unfolding** *subset-eq mem-ball Ball-def dist-norm* **by**(*auto simp add*: *algebra-simps*)
thus *z* ∈ *s* **using** *u* **by** (*auto simp add*: *algebra-simps*) **qed**(*insert u ed*(*3*−*4*), *auto*) **qed**

lemma *convex-hull-eq-empty*[*simp*]: *convex hull s* = {} ⟷ *s* = {}
using *hull-subset*[*of s convex*] *convex-hull-empty* **by** *auto*

19.26 Moving and scaling convex hulls.

lemma *convex-hull-translation-lemma*:
convex hull $((\lambda x.\ a + x)\ `\ s) \subseteq (\lambda x.\ a + x)\ `\ (\textit{convex hull}\ s)$
by (*metis convex-convex-hull convex-translation hull-minimal hull-subset image-mono mem-def*)

lemma *convex-hull-bilemma*: **fixes** *neg*
assumes $(\forall s\ a.\ (\textit{convex hull}\ (\textit{up}\ a\ s)) \subseteq \textit{up}\ a\ (\textit{convex hull}\ s))$
shows $(\forall s.\ \textit{up}\ a\ (\textit{up}\ (\textit{neg}\ a)\ s) = s) \wedge (\forall s.\ \textit{up}\ (\textit{neg}\ a)\ (\textit{up}\ a\ s) = s) \wedge (\forall s\ t\ a.\ s \subseteq t \longrightarrow \textit{up}\ a\ s \subseteq \textit{up}\ a\ t)$
$\Longrightarrow \forall s.\ (\textit{convex hull}\ (\textit{up}\ a\ s)) = \textit{up}\ a\ (\textit{convex hull}\ s)$
using *assms* **by**(*metis subset-antisym*)

lemma *convex-hull-translation*:
convex hull $((\lambda x.\ a + x)\ `\ s) = (\lambda x.\ a + x)\ `\ (\textit{convex hull}\ s)$
apply(*rule convex-hull-bilemma*[*rule-format*, *of* - - $\lambda a.\ -a$], *rule convex-hull-translation-lemma*)
unfolding *image-image* **by** *auto*

lemma *convex-hull-scaling-lemma*:
$(\textit{convex hull}\ ((\lambda x.\ c *_R x)\ `\ s)) \subseteq (\lambda x.\ c *_R x)\ `\ (\textit{convex hull}\ s)$
by (*metis convex-convex-hull convex-scaling hull-subset mem-def subset-hull subset-image-iff*)

lemma *convex-hull-scaling*:
convex hull $((\lambda x.\ c *_R x)\ `\ s) = (\lambda x.\ c *_R x)\ `\ (\textit{convex hull}\ s)$
apply(*cases c=0*) **defer apply**(*rule convex-hull-bilemma*[*rule-format*, *of* - - *inverse*]) **apply**(*rule convex-hull-scaling-lemma*)
unfolding *image-image scaleR-scaleR* **by**(*auto simp add*:*image-constant-conv*)

lemma *convex-hull-affinity*:
convex hull $((\lambda x.\ a + c *_R x)\ `\ s) = (\lambda x.\ a + c *_R x)\ `\ (\textit{convex hull}\ s)$
by(*simp only*: *image-image*[*THEN sym*] *convex-hull-scaling convex-hull-translation*)

19.27 Convex set as intersection of halfspaces.

lemma *convex-halfspace-intersection*:
fixes *s* :: (*real* ^ -) *set*
assumes *closed s convex s*
shows $s = \bigcap \{h.\ s \subseteq h \wedge (\exists a\ b.\ h = \{x.\ \textit{inner}\ a\ x \leq b\})\}$
apply(*rule set-ext*, *rule*) **unfolding** *Inter-iff Ball-def mem-Collect-eq* **apply**(*rule*,*rule*,*erule conjE*) **proof**−
fix *x* **assume** $\forall xa.\ s \subseteq xa \wedge (\exists a\ b.\ xa = \{x.\ \textit{inner}\ a\ x \leq b\}) \longrightarrow x \in xa$
hence $\forall a\ b.\ s \subseteq \{x.\ \textit{inner}\ a\ x \leq b\} \longrightarrow x \in \{x.\ \textit{inner}\ a\ x \leq b\}$ **by** *blast*
thus *x*∈*s* **apply**(*rule-tac ccontr*) **apply**(*drule separating-hyperplane-closed-point*[*OF assms*(*2*,*1*)])
apply(*erule exE*)+ **apply**(*erule-tac x*=−*a* **in** *allE*, *erule-tac x*=−*b* **in** *allE*)
by *auto*
qed *auto*

19.28 Radon's theorem (from Lars Schewe).

lemma *radon-ex-lemma*:
assumes *finite c affine-dependent c*
shows $\exists u.\ setsum\ u\ c = 0 \wedge (\exists v{\in}c.\ u\ v \neq 0) \wedge setsum\ (\lambda v.\ u\ v *_R v)\ c = 0$
proof− **from** *assms*(2)[*unfolded affine-dependent-explicit*] **guess** *s* **.. then guess** *u* **..**
thus *?thesis* **apply**(*rule-tac x*=$\lambda v.$ *if* $v{\in}s$ *then u v else 0* **in** *exI*) **unfolding** *if-smult scaleR-zero-left*
and *setsum-restrict-set*[*OF assms*(1), *THEN sym*] **by**(*auto simp add*: *Int-absorb1*)
qed

lemma *radon-s-lemma*:
assumes *finite s setsum f s* = (*0*::*real*)
shows *setsum f* $\{x{\in}s.\ 0 < f\ x\} = -$ *setsum f* $\{x{\in}s.\ f\ x < 0\}$
proof− **have** ∗:$\bigwedge x.$ (*if f x* < *0 then f x else 0*) + (*if 0* < *f x then f x else 0*) = *f x* **by** *auto*
show *?thesis* **unfolding** *real-add-eq-0-iff*[*THEN sym*] **and** *setsum-restrict-set''*[*OF assms*(1)] **and** *setsum-addf*[*THEN sym*] **and** ∗
using *assms*(2) **by** *assumption* **qed**

lemma *radon-v-lemma*:
assumes *finite s setsum f s* = *0* $\forall x.\ g\ x$ = (*0*::*real*) $\longrightarrow$ *f x* = (*0*::*real*^-)
shows (*setsum f* $\{x{\in}s.\ 0 < g\ x\}) = -$ *setsum f* $\{x{\in}s.\ g\ x < 0\}$
proof−
have ∗:$\bigwedge x.$ (*if 0* < *g x then f x else 0*) + (*if g x* < *0 then f x else 0*) = *f x* **using** *assms*(3) **by** *auto*
show *?thesis* **unfolding** *eq-neg-iff-add-eq-0* **and** *setsum-restrict-set''*[*OF assms*(1)] **and** *setsum-addf*[*THEN sym*] **and** ∗
using *assms*(2) **by** *assumption* **qed**

lemma *radon-partition*:
assumes *finite c affine-dependent c*
shows $\exists m\ p.\ m \cap p = \{\} \wedge m \cup p = c \wedge$ (*convex hull m*) $\cap$ (*convex hull p*) $\neq \{\}$ **proof**−
obtain *u v* **where** *uv*:*setsum u c* = *0* $v{\in}c$ *u v* $\neq$ *0* $(\sum v{\in}c.\ u\ v *_R v) = 0$ **using** *radon-ex-lemma*[*OF assms*] **by** *auto*
have *fin*:*finite* $\{x \in c.\ 0 < u\ x\}$ *finite* $\{x \in c.\ 0 > u\ x\}$ **using** *assms*(1) **by** *auto*
def $z \equiv (inverse\ (setsum\ u\ \{x{\in}c.\ u\ x > 0\})) *_R setsum\ (\lambda x.\ u\ x *_R x)\ \{x{\in}c.\ u\ x > 0\}$
have *setsum u* $\{x \in c.\ 0 < u\ x\} \neq 0$ **proof**(*cases u v* $\geq$ *0*)
case *False* **hence** *u v* < *0* **by** *auto*
thus *?thesis* **proof**(*cases* $\exists w{\in}\{x \in c.\ 0 < u\ x\}.\ u\ w > 0$)
case *True* **thus** *?thesis* **using** *setsum-nonneg-eq-0-iff*[*of - u, OF fin*(1)] **by** *auto*
next
case *False* **hence** *setsum u c* $\leq$ *setsum* ($\lambda x.$ *if* x=*v then u v else 0*) *c* **apply**(*rule-tac setsum-mono*) **by** *auto*
thus *?thesis* **unfolding** *setsum-delta*[*OF assms*(1)] **using** *uv*(2) **and** ‹*u v* <

0› **and** *uv(1)* **by** *auto* **qed**
qed (*insert setsum-nonneg-eq-0-iff*[*of - u, OF fin(1)*] *uv(2−3), auto*)

hence ∗:*setsum u* {*x*∈*c. u x > 0*} > *0* **unfolding** *less-le* **apply**(*rule-tac conjI, rule-tac setsum-nonneg*) **by** *auto*
moreover have *setsum u* ({*x* ∈ *c. 0 < u x*} ∪ {*x* ∈ *c. u x < 0*}) = *setsum u c* (∑ *x*∈{*x* ∈ *c. 0 < u x*} ∪ {*x* ∈ *c. u x < 0*}. *u x* ∗$_R$ *x*) = (∑ *x*∈*c. u x* ∗$_R$ *x*)
using *assms(1)* **apply**(*rule-tac*[!] *setsum-mono-zero-left*) **by** *auto*
hence *setsum u* {*x* ∈ *c. 0 < u x*} = − *setsum u* {*x* ∈ *c. 0 > u x*} (∑ *x*∈{*x* ∈ *c. 0 < u x*}. *u x* ∗$_R$ *x*) = − (∑ *x*∈{*x* ∈ *c. 0 > u x*}. *u x* ∗$_R$ *x*)
unfolding *eq-neg-iff-add-eq-0* **using** *uv(1,4)* **by** (*auto simp add*: *setsum-Un-zero*[*OF fin, THEN sym*])
moreover have ∀ *x*∈{*v* ∈ *c. u v < 0*}. *0* ≤ *inverse* (*setsum u* {*x* ∈ *c. 0 < u x*}) ∗ − *u x*
apply (*rule*) **apply** (*rule mult-nonneg-nonneg*) **using** ∗ **by** *auto*

ultimately have *z* ∈ *convex hull* {*v* ∈ *c. u v* ≤ *0*} **unfolding** *convex-hull-explicit mem-Collect-eq*
apply(*rule-tac x*={*v* ∈ *c. u v < 0*} **in** *exI*, *rule-tac x*=λ*y. inverse* (*setsum u* {*x*∈*c. u x > 0*}) ∗ − *u y* **in** *exI*)
using *assms(1)* **unfolding** *scaleR-scaleR*[*THEN sym*] *scaleR-right.setsum* [*symmetric*] **and** *z-def*
by(*auto simp add*: *setsum-negf vector-smult-lneg mult-right.setsum*[*THEN sym*])
moreover have ∀ *x*∈{*v* ∈ *c. 0 < u v*}. *0* ≤ *inverse* (*setsum u* {*x* ∈ *c. 0 < u x*}) ∗ *u x*
apply (*rule*) **apply** (*rule mult-nonneg-nonneg*) **using** ∗ **by** *auto*
hence *z* ∈ *convex hull* {*v* ∈ *c. u v > 0*} **unfolding** *convex-hull-explicit mem-Collect-eq*
apply(*rule-tac x*={*v* ∈ *c. 0 < u v*} **in** *exI*, *rule-tac x*=λ*y. inverse* (*setsum u* {*x*∈*c. u x > 0*}) ∗ *u y* **in** *exI*)
using *assms(1)* **unfolding** *scaleR-scaleR*[*THEN sym*] *scaleR-right.setsum* [*symmetric*] **and** *z-def* **using** ∗
by(*auto simp add*: *setsum-negf vector-smult-lneg mult-right.setsum*[*THEN sym*])
ultimately show *?thesis* **apply**(*rule-tac x*={*v*∈*c. u v* ≤ *0*} **in** *exI*, *rule-tac x*={*v*∈*c. u v > 0*} **in** *exI*) **by** *auto*
qed

lemma *radon*: **assumes** *affine-dependent c*
obtains *m p* **where** *m*⊆*c p*⊆*c m* ∩ *p* = {} (*convex hull m*) ∩ (*convex hull p*) ≠ {}
proof− **from** *assms*[*unfolded affine-dependent-explicit*] **guess** *s* **.. then guess** *u* **..**
hence ∗:*finite s affine-dependent s* **and** *s*:*s* ⊆ *c* **unfolding** *affine-dependent-explicit* **by** *auto*
from *radon-partition*[*OF* ∗] **guess** *m* **.. then guess** *p* **..**
thus *?thesis* **apply**(*rule-tac that*[*of p m*]) **using** *s* **by** *auto* **qed**

19.29 Helly's theorem.

lemma *helly-induct*: **fixes** *f*::(*real*^*'n*) *set set*

assumes *card f = n n ≥ CARD('n) + 1*
∀ s∈f. convex s ∀ t⊆f. card t = CARD('n) + 1 ⟶ ⋂ t ≠ {}
shows *⋂ f ≠ {}*
using *assms* **proof**(*induct n arbitrary*: *f*)
case (*Suc n*)
have *finite f* **using** ‹*card f = Suc n*› **by** (*auto intro*: *card-ge-0-finite*)
show *⋂ f ≠ {}* **apply**(*cases n = CARD('n)*) **apply**(*rule Suc(5)*[*rule-format*])
unfolding ‹*card f = Suc n*› **proof**−
assume *ng*:*n ≠ CARD('n)* **hence** *∃ X. ∀ s∈f. X s ∈ ⋂(f − {s})* **apply**(*rule-tac bchoice*) **unfolding** *ex-in-conv*
apply(*rule*, *rule Suc(1)*[*rule-format*]) **unfolding** *card-Diff-singleton-if*[*OF* ‹*finite f*›] ‹*card f = Suc n*›
defer defer apply(*rule Suc(4)*[*rule-format*]) **defer apply**(*rule Suc(5)*[*rule-format*])
using *Suc(3)* ‹*finite f*› **by** *auto*
then obtain *X* **where** *X*:*∀ s∈f. X s ∈ ⋂(f − {s})* **by** *auto*
show *?thesis* **proof**(*cases inj-on X f*)
case *False* **then obtain** *s t* **where** *st*:*s≠t s∈f t∈f X s = X t* **unfolding** *inj-on-def* **by** *auto*
hence *∗*:*⋂ f = ⋂ (f − {s}) ∩ ⋂ (f − {t})* **by** *auto*
show *?thesis* **unfolding** *∗* **unfolding** *ex-in-conv*[*THEN sym*] **apply**(*rule-tac x*=*X s* **in** *exI*)
apply(*rule*, *rule X*[*rule-format*]) **using** *X st* **by** *auto*
next case *True* **then obtain** *m p* **where** *mp*:*m ∩ p = {} m ∪ p = X ‘ f convex hull m ∩ convex hull p ≠ {}*
using *radon-partition*[*of X ‘ f*] **and** *affine-dependent-biggerset*[*of X ‘ f*]
unfolding *card-image*[*OF True*] **and** ‹*card f = Suc n*› **using** *Suc(3)* ‹*finite f*› **and** *ng* **by** *auto*
have *m ⊆ X ‘ f p ⊆ X ‘ f* **using** *mp(2)* **by** *auto*
then obtain *g h* **where** *gh*:*m = X ‘ g p = X ‘ h g ⊆ f h ⊆ f* **unfolding** *subset-image-iff* **by** *auto*
hence *f ∪ (g ∪ h) = f* **by** *auto*
hence *f*:*f = g ∪ h* **using** *inj-on-Un-image-eq-iff*[*of X f g ∪ h*] **and** *True*
unfolding *mp(2)*[*unfolded image-Un*[*THEN sym*] *gh*] **by** *auto*
have *∗*:*g ∩ h = {}* **using** *mp(1)* **unfolding** *gh* **using** *inj-on-image-Int*[*OF True gh(3,4)*] **by** *auto*
have *convex hull (X ‘ h) ⊆ ⋂ g convex hull (X ‘ g) ⊆ ⋂ h*
apply(*rule-tac* [!] *hull-minimal*) **using** *Suc gh(3−4)* **unfolding** *mem-def* **unfolding** *subset-eq*
apply(*rule-tac* [2] *convex-Inter*, *rule-tac* [4] *convex-Inter*) **apply** *rule* **prefer** *3* **apply** *rule* **proof**−
fix *x* **assume** *x∈X ‘ g* **then guess** *y* **unfolding** *image-iff* **..**
thus *x∈⋂h* **using** *X*[*THEN bspec*[**where** *x*=*y*]] **using** *∗ f* **by** *auto* **next**
fix *x* **assume** *x∈X ‘ h* **then guess** *y* **unfolding** *image-iff* **..**
thus *x∈⋂ g* **using** *X*[*THEN bspec*[**where** *x*=*y*]] **using** *∗ f* **by** *auto*
qed(*auto*)
thus *?thesis* **unfolding** *f* **using** *mp(3)*[*unfolded gh*] **by** *blast* **qed**
qed(*insert dimindex-ge-1*, *auto*) **qed**(*auto*)

lemma *helly*: **fixes** *f*::(*real^'n*) *set set*

assumes *card f* $\geq$ *CARD*('*n*) + *1* $\forall s{\in}f.$ *convex s*
$\forall t{\subseteq}f.$ *card t* = *CARD*('*n*) + *1* $\longrightarrow \bigcap$ *t* $\neq$ {}
shows $\bigcap$ *f* $\neq${}
apply(*rule helly-induct*) **using** *assms* **by** *auto*

19.30 Convex hull is "preserved" by a linear function.

lemma *convex-hull-linear-image*:
assumes *bounded-linear f*
shows *f* ' (*convex hull s*) = *convex hull* (*f* ' *s*)
apply *rule* **unfolding** *subset-eq ball-simps* **apply**(*rule-tac*[!] *hull-induct*, *rule hull-inc*) **prefer** *3*
apply(*erule imageE*)**apply**(*rule-tac x*=*xa* **in** *image-eqI*) **apply** *assumption*
apply(*rule hull-subset*[*unfolded subset-eq*, *rule-format*]) **apply** *assumption*
proof−
interpret *f*: *bounded-linear f* **by** *fact*
show *convex* {*x*. *f x* $\in$ *convex hull f* ' *s*}
unfolding *convex-def* **by**(*auto simp add*: *f.scaleR f.add convex-convex-hull*[*unfolded convex-def*, *rule-format*]) **next**
interpret *f*: *bounded-linear f* **by** *fact*
show *convex* {*x*. *x* $\in$ *f* ' (*convex hull s*)} **using** *convex-convex-hull*[*unfolded convex-def*, *of s*]
unfolding *convex-def* **by** (*auto simp add*: *f.scaleR* [*symmetric*] *f.add* [*symmetric*])
qed *auto*

lemma *in-convex-hull-linear-image*:
assumes *bounded-linear f x* $\in$ *convex hull s*
shows (*f x*) $\in$ *convex hull* (*f* ' *s*)
using *convex-hull-linear-image*[*OF assms*(*1*)] *assms*(*2*) **by** *auto*

19.31 Homeomorphism of all convex compact sets with nonempty interior.

lemma *compact-frontier-line-lemma*:
fixes *s* :: (*real* ^ -) *set*
assumes *compact s 0* $\in$ *s x* $\neq$ *0*
obtains *u* **where** *0* $\leq$ *u* (*u* $*_R$ *x*) $\in$ *frontier s* $\forall v{>}u.$ (*v* $*_R$ *x*) $\notin$ *s*
proof−
obtain *b* **where** *b*:*b*>*0* $\forall x{\in}s.$ *norm x* $\leq$ *b* **using** *compact-imp-bounded*[*OF assms*(*1*), *unfolded bounded-pos*] **by** *auto*
let *?A* = {*y*. $\exists u.$ *0* $\leq$ *u* $\wedge$ *u* $\leq$ *b* / *norm*(*x*) $\wedge$ (*y* = *u* $*_R$ *x*)}
have *A*:*?A* = ($\lambda u.$ *u* $*_R$ *x*) ' {*0* .. *b* / *norm x*}
by *auto*
have *compact ?A* **unfolding** *A* **apply**(*rule compact-continuous-image*, *rule continuous-at-imp-continuous-on*)
apply(*rule*, *rule continuous-vmul*)
apply(*rule continuous-at-id*) **by**(*rule compact-real-interval*)
moreover have {*y*. $\exists u{\geq}0.$ *u* $\leq$ *b* / *norm x* $\wedge$ *y* = *u* $*_R$ *x*} $\cap$ *s* $\neq$ {} **apply**(*rule not-disjointI*[*OF* - *assms*(*2*)])
unfolding *mem-Collect-eq* **using** ‹*b*>*0*› *assms*(*3*) **by**(*auto intro*!: *divide-nonneg-pos*)

ultimately obtain u y **where** *obt*: $u{\geq}0$ $u \leq b$ / *norm* x $y = u *_R x$ $y{\in}?A$ $y{\in}s$ $\forall z{\in}?A \cap s$. *dist* 0 $z \leq$ *dist* 0 y **using** *distance-attains-sup*[*OF compact-inter*[*OF* - *assms*(1), *of* ?*A*], *of* 0] **by** *auto*

have *norm* $x > 0$ **using** *assms*(3)[*unfolded zero-less-norm-iff*[*THEN sym*]] **by** *auto*
{ fix v **assume** *as*:$v > u$ $v *_R x \in s$
hence $v \leq b$ / *norm* x **using** *b*(2)[*rule-format*, *OF as*(2)]
using ‹$u{\geq}0$› **unfolding** *pos-le-divide-eq*[*OF* ‹*norm* $x > 0$›] **by** *auto*
hence *norm* $(v *_R x) \leq$ *norm* y **apply**(*rule-tac obt*(6)[*rule-format*, *unfolded dist-0-norm*]) **apply**(*rule IntI*) **defer**
apply(*rule as*(2)) **unfolding** *mem-Collect-eq* **apply**(*rule-tac* $x{=}v$ **in** *exI*)
using *as*(1) ‹$u{\geq}0$› **by**(*auto simp add*:*field-simps*)
hence *False* **unfolding** *obt*(3) **using** ‹$u{\geq}0$› ‹*norm* $x > 0$› ‹$v{>}u$› **by**(*auto simp add*:*field-simps*)
} **note** *u-max* = *this*

have $u *_R x \in$ *frontier* s **unfolding** *frontier-straddle* **apply**(*rule*,*rule*,*rule*) **apply**(*rule-tac* $x{=}u *_R x$ **in** *bexI*) **unfolding** *obt*(3)[*THEN sym*]
prefer 3 **apply**(*rule-tac* $x{=}(u + (e$ / 2) / *norm* $x) *_R x$ **in** *exI*) **apply**(*rule*, *rule*) **proof**−
fix e **assume** $0 < e$ **and** *as*:$(u + e$ / 2 / *norm* $x) *_R x \in s$
hence $u + e$ / 2 / *norm* $x > u$ **using**‹*norm* $x > 0$› **by**(*auto simp del*:*zero-less-norm-iff intro*!: *divide-pos-pos*)
thus *False* **using** *u-max*[*OF* - *as*] **by** *auto*
qed(*insert* ‹$y{\in}s$›, *auto simp add*: *dist-norm scaleR-left-distrib obt*(3))
thus *?thesis* **by**(*metis that*[*of u*] *u-max obt*(1))
qed

lemma *starlike-compact-projective*:
assumes *compact* s *cball* $(0$::*real*^'*n*$)$ $1 \subseteq s$
$\forall x{\in}s.\ \forall u.\ 0 \leq u \wedge u < 1 \longrightarrow (u *_R x) \in (s -$ *frontier* s $)$
shows s *homeomorphic* $($*cball* $(0$::*real*^'*n*$)$ 1$)$
proof−
have *fs*:*frontier* $s \subseteq s$ **apply**(*rule frontier-subset-closed*) **using** *compact-imp-closed*[*OF assms*(1)] **by** *simp*
def *pi* $\equiv \lambda x$::*real*^'*n*. *inverse* $($*norm* $x) *_R x$
have $0 \notin$ *frontier* s **unfolding** *frontier-straddle* **apply**(*rule ccontr*) **unfolding** *not-not* **apply**(*erule-tac* $x{=}1$ **in** *allE*)
using *assms*(2)[*unfolded subset-eq Ball-def mem-cball*] **by** *auto*
have *injpi*:$\bigwedge x\ y.$ *pi* $x =$ *pi* $y \wedge$ *norm* $x =$ *norm* $y \longleftrightarrow x = y$ **unfolding** *pi-def* **by** *auto*

have *contpi*:*continuous-on* $(UNIV - \{0\})$ *pi* **apply**(*rule continuous-at-imp-continuous-on*)
apply *rule* **unfolding** *pi-def*
apply (*rule continuous-mul*)
apply (*rule continuous-at-inv*[*unfolded o-def*])
apply (*rule continuous-at-norm*)
apply *simp*

apply (*rule continuous-at-id*)
done
def *sphere* ≡ {*x*::*real*^'*n*. *norm x* = *1*}
have *pi*:⋀*x*. *x* ≠ *0* ⟹ *pi x* ∈ *sphere* ⋀*x u*. *u*>*0* ⟹ *pi* (*u* *$_R$ *x*) = *pi x*
unfolding *pi-def sphere-def* **by** *auto*

have *0*∈*s* **using** *assms*(*2*) **and** *centre-in-cball*[*of 0 1*] **by** *auto*
have *front-smul*:∀ *x*∈*frontier s*. ∀ *u*≥*0*. *u* *$_R$ *x* ∈ *s* ⟷ *u* ≤ *1* **proof**(*rule*,*rule*,*rule*)
fix *x u* **assume** *x*:*x*∈*frontier s* **and** (*0*::*real*)≤*u*
hence *x*≠*0* **using** ‹*0*∉*frontier s*› **by** *auto*
obtain *v* **where** *v*:*0* ≤ *v v* *$_R$ *x* ∈ *frontier s* ∀ *w*>*v*. *w* *$_R$ *x* ∉ *s*
using *compact-frontier-line-lemma*[*OF assms*(*1*) ‹*0*∈*s*› ‹*x*≠*0*›] **by** *auto*
have *v*=*1* **apply**(*rule ccontr*) **unfolding** *neq-iff* **apply**(*erule disjE*) **proof**−
assume *v*<*1* **thus** *False* **using** *v*(*3*)[*THEN spec*[**where** *x*=*1*]] **using** *x* **and** *fs* **by** *auto* **next**
assume *v*>*1* **thus** *False* **using** *assms*(*3*)[*THEN bspec*[**where** *x*=*v* *$_R$ *x*], *THEN spec*[**where** *x*=*inverse v*]]
using *v* **and** *x* **and** *fs* **unfolding** *inverse-less-1-iff* **by** *auto* **qed**
show *u* *$_R$ *x* ∈ *s* ⟷ *u* ≤ *1* **apply** *rule* **using** *v*(*3*)[*unfolded* ‹*v*=*1*›, *THEN spec*[**where** *x*=*u*]] **proof**−
assume *u*≤*1* **thus** *u* *$_R$ *x* ∈ *s* **apply**(*cases u*=*1*)
using *assms*(*3*)[*THEN bspec*[**where** *x*=*x*], *THEN spec*[**where** *x*=*u*]] **using** ‹*0*≤*u*› **and** *x* **and** *fs* **by** *auto* **qed** *auto* **qed**

have ∃ *surf*. *homeomorphism* (*frontier s*) *sphere pi surf*
apply(*rule homeomorphism-compact*) **apply**(*rule compact-frontier*[*OF assms*(*1*)])
apply(*rule continuous-on-subset*[*OF contpi*]) **defer apply**(*rule set-ext*,*rule*)
unfolding *inj-on-def* **prefer** *3* **apply**(*rule*,*rule*,*rule*)
proof− **fix** *x* **assume** *x*∈*pi* ‘ *frontier s* **then obtain** *y* **where** *y*∈*frontier s x* = *pi y* **by** *auto*
thus *x* ∈ *sphere* **using** *pi*(*1*)[*of y*] **and** ‹*0* ∉ *frontier s*› **by** *auto*
next fix *x* **assume** *x*∈*sphere* **hence** *norm x* = *1 x*≠*0* **unfolding** *sphere-def* **by** *auto*
then obtain *u* **where** *0* ≤ *u u* *$_R$ *x* ∈ *frontier s* ∀ *v*>*u*. *v* *$_R$ *x* ∉ *s*
using *compact-frontier-line-lemma*[*OF assms*(*1*) ‹*0*∈*s*›, *of x*] **by** *auto*
thus *x* ∈ *pi* ‘ *frontier s* **unfolding** *image-iff le-less pi-def* **apply**(*rule-tac x*=*u* *$_R$ *x* **in** *bexI*) **using** ‹*norm x* = *1*› ‹*0*∉*frontier s*› **by** *auto*
next fix *x y* **assume** *as*:*x* ∈ *frontier s y* ∈ *frontier s pi x* = *pi y*
hence *xys*:*x*∈*s y*∈*s* **using** *fs* **by** *auto*
from *as*(*1*,*2*) **have** *nor*:*norm x* ≠ *0 norm y* ≠ *0* **using** ‹*0*∉*frontier s*› **by** *auto*

from *nor* **have** *x*:*x* = *norm x* *$_R$ ((*inverse* (*norm y*)) *$_R$ *y*) **unfolding** *as*(*3*)[*unfolded pi-def*, *THEN sym*] **by** *auto*
from *nor* **have** *y*:*y* = *norm y* *$_R$ ((*inverse* (*norm x*)) *$_R$ *x*) **unfolding** *as*(*3*)[*unfolded pi-def*] **by** *auto*
have *0* ≤ *norm y* ∗ *inverse* (*norm x*) *0* ≤ *norm x* ∗ *inverse* (*norm y*)
unfolding *divide-inverse*[*THEN sym*] **apply**(*rule-tac*[!] *divide-nonneg-pos*) **using** *nor* **by** *auto*
hence *norm x* = *norm y* **apply**(*rule-tac ccontr*) **unfolding** *neq-iff*

using *x y* **and** *front-smul*[*THEN bspec*, *OF as*(*1*), *THEN spec*[**where** *x*=*norm y* ∗ (*inverse* (*norm x*))]]
using *front-smul*[*THEN bspec*, *OF as*(*2*), *THEN spec*[**where** *x*=*norm x* ∗ (*inverse* (*norm y*))]]
using *xys nor* **by**(*auto simp add*:*field-simps divide-le-eq-1 divide-inverse*[*THEN sym*])
thus *x* = *y* **apply**(*subst injpi*[*THEN sym*]) **using** *as*(*3*) **by** *auto*
qed(*insert* ‹*0* ∉ *frontier s*›, *auto*)
then obtain *surf* **where** *surf*:∀ *x*∈*frontier s*. *surf* (*pi x*) = *x* *pi* ‘ *frontier s* = *sphere continuous-on* (*frontier s*) *pi*
∀ *y*∈*sphere*. *pi* (*surf y*) = *y surf* ‘ *sphere* = *frontier s continuous-on sphere surf*
unfolding *homeomorphism-def* **by** *auto*

have *cont-surfpi*:*continuous-on* (*UNIV* − {*0*}) (*surf* ∘ *pi*) **apply**(*rule continuous-on-compose*, *rule contpi*)
apply(*rule continuous-on-subset*[*of sphere*], *rule surf*(*6*)) **using** *pi*(*1*) **by** *auto*

{ fix *x* **assume** *as*:*x* ∈ *cball* (*0*::*real*^′*n*) *1*
have *norm x* ∗*R surf* (*pi x*) ∈ *s* **proof**(*cases x*=*0* ∨ *norm x* = *1*)
case *False* **hence** *pi x* ∈ *sphere norm x* < *1* **using** *pi*(*1*)[*of x*] *as* **by**(*auto simp add*: *dist-norm*)
thus *?thesis* **apply**(*rule-tac assms*(*3*)[*rule-format*, *THEN DiffD1*])
apply(*rule-tac fs*[*unfolded subset-eq*, *rule-format*])
unfolding *surf*(*5*)[*THEN sym*] **by** *auto*
next case *True* **thus** *?thesis* **apply** *rule* **defer unfolding** *pi-def* **apply**(*rule fs*[*unfolded subset-eq*, *rule-format*])
unfolding *surf*(*5*)[*unfolded sphere-def*, *THEN sym*] **using** ‹*0*∈*s*› **by** *auto*
qed } note *hom* = *this*

{ fix *x* **assume** *x*∈*s*
hence *x* ∈ (λ*x*. *norm x* ∗*R surf* (*pi x*)) ‘ *cball 0 1* **proof**(*cases x*=*0*)
case *True* **show** *?thesis* **unfolding** *image-iff True* **apply**(*rule-tac x*=*0* **in** *bexI*) **by** *auto*
next let *?a* = *inverse* (*norm* (*surf* (*pi x*)))
case *False* **hence** *invn*:*inverse* (*norm x*) ≠ *0* **by** *auto*
from *False* **have** *pix*:*pi x*∈*sphere* **using** *pi*(*1*) **by** *auto*
hence *pi* (*surf* (*pi x*)) = *pi x* **apply**(*rule-tac surf*(*4*)[*rule-format*]) **by** *assumption*
hence ∗∗:*norm x* ∗*R* (*?a* ∗*R surf* (*pi x*)) = *x* **apply**(*rule-tac scaleR-left-imp-eq*[*OF invn*]) **unfolding** *pi-def* **using** *invn* **by** *auto*
hence ∗:*?a* ∗ *norm x* > *0* **and** *?a* > *0* *?a* ≠ *0* **using** *surf*(*5*) ‹*0*∉*frontier s*› **apply** −
apply(*rule-tac mult-pos-pos*) **using** *False*[*unfolded zero-less-norm-iff*[*THEN sym*]] **by** *auto*
have *norm* (*surf* (*pi x*)) ≠ *0* **using** ∗∗ *False* **by** *auto*
hence *norm x* = *norm* ((*?a* ∗ *norm x*) ∗*R surf* (*pi x*))
unfolding *norm-scaleR abs-mult abs-norm-cancel abs-of-pos*[*OF* ‹*?a* > *0*›] **by** *auto*
moreover have *pi x* = *pi* ((*inverse* (*norm* (*surf* (*pi x*))) ∗ *norm x*) ∗*R surf*

(*pi x*))
unfolding *pi*(*2*)[*OF* ∗] *surf*(*4*)[*rule-format*, *OF pix*] **..**
moreover have *surf* (*pi x*) ∈ *frontier s* **using** *surf*(*5*) *pix* **by** *auto*
hence *dist 0* (*inverse* (*norm* (*surf* (*pi x*))) ∗$_R$ *x*) ≤ *1* **unfolding** *dist-norm*
using ∗∗ **and** ∗ **using** *front-smul*[*THEN bspec*[**where** *x*=*surf* (*pi x*)], *THEN spec*[**where** *x*=*norm x* ∗ *?a*]]
using *False* ‹*x*∈*s*› **by**(*auto simp add*:*field-simps*)
ultimately show *?thesis* **unfolding** *image-iff* **apply**(*rule-tac x*=*inverse* (*norm* (*surf*(*pi x*))) ∗$_R$ *x* **in** *bexI*)
apply(*subst injpi*[*THEN sym*]) **unfolding** *abs-mult abs-norm-cancel abs-of-pos*[*OF* ‹*?a* > *0*›]
unfolding *pi*(*2*)[*OF* ‹*?a* > *0*›] **by** *auto*
qed } **note** *hom2* = *this*

show *?thesis* **apply**(*subst homeomorphic-sym*) **apply**(*rule homeomorphic-compact*[**where** *f*=λ*x*. *norm x* ∗$_R$ *surf* (*pi x*)])
apply(*rule compact-cball*) **defer apply**(*rule set-ext*, *rule*, *erule imageE*, *drule hom*)
prefer *4* **apply**(*rule continuous-at-imp-continuous-on*, *rule*) **apply**(*rule-tac* [*3*] *hom2*) **proof**−
fix *x*::*real*^'*n* **assume** *as*:*x* ∈ *cball 0 1*
thus *continuous* (*at x*) (λ*x*. *norm x* ∗$_R$ *surf* (*pi x*)) **proof**(*cases x*=*0*)
case *False* **thus** *?thesis* **apply**(*rule-tac continuous-mul*, *rule-tac continuous-at-norm*)
using *cont-surfpi* **unfolding** *continuous-on-eq-continuous-at*[*OF open-delete*[*OF open-UNIV*]] *o-def* **by** *auto*
next guess *a* **using** *UNIV-witness*[**where** *'a* = *'n*] **..**
obtain *B* **where** *B*:∀ *x*∈*s*. *norm x* ≤ *B* **using** *compact-imp-bounded*[*OF assms*(*1*)] **unfolding** *bounded-iff* **by** *auto*
hence *B* > *0* **using** *assms*(*2*) **unfolding** *subset-eq* **apply**(*erule-tac x*=*basis a* **in** *ballE*) **defer apply**(*erule-tac x*=*basis a* **in** *ballE*)
unfolding *Ball-def mem-cball dist-norm* **by** (*auto simp add*: *norm-basis*[*unfolded One-nat-def*])
case *True* **show** *?thesis* **unfolding** *True continuous-at Lim-at* **apply**(*rule*,*rule*) **apply**(*rule-tac x*=*e* / *B* **in** *exI*)
apply(*rule*) **apply**(*rule divide-pos-pos*) **prefer** *3* **apply**(*rule*,*rule*,*erule conjE*)
unfolding *norm-zero scaleR-zero-left dist-norm diff-0-right norm-scaleR abs-norm-cancel* **proof**−
fix *e* **and** *x*::*real*^'*n* **assume** *as*:*norm x* < *e* / *B 0* < *norm x 0*<*e*
hence *surf* (*pi x*) ∈ *frontier s* **using** *pi*(*1*)[*of x*] **unfolding** *surf*(*5*)[*THEN sym*] **by** *auto*
hence *norm* (*surf* (*pi x*)) ≤ *B* **using** *B fs* **by** *auto*
hence *norm x* ∗ *norm* (*surf* (*pi x*)) ≤ *norm x* ∗ *B* **using** *as*(*2*) **by** *auto*
also have ... < *e* / *B* ∗ *B* **apply**(*rule mult-strict-right-mono*) **using** *as*(*1*) ‹*B*>*0*› **by** *auto*
also have ... = *e* **using** ‹*B*>*0*› **by** *auto*
finally show *norm x* ∗ *norm* (*surf* (*pi x*)) < *e* **by** *assumption*
qed(*insert* ‹*B*>*0*›, *auto*) **qed**
next { **fix** *x* **assume** *as*:*surf* (*pi x*) = *0*

have $x = 0$ **proof**(*rule ccontr*)
assume $x \neq 0$ **hence** $pi\ x \in sphere$ **using** $pi(1)$ **by** *auto*
hence $surf\ (pi\ x) \in frontier\ s$ **using** $surf(5)$ **by** *auto*
thus *False* **using** ‹$0 \notin frontier\ s$› **unfolding** *as* **by** *simp* **qed**
} **note** *surf-0 = this*
show *inj-on* $(\lambda x.\ norm\ x *_R surf\ (pi\ x))\ (cball\ 0\ 1)$ **unfolding** *inj-on-def* **proof**(*rule*,*rule*,*rule*)
fix $x\ y$ **assume** *as*:$x \in cball\ 0\ 1\ y \in cball\ 0\ 1\ norm\ x *_R surf\ (pi\ x) = norm\ y *_R surf\ (pi\ y)$
thus $x=y$ **proof**(*cases* $x=0 \vee y=0$)
case *True* **thus** *?thesis* **using** *as* **by**(*auto elim*: *surf-0*) **next**
case *False*
hence $pi\ (surf\ (pi\ x)) = pi\ (surf\ (pi\ y))$ **using** $as(3)$
using $pi(2)$[*of norm x surf* $(pi\ x)$] $pi(2)$[*of norm y surf* $(pi\ y)$] **by** *auto*
moreover have $pi\ x \in sphere\ pi\ y \in sphere$ **using** $pi(1)$ *False* **by** *auto*
ultimately have $*$:$pi\ x = pi\ y$ **using** $surf(4)$[*THEN bspec*[**where** x=$pi\ x$]] $surf(4)$[*THEN bspec*[**where** x=$pi\ y$]] **by** *auto*
moreover have $norm\ x = norm\ y$ **using** $as(3)$[*unfolded* $*$] **using** *False* **by**(*auto dest*:*surf-0*)
ultimately show *?thesis* **using** *injpi* **by** *auto* **qed qed**
qed *auto* **qed**

lemma *homeomorphic-convex-compact-lemma*: **fixes** s::$(real\hat{}'n)\ set$
assumes *convex s compact s cball 0 1* $\subseteq s$
shows *s homeomorphic* $(cball\ (0::real\hat{}'n)\ 1)$
apply(*rule starlike-compact-projective*[*OF assms*(2−3)]) **proof**(*rule*,*rule*,*rule*,*erule conjE*)
fix $x\ u$ **assume** *as*:$x \in s\ 0 \leq u\ u < (1::real)$
hence $u *_R x \in interior\ s$ **unfolding** *interior-def mem-Collect-eq*
apply(*rule-tac* x=*ball* $(u *_R x)\ (1 - u)$ **in** *exI*) **apply**(*rule*, *rule open-ball*)
unfolding *centre-in-ball* **apply** *rule* **defer apply**(*rule*) **unfolding** *mem-ball* **proof**−
fix y **assume** $dist\ (u *_R x)\ y < 1 - u$
hence $inverse\ (1 - u) *_R (y - u *_R x) \in s$
using *assms*(3) **apply**(*erule-tac subsetD*) **unfolding** *mem-cball dist-commute dist-norm*
unfolding *group-add-class.diff-0 group-add-class.diff-0-right norm-minus-cancel norm-scaleR*
apply (*rule mult-left-le-imp-le*[*of* $1 - u$])
unfolding *mult-assoc*[*symmetric*] **using** ‹$u<1$› **by** *auto*
thus $y \in s$ **using** *assms*(1)[*unfolded convex-def*, *rule-format*, *of inverse*$(1 - u) *_R (y - u *_R x)\ x\ 1 - u\ u$]
using *as* **unfolding** *scaleR-scaleR* **by** *auto* **qed** *auto*
thus $u *_R x \in s - frontier\ s$ **using** *frontier-def* **and** *interior-subset* **by** *auto* **qed**

lemma *homeomorphic-convex-compact-cball*: **fixes** e::*real* **and** s::$(real\hat{}'n)\ set$
assumes *convex s compact s interior s* $\neq \{\}\ 0 < e$
shows *s homeomorphic* $(cball\ (b::real\hat{}'n)\ e)$

proof− **obtain** *a* **where** *a*∈*interior s* **using** *assms*(*3*) **by** *auto*
then obtain *d* **where** *d*>*0* **and** *d*:*cball a d* ⊆ *s* **unfolding** *mem-interior-cball*
by *auto*
let *?d* = *inverse d* **and** *?n* = *0*::*real*^'*n*
have *cball ?n 1* ⊆ (λ*x*. *inverse d* ∗*R* (*x* − *a*)) ‘ *s*
apply(*rule*, *rule-tac x*=*d* ∗*R x* + *a* **in** *image-eqI*) **defer**
apply(*rule d*[*unfolded subset-eq*, *rule-format*]) **using** ‹*d*>*0*› **unfolding** *mem-cball*
dist-norm
by(*auto simp add*: *mult-right-le-one-le*)
hence (λ*x*. *inverse d* ∗*R* (*x* − *a*)) ‘ *s homeomorphic cball ?n 1*
using *homeomorphic-convex-compact-lemma*[*of* (λ*x*. *?d* ∗*R* −*a* + *?d* ∗*R x*) ‘
s, *OF convex-affinity compact-affinity*]
using *assms*(*1*,*2*) **by**(*auto simp add*: *uminus-add-conv-diff scaleR-right-diff-distrib*)
thus *?thesis* **apply**(*rule-tac homeomorphic-trans*[*OF - homeomorphic-balls*(*2*)[*of*
1 - ?n]])
apply(*rule homeomorphic-trans*[*OF homeomorphic-affinity*[*of ?d s ?d* ∗*R* −*a*]])
using ‹*d*>*0*› ‹*e*>*0*› **by**(*auto simp add*: *uminus-add-conv-diff scaleR-right-diff-distrib*)
qed

lemma *homeomorphic-convex-compact*: **fixes** *s*::(*real*^'*n*) *set* **and** *t*::(*real*^'*n*) *set*
assumes *convex s compact s interior s* ≠ {}
convex t compact t interior t ≠ {}
shows *s homeomorphic t*
using *assms* **by**(*meson zero-less-one homeomorphic-trans homeomorphic-convex-compact-cball*
homeomorphic-sym)

19.32 Epigraphs of convex functions.

definition *epigraph s* (*f*::- ⇒ *real*) = {*xy*. *fst xy* ∈ *s* ∧ *f* (*fst xy*) ≤ *snd xy*}

lemma *mem-epigraph*: (*x*, *y*) ∈ *epigraph s f* ⟷ *x* ∈ *s* ∧ *f x* ≤ *y* **unfolding**
epigraph-def **by** *auto*

lemma *convex-epigraph*:
convex(*epigraph s f*) ⟷ *convex-on s f* ∧ *convex s*
unfolding *convex-def convex-on-def*
unfolding *Ball-def split-paired-All epigraph-def*
unfolding *mem-Collect-eq fst-conv snd-conv fst-add snd-add fst-scaleR snd-scaleR*
Ball-def[*symmetric*]
apply *safe* **defer apply**(*erule-tac x*=*x* **in** *allE*,*erule-tac x*=*f x* **in** *allE*) **apply**
safe
apply(*erule-tac x*=*xa* **in** *allE*,*erule-tac x*=*f xa* **in** *allE*) **prefer** *3*
apply(*rule-tac y*=*u* ∗ *f a* + *v* ∗ *f aa* **in** *order-trans*) **defer by**(*auto intro*!:*mult-left-mono*
add-mono)

lemma *convex-epigraphI*:
convex-on s f ⟹ *convex s* ⟹ *convex*(*epigraph s f*)
unfolding *convex-epigraph* **by** *auto*

lemma *convex-epigraph-convex*:
convex s $\Longrightarrow$ *convex-on s f* $\longleftrightarrow$ *convex*(*epigraph s f*)
by(*simp add*: *convex-epigraph*)

19.33 Use this to derive general bound property of convex function.

lemma *convex-on*:
assumes *convex s*
shows *convex-on s f* $\longleftrightarrow$ ($\forall$ *k u x*. ($\forall$ *i*$\in${*1..k*::*nat*}. *0* $\leq$ *u i* $\wedge$ *x i* $\in$ *s*) $\wedge$ *setsum u* {*1..k*} = *1* $\longrightarrow$
f (*setsum* (λ*i*. *u i* $*_R$ *x i*) {*1..k*}) $\leq$ *setsum* (λ*i*. *u i* $*$ *f*(*x i*)) {*1..k*})
unfolding *convex-epigraph-convex*[*OF assms*] *convex epigraph-def Ball-def mem-Collect-eq*
unfolding *fst-setsum snd-setsum fst-scaleR snd-scaleR*
apply *safe*
apply (*drule-tac x*=*k* **in** *spec*)
apply (*drule-tac x*=*u* **in** *spec*)
apply (*drule-tac x*=λ*i*. (*x i*, *f* (*x i*)) **in** *spec*)
apply *simp*
using *assms*[*unfolded convex*] **apply** *simp*
apply(*rule-tac y*=$\sum$ *i* = *1..k*. *u i* $*$ *f* (*fst* (*x i*)) **in** *order-trans*)
defer apply(*rule setsum-mono*) **apply**(*erule-tac x*=*i* **in** *allE*) **unfolding** *real-scaleR-def*
apply(*rule mult-left-mono*)**using** *assms*[*unfolded convex*] **by** *auto*

19.34 Convexity of general and special intervals.

lemma *is-interval-convex*:
fixes *s* :: (*real* ^ -) *set*
assumes *is-interval s* **shows** *convex s*
unfolding *convex-def* **apply**(*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*rule*) **proof**−
fix *x y u v* **assume** *as*:*x* $\in$ *s y* $\in$ *s* *0* $\leq$ *u* *0* $\leq$ *v u* + *v* = (*1*::*real*)
hence $*$:*u* = *1* − *v* *1* − *v* $\geq$ *0* **and** $**$:*v* = *1* − *u* *1* − *u* $\geq$ *0* **by** *auto*
{ fix *a b* **assume** ¬ *b* $\leq$ *u* $*$ *a* + *v* $*$ *b*
hence *u* $*$ *a* < (*1* − *v*) $*$ *b* **unfolding** *not-le* **using** *as*(*4*) **by**(*auto simp add*: *field-simps*)
hence *a* < *b* **unfolding** $*$ **using** *as*(*4*) $*$(*2*) **apply**(*rule-tac mult-left-less-imp-less*[*of* *1* − *v*]) **by**(*auto simp add*: *field-simps*)
hence *a* $\leq$ *u* $*$ *a* + *v* $*$ *b* **unfolding** $*$ **using** *as*(*4*) **by** (*auto simp add*: *field-simps intro*!:*mult-right-mono*)
} moreover
{ fix *a b* **assume** ¬ *u* $*$ *a* + *v* $*$ *b* $\leq$ *a*
hence *v* $*$ *b* > (*1* − *u*) $*$ *a* **unfolding** *not-le* **using** *as*(*4*) **by**(*auto simp add*: *field-simps*)
hence *a* < *b* **unfolding** $*$ **using** *as*(*4*) **apply**(*rule-tac mult-left-less-imp-less*) **by**(*auto simp add*: *field-simps*)
hence *u* $*$ *a* + *v* $*$ *b* $\leq$ *b* **unfolding** $**$ **using** $**$(*2*) *as*(*3*) **by**(*auto simp add*: *field-simps intro*!:*mult-right-mono*) **}**
ultimately show *u* $*_R$ *x* + *v* $*_R$ *y* $\in$ *s* **apply**− **apply**(*rule assms*[*unfolded*

is-interval-def, *rule-format*, *OF as*(*1*,*2*)])
using *as*(*3*−) *dimindex-ge-1* **by** *auto* **qed**

lemma *is-interval-connected*:
fixes *s* :: (*real* ^ -) *set*
shows *is-interval s* ⟹ *connected s*
using *is-interval-convex convex-connected* **by** *auto*

lemma *convex-interval*: *convex* {*a* .. *b*} *convex* {*a*<..<*b*::*real*^'*n*}
apply(*rule-tac*[!] *is-interval-convex*) **using** *is-interval-interval* **by** *auto*

19.35 Another intermediate value theorem formulation.

lemma *ivt-increasing-component-on-1*: **fixes** *f*::*real* ⇒ *real*^'*n*
assumes *a* ≤ *b continuous-on* {*a* .. *b*} *f* (*f a*)$*k* ≤ *y y* ≤ (*f b*)$*k*
shows ∃ *x*∈{*a*..*b*}. (*f x*)$*k* = *y*
proof− **have** *f a* ∈ *f* ' {*a*..*b*} *f b* ∈ *f* ' {*a*..*b*} **apply**(*rule-tac*[!] *imageI*)
using *assms*(*1*) **by**(*auto simp add*: *vector-le-def*)
thus *?thesis* **using** *connected-ivt-component*[*of f* ' {*a*..*b*} *f a f b k y*]
using *connected-continuous-image*[*OF assms*(*2*) *convex-connected*[*OF convex-real-interval*(*5*)]]
using *assms* **by**(*auto intro*!: *imageI*) **qed**

lemma *ivt-increasing-component-1*: **fixes** *f*::*real* ⇒ *real*^'*n*
shows *a* ≤ *b* ⟹ ∀ *x*∈{*a* .. *b*}. *continuous* (*at x*) *f*
⟹ *f a*$*k* ≤ *y* ⟹ *y* ≤ *f b*$*k* ⟹ ∃ *x*∈{*a*..*b*}. (*f x*)$*k* = *y*
by(*rule ivt-increasing-component-on-1*)
(*auto simp add*: *continuous-at-imp-continuous-on*)

lemma *ivt-decreasing-component-on-1*: **fixes** *f*::*real* ⇒ *real*^'*n*
assumes *a* ≤ *b continuous-on* {*a* .. *b*} *f* (*f b*)$*k* ≤ *y y* ≤ (*f a*)$*k*
shows ∃ *x*∈{*a*..*b*}. (*f x*)$*k* = *y*
apply(*subst neg-equal-iff-equal*[*THEN sym*]) **unfolding** *vector-uminus-component*[*THEN sym*]
apply(*rule ivt-increasing-component-on-1*) **using** *assms* **using** *continuous-on-neg*
by *auto*

lemma *ivt-decreasing-component-1*: **fixes** *f*::*real* ⇒ *real*^'*n*
shows *a* ≤ *b* ⟹ ∀ *x*∈{*a* .. *b*}. *continuous* (*at x*) *f*
⟹ *f b*$*k* ≤ *y* ⟹ *y* ≤ *f a*$*k* ⟹ ∃ *x*∈{*a*..*b*}. (*f x*)$*k* = *y*
by(*rule ivt-decreasing-component-on-1*)
(*auto simp*: *continuous-at-imp-continuous-on*)

19.36 A bound within a convex hull, and so an interval.

lemma *convex-on-convex-hull-bound*:
assumes *convex-on* (*convex hull s*) *f* ∀ *x*∈*s*. *f x* ≤ *b*
shows ∀ *x*∈ *convex hull s*. *f x* ≤ *b* **proof**
fix *x* **assume** *x*∈*convex hull s*
then obtain *k u v* **where** *obt*:∀ *i*∈{*1*..*k*::*nat*}. *0* ≤ *u i* ∧ *v i* ∈ *s setsum u* {*1*..*k*}
= *1* (∑ *i* = *1*..*k*. *u i* ∗$_R$ *v i*) = *x*

unfolding *convex-hull-indexed mem-Collect-eq* **by** *auto*
have $(\sum i = 1..k.\ u\ i * f\ (v\ i)) \leq b$ **using** *setsum-mono*[*of* $\{1..k\}$ $\lambda i.\ u\ i * f\ (v\ i)$ $\lambda i.\ u\ i * b$]
unfolding *setsum-left-distrib*[*THEN sym*] *obt*(*2*) *mult-1* **apply**(*drule-tac meta-mp*) **apply**(*rule mult-left-mono*)
using *assms*(*2*) *obt*(*1*) **by** *auto*
thus $f\ x \leq b$ **using** *assms*(*1*)[*unfolded convex-on*[*OF convex-convex-hull*], *rule-format*, *of k u v*]
unfolding *obt*(*2*−*3*) **using** *obt*(*1*) **and** *hull-subset*[*unfolded subset-eq*, *rule-format*, *of - s*] **by** *auto* **qed**

lemma *unit-interval-convex-hull*:
$\{0::real\hat{}'n\ ..\ 1\} = convex\ hull\ \{x.\ \forall i.\ (x\$i = 0) \vee (x\$i = 1)\}$ (**is** *?int* = *convex hull ?points*)
proof− **have** *01*:$\{0,1\} \subseteq$ *convex hull ?points* **apply** *rule* **apply**(*rule-tac hull-subset*[*unfolded subset-eq*, *rule-format*]) **by** *auto*
{ fix *n x* **assume** $x \in \{0::real\hat{}'n\ ..\ 1\}$ $n \leq CARD('n)$ *card* $\{i.\ x\$i \neq 0\} \leq n$
hence $x \in$ *convex hull ?points* **proof**(*induct n arbitrary*: *x*)
case *0* **hence** $x = 0$ **apply**(*subst Cart-eq*) **apply** *rule* **by** *auto*
thus $x \in$ *convex hull ?points* **using** *01* **by** *auto*
next
case (*Suc n*) **show** $x \in$ *convex hull ?points* **proof**(*cases* $\{i.\ x\$i \neq 0\} = \{\}$)
case *True* **hence** $x = 0$ **unfolding** *Cart-eq* **by** *auto*
thus $x \in$ *convex hull ?points* **using** *01* **by** *auto*
next
case *False* **def** *xi* ≡ *Min* $((\lambda i.\ x\$i)\ `\ \{i.\ x\$i \neq 0\})$
have *xi* $\in (\lambda i.\ x\$i)\ `\ \{i.\ x\$i \neq 0\}$ **unfolding** *xi-def* **apply**(*rule Min-in*) **using** *False* **by** *auto*
then obtain *i* **where** *i'*:$x\$i = xi\ x\$i \neq 0$ **by** *auto*
have *i*:$\bigwedge j.\ x\$j > 0 \Longrightarrow x\$i \leq x\$j$
unfolding *i'*(*1*) *xi-def* **apply**(*rule-tac Min-le*) **unfolding** *image-iff*
defer apply(*rule-tac x*=*j* **in** *bexI*) **using** *i'* **by** *auto*
have *i01*:$x\$i \leq 1\ x\$i > 0$ **using** *Suc*(*2*)[*unfolded mem-interval*,*rule-format*,*of i*] **using** *i'*(*2*) ‹$x\$i \neq 0$› **by** *auto*
show *?thesis* **proof**(*cases x*$*i*=*1*)
case *True* **have** $\forall j \in \{i.\ x\$i \neq 0\}.\ x\$j = 1$ **apply**(*rule*, *rule ccontr*) **unfolding** *mem-Collect-eq* **proof**−
fix *j* **assume** $x\ \$\ j \neq 0\ x\ \$\ j \neq 1$
hence *j*:$x\$j \in \{0<..<1\}$ **using** *Suc*(*2*) **by**(*auto simp add*: *vector-le-def elim*!:*allE*[**where** *x*=*j*])
hence $x\$j \in op\ \$\ x\ `\ \{i.\ x\ \$\ i \neq 0\}$ **by** *auto*
hence $x\$j \geq x\$i$ **unfolding** *i'*(*1*) *xi-def* **apply**(*rule-tac Min-le*) **by** *auto*
thus *False* **using** *True Suc*(*2*) *j* **by**(*auto simp add*: *vector-le-def elim*!:*ballE*[**where** *x*=*j*]) **qed**
thus *x*∈*convex hull ?points* **apply**(*rule-tac hull-subset*[*unfolded subset-eq*, *rule-format*])
by *auto*
next let *?y* = $\lambda j.$ *if* $x\$j = 0$ *then 0 else* $(x\$j - x\$i)\ /\ (1 - x\$i)$

case *False* **hence** $*{:}x = x\$i *_R (\chi\ j.\ \textit{if } x\$j = 0 \textit{ then } 0 \textit{ else } 1) + (1 - x\$i) *_R (\chi\ j.\ ?y\ j)$ **unfolding** *Cart-eq*
by(*auto simp add*: *field-simps*)
{ fix *j* **have** $x\$j \neq 0 \Longrightarrow 0 \leq (x\ \$\ j - x\ \$\ i) / (1 - x\ \$\ i)$ $(x\ \$\ j - x\ \$\ i) / (1 - x\ \$\ i) \leq 1$
apply(*rule-tac divide-nonneg-pos*) **using** *i*(*1*)[*of j*] **using** *False i01*
using *Suc*(*2*)[*unfolded mem-interval*, *rule-format*, *of j*] **by**(*auto simp add*:*field-simps*)
hence $0 \leq ?y\ j \wedge ?y\ j \leq 1$ **by** *auto* **}**
moreover have $i{\in}\{j.\ x\$j \neq 0\} - \{j.\ ((\chi\ j.\ ?y\ j){::}real\hat{}'n)\ \$\ j \neq 0\}$ **using** *i01* **by** *auto*
hence $\{j.\ x\$j \neq 0\} \neq \{j.\ ((\chi\ j.\ ?y\ j){::}real\hat{}'n)\ \$\ j \neq 0\}$ **by** *auto*
hence $**{:}\{j.\ ((\chi\ j.\ ?y\ j){::}real\hat{}'n)\ \$\ j \neq 0\} \subset \{j.\ x\$j \neq 0\}$ **apply** − **apply** *rule* **by** *auto*
have *card* $\{j.\ ((\chi\ j.\ ?y\ j){::}real\hat{}'n)\ \$\ j \neq 0\} \leq n$ **using** *less-le-trans*[*OF psubset-card-mono*[*OF - ***] *Suc*(*4*)] **by** *auto*
ultimately show *?thesis* **apply**(*subst* $*$) **apply**(*rule convex-convex-hull*[*unfolded convex-def*, *rule-format*])
apply(*rule-tac hull-subset*[*unfolded subset-eq*, *rule-format*]) **defer apply**(*rule Suc*(*1*))
unfolding *mem-interval* **using** *i01 Suc*(*3*) **by** *auto*
qed qed qed } note $* = this$
show *?thesis* **apply** *rule* **defer apply**(*rule hull-minimal*) **unfolding** *subset-eq* **prefer** *3* **apply** *rule*
apply(*rule-tac n2=CARD*('*n*) **in** $*$) **prefer** *3* **apply**(*rule card-mono*) **using** *01* **and** *convex-interval*(*1*) **prefer** *5* **apply** − **apply** *rule*
unfolding *mem-interval* **apply** *rule* **unfolding** *mem-Collect-eq* **apply**(*erule-tac x=i* **in** *allE*)
by(*auto simp add*: *vector-le-def mem-def*[*of - convex*]) **qed**

19.37 And this is a finite set of vertices.

lemma *unit-cube-convex-hull*: **obtains** *s* **where** *finite s* $\{0\ ..\ 1{::}real\hat{}'n\} = convex\ hull\ s$
apply(*rule that*[*of* $\{x{::}real\hat{}'n.\ \forall\ i.\ x\$i{=}0 \vee x\$i{=}1\}$])
apply(*rule finite-subset*[*of* - $(\lambda s.\ (\chi\ i.\ \textit{if } i{\in}s \textit{ then } 1{::}real \textit{ else } 0){::}real\hat{}'n)$ ' *UNIV*])
prefer *3* **apply**(*rule unit-interval-convex-hull*) **apply** *rule* **unfolding** *mem-Collect-eq*
proof−
fix $x{::}real\hat{}'n$ **assume** *as*:$\forall\ i.\ x\ \$\ i = 0 \vee x\ \$\ i = 1$
show $x \in (\lambda s.\ \chi\ i.\ \textit{if } i \in s \textit{ then } 1 \textit{ else } 0)$ ' *UNIV* **apply**(*rule image-eqI*[**where** $x{=}\{i.\ x\$i = 1\}$])
unfolding *Cart-eq* **using** *as* **by** *auto* **qed** *auto*

19.38 Hence any cube (could do any nonempty interval).

lemma *cube-convex-hull*:
assumes $0 < d$ **obtains** $s{::}(real\hat{}'n)\ set$ **where** *finite s* $\{x - (\chi\ i.\ d)\ ..\ x + (\chi\ i.\ d)\} = convex\ hull\ s$ **proof**−
let $?d = (\chi\ i.\ d){::}real\hat{}'n$

have $*$:$\{x - ?d \mathinner{..} x + ?d\} = (\lambda y.\ x - ?d + (2 * d) *_R y)$ ‘ $\{0 \mathinner{..} 1\}$ **apply**(*rule set-ext*, *rule*)
unfolding *image-iff* **defer apply**(*erule bexE*) **proof**−
fix y **assume** *as*:$y \in \{x - ?d \mathinner{..} x + ?d\}$
{ fix i::$'n$ **have** $x \$ i \leq d + y \$ i\ y \$ i \leq d + x \$ i$ **using** *as*[*unfolded mem-interval*, *THEN spec*[**where** x=i]]
by *auto*
hence $1 \geq inverse\ d * (x \$ i - y \$ i)\ 1 \geq inverse\ d * (y \$ i - x \$ i)$
apply(*rule-tac*[!] *mult-left-le-imp-le*[*OF - assms*]) **unfolding** *mult-assoc*[*THEN sym*]
using *assms* **by**(*auto simp add*: *field-simps*)
hence $inverse\ d * (x \$ i * 2) \leq 2 + inverse\ d * (y \$ i * 2)$
$inverse\ d * (y \$ i * 2) \leq 2 + inverse\ d * (x \$ i * 2)$ **by**(*auto simp add*:*field-simps*) **}**
hence $inverse\ (2 * d) *_R (y - (x - ?d)) \in \{0..1\}$ **unfolding** *mem-interval* **using** *assms*
by(*auto simp add*: *Cart-eq field-simps*)
thus $\exists z \in \{0..1\}.\ y = x - ?d + (2 * d) *_R z$ **apply**− **apply**(*rule-tac* x=$inverse\ (2 * d) *_R (y - (x - ?d))$ **in** *bexI*)
using *assms* **by**(*auto simp add*: *Cart-eq vector-le-def*)
next
fix $y\ z$ **assume** *as*:$z \in \{0..1\}\ y = x - ?d + (2*d) *_R z$
have $\bigwedge i.\ 0 \leq d * z \$ i \wedge d * z \$ i \leq d$ **using** *assms as*(*1*)[*unfolded mem-interval*] **apply**(*erule-tac* x=i **in** *allE*)
apply *rule* **apply**(*rule mult-nonneg-nonneg*) **prefer** *3* **apply**(*rule mult-right-le-one-le*)
using *assms* **by**(*auto simp add*: *Cart-eq*)
thus $y \in \{x - ?d..x + ?d\}$ **unfolding** *as*(*2*) *mem-interval* **apply**− **apply** *rule* **using** *as*(*1*)[*unfolded mem-interval*]
apply(*erule-tac* x=i **in** *allE*) **using** *assms* **by**(*auto simp add*: *Cart-eq*) **qed**
obtain s **where** *finite* s $\{0..1::real\hat{}'n\} = convex\ hull\ s$ **using** *unit-cube-convex-hull* **by** *auto*
thus *?thesis* **apply**(*rule-tac that*[*of* $(\lambda y.\ x - ?d + (2 * d) *_R y)$‘ s]) **unfolding** $*$ **and** *convex-hull-affinity* **by** *auto* **qed**

19.39 Bounded convex function on open set is continuous.

lemma *convex-on-bounded-continuous*:
fixes s :: ($'a$::*real-normed-vector*) *set*
assumes *open s convex-on s f* $\forall x \in s.\ abs(f\ x) \leq b$
shows *continuous-on s f*
apply(*rule continuous-at-imp-continuous-on*) **unfolding** *continuous-at-real-range*
proof(*rule*,*rule*,*rule*)
fix $x\ e$ **assume** $x \in s\ (0::real) < e$
def $B \equiv abs\ b + 1$
have B:$0 < B\ \bigwedge x.\ x \in s \Longrightarrow abs\ (f\ x) \leq B$
unfolding *B-def* **defer apply**(*drule assms*(*3*)[*rule-format*]) **by** *auto*
obtain k **where** k>0**and** k:*cball x k* $\subseteq s$ **using** *assms*(*1*)[*unfolded open-contains-cball*, *THEN bspec*[**where** x=x]] **using** ‹$x \in s$› **by** *auto*
show $\exists d > 0.\ \forall x'.\ norm\ (x' - x) < d \longrightarrow |f\ x' - f\ x| < e$

apply(*rule-tac x=min* (*k* / 2) (*e* / (2 * *B*) * *k*) **in** *exI*) **apply** *rule* **defer** **proof**(*rule*,*rule*)
fix *y* **assume** *as*:*norm* (*y* − *x*) < *min* (*k* / 2) (*e* / (2 * *B*) * *k*)
show |*f y* − *f x*| < *e* **proof**(*cases y=x*)
case *False* **def** *t* ≡ *k* / *norm* (*y* − *x*)
have 2 < *t* 0<*t* **unfolding** *t-def* **using** *as False* **and** ‹*k*>0› **by**(*auto simp add*:*field-simps*)
have *y*∈*s* **apply**(*rule k*[*unfolded subset-eq*,*rule-format*]) **unfolding** *mem-cball dist-norm*
apply(*rule order-trans*[*of* - 2 * *norm* (*x* − *y*)]) **using** *as* **by**(*auto simp add*: *field-simps norm-minus-commute*)
{ def *w* ≡ *x* + *t* *$_R$ (*y* − *x*)
have *w*∈*s* **unfolding** *w-def* **apply**(*rule k*[*unfolded subset-eq*,*rule-format*]) **unfolding** *mem-cball dist-norm*
unfolding *t-def* **using** ‹*k*>0› **by** *auto*
have (1 / *t*) *$_R$ *x* + − *x* + ((*t* − 1) / *t*) *$_R$ *x* = (1 / *t* − 1 + (*t* − 1) / *t*) *$_R$ *x* **by** (*auto simp add*: *algebra-simps*)
also have ... = 0 **using** ‹*t*>0› **by**(*auto simp add*:*field-simps*)
finally have *w*:(1 / *t*) *$_R$ *w* + ((*t* − 1) / *t*) *$_R$ *x* = *y* **unfolding** *w-def* **using** *False* **and** ‹*t*>0› **by** (*auto simp add*: *algebra-simps*)
have 2 * *B* < *e* * *t* **unfolding** *t-def* **using** ‹0<*e*› ‹0<*k*› ‹*B*>0› **and** *as* **and** *False* **by** (*auto simp add*:*field-simps*)
hence (*f w* − *f x*) / *t* < *e*
using *B*(2)[*OF* ‹*w*∈*s*›] **and** *B*(2)[*OF* ‹*x*∈*s*›] **using** ‹*t*>0› **by**(*auto simp add*:*field-simps*)
hence *th1*:*f y* − *f x* < *e* **apply**− **apply**(*rule le-less-trans*) **defer apply** *assumption*
using *assms*(2)[*unfolded convex-on-def*,*rule-format*,*of w x* 1/*t* (*t* − 1)/*t*, *unfolded w*]
using ‹0<*t*› ‹2<*t*› **and** ‹*x*∈*s*› ‹*w*∈*s*› **by**(*auto simp add*:*field-simps*) **}**
moreover
{ def *w* ≡ *x* − *t* *$_R$ (*y* − *x*)
have *w*∈*s* **unfolding** *w-def* **apply**(*rule k*[*unfolded subset-eq*,*rule-format*]) **unfolding** *mem-cball dist-norm*
unfolding *t-def* **using** ‹*k*>0› **by** *auto*
have (1 / (1 + *t*)) *$_R$ *x* + (*t* / (1 + *t*)) *$_R$ *x* = (1 / (1 + *t*) + *t* / (1 + *t*)) *$_R$ *x* **by** (*auto simp add*: *algebra-simps*)
also have ...=*x* **using** ‹*t*>0› **by** (*auto simp add*:*field-simps*)
finally have *w*:(1 / (1+*t*)) *$_R$ *w* + (*t* / (1 + *t*)) *$_R$ *y* = *x* **unfolding** *w-def* **using** *False* **and** ‹*t*>0› **by** (*auto simp add*: *algebra-simps*)
have 2 * *B* < *e* * *t* **unfolding** *t-def* **using** ‹0<*e*› ‹0<*k*› ‹*B*>0› **and** *as* **and** *False* **by** (*auto simp add*:*field-simps*)
hence *:(*f w* − *f y*) / *t* < *e* **using** *B*(2)[*OF* ‹*w*∈*s*›] **and** *B*(2)[*OF* ‹*y*∈*s*›] **using** ‹*t*>0› **by**(*auto simp add*:*field-simps*)
have *f x* ≤ 1 / (1 + *t*) * *f w* + (*t* / (1 + *t*)) * *f y*
using *assms*(2)[*unfolded convex-on-def*,*rule-format*,*of w y* 1/(1+*t*) *t* / (1+*t*),*unfolded w*]
using ‹0<*t*› ‹2<*t*› **and** ‹*y*∈*s*› ‹*w*∈*s*› **by** (*auto simp add*:*field-simps*)
also have ... = (*f w* + *t* * *f y*) / (1 + *t*) **using** ‹*t*>0› **unfolding**

divide-inverse **by** (*auto simp add*:*field-simps*)
also have ... < *e* + *f y* **using** ‹*t*>*0*› * ‹*e*>*0*› **by**(*auto simp add*:*field-simps*)
finally have *f x* − *f y* < *e* **by** *auto* **}**
ultimately show *?thesis* **by** *auto*
qed(*insert* ‹*0*<*e*›, *auto*)
qed(*insert* ‹*0*<*e*› ‹*0*<*k*› ‹*0*<*B*›, *auto simp add*:*field-simps intro*!:*mult-pos-pos*)
qed

19.40 Upper bound on a ball implies upper and lower bounds.

lemma *scaleR-2*:
fixes *x* :: *'a*::*real-vector*
shows *scaleR 2 x* = *x* + *x*
unfolding *one-add-one-is-two* [*symmetric*] *scaleR-left-distrib* **by** *simp*

lemma *convex-bounds-lemma*:
fixes *x* :: *'a*::*real-normed-vector*
assumes *convex-on* (*cball x e*) *f* ∀ *y* ∈ *cball x e*. *f y* ≤ *b*
shows ∀ *y* ∈ *cball x e*. *abs*(*f y*) ≤ *b* + *2* * *abs*(*f x*)
apply(*rule*) **proof**(*cases 0* ≤ *e*) **case** *True*
fix *y* **assume** *y*:*y*∈*cball x e* **def** *z* ≡ *2* *$_R$ *x* − *y*
have *:*x* − (*2* *$_R$ *x* − *y*) = *y* − *x* **by** (*simp add*: *scaleR-2*)
have *z*:*z*∈*cball x e* **using** *y* **unfolding** *z-def mem-cball dist-norm* * **by**(*auto simp add*: *norm-minus-commute*)
have (*1* / *2*) *$_R$ *y* + (*1* / *2*) *$_R$ *z* = *x* **unfolding** *z-def* **by** (*auto simp add*: *algebra-simps*)
thus |*f y*| ≤ *b* + *2* * |*f x*| **using** *assms*(*1*)[*unfolded convex-on-def*,*rule-format*, *OF y z*, *of 1*/*2 1*/*2*]
using *assms*(*2*)[*rule-format*,*OF y*] *assms*(*2*)[*rule-format*,*OF z*] **by**(*auto simp add*:*field-simps*)
next case *False* **fix** *y* **assume** *y*∈*cball x e*
hence *dist x y* < *0* **using** *False* **unfolding** *mem-cball not-le* **by** (*auto simp del*: *dist-not-less-zero*)
thus |*f y*| ≤ *b* + *2* * |*f x*| **using** *zero-le-dist*[*of x y*] **by** *auto* **qed**

19.41 Hence a convex function on an open set is continuous.

lemma *convex-on-continuous*:
assumes *open* (*s*::(*real^'n*) *set*) *convex-on s f*
shows *continuous-on s f*
unfolding *continuous-on-eq-continuous-at*[*OF assms*(*1*)] **proof**
note *dimge1* = *dimindex-ge-1*[**where** *'a*=*'n*]
fix *x* **assume** *x*∈*s*
then obtain *e* **where** *e*:*cball x e* ⊆ *s e*>*0* **using** *assms*(*1*) **unfolding** *open-contains-cball* **by** *auto*
def *d* ≡ *e* / *real CARD*(*'n*)
have *0* < *d* **unfolding** *d-def* **using** ‹*e*>*0*› *dimge1* **by**(*rule-tac divide-pos-pos*, *auto*)
let *?d* = (χ *i*. *d*)::*real^'n*

obtain *c* **where** *c*:*finite c* $\{x - ?d..x + ?d\} = convex\ hull\ c$ **using** *cube-convex-hull*[*OF* ‹*d*>*0*›, *of x*] **by** *auto*
have $x \in \{x - ?d..x + ?d\}$ **using** ‹*d*>*0*› **unfolding** *mem-interval* **by** *auto*
hence $c \neq \{\}$ **using** *c* **by** *auto*
def $k \equiv Max\ (f\ `\ c)$
have *convex-on* $\{x - ?d..x + ?d\}\ f$ **apply**(*rule convex-on-subset*[*OF assms*(*2*)])
apply(*rule subset-trans*[*OF - e*(*1*)]) **unfolding** *subset-eq mem-cball* **proof**
fix *z* **assume** *z*:$z \in \{x - ?d..x + ?d\}$
have *e*:$e = setsum\ (\lambda i.\ d)\ (UNIV::'n\ set)$ **unfolding** *setsum-constant d-def* **using** *dimge1*
by (*metis eq-divide-imp times-divide-eq-left real-dimindex-gt-0 real-eq-of-nat less-le mult-commute*)
show *dist x z* $\leq$ *e* **unfolding** *dist-norm e* **apply**(*rule-tac order-trans*[*OF norm-le-l1*], *rule setsum-mono*)
using *z*[*unfolded mem-interval*] **apply**(*erule-tac x*=*i* **in** *allE*) **by** *auto* **qed**
hence *k*:$\forall\ y \in \{x - ?d..x + ?d\}.\ f\ y \leq k$ **unfolding** *c*(*2*) **apply**(*rule-tac convex-on-convex-hull-bound*) **apply** *assumption*
unfolding *k-def* **apply**(*rule*, *rule Max-ge*) **using** *c*(*1*) **by** *auto*
have *d* $\leq$ *e* **unfolding** *d-def* **apply**(*rule mult-imp-div-pos-le*) **using** ‹*e*>*0*› *dimge1* **unfolding** *mult-le-cancel-left1* **using** *real-dimindex-ge-1* **by** *auto*
hence *dsube*:*cball x d* $\subseteq$ *cball x e* **unfolding** *subset-eq Ball-def mem-cball* **by** *auto*
have *conv*:*convex-on* (*cball x d*) *f* **apply**(*rule convex-on-subset*, *rule convex-on-subset*[*OF assms*(*2*)]) **apply**(*rule e*(*1*)) **using** *dsube* **by** *auto*
hence $\forall\ y \in cball\ x\ d.\ abs\ (f\ y) \leq k + 2 * abs\ (f\ x)$ **apply**(*rule-tac convex-bounds-lemma*) **apply** *assumption* **proof**
fix *y* **assume** *y*:$y \in cball\ x\ d$
{ **fix** *i*::$'n$ **have** $x\ \$\ i - d \leq y\ \$\ i$ $y\ \$\ i \leq x\ \$\ i + d$
using *order-trans*[*OF component-le-norm y*[*unfolded mem-cball dist-norm*], *of i*] **by** *auto* }
thus *f y* $\leq$ *k* **apply**(*rule-tac k*[*rule-format*]) **unfolding** *mem-cball mem-interval dist-norm*
by *auto* **qed**
hence *continuous-on* (*ball x d*) *f* **apply**(*rule-tac convex-on-bounded-continuous*)
apply(*rule open-ball*, *rule convex-on-subset*[*OF conv*], *rule ball-subset-cball*)
apply *force*
done
thus *continuous* (*at x*) *f* **unfolding** *continuous-on-eq-continuous-at*[*OF open-ball*]
using ‹*d*>*0*› **by** *auto*
qed

19.42 Line segments, Starlike Sets, etc.

definition
midpoint :: $'a$::*real-vector* $\Rightarrow$ $'a \Rightarrow$ $'a$ **where**
midpoint a b = (*inverse* (*2*::*real*)) $*_R$ $(a + b)$

definition
open-segment :: $'a$::*real-vector* $\Rightarrow$ $'a \Rightarrow$ $'a$ *set* **where**

open-segment a b = $\{(1 - u) *_R a + u *_R b \mid u$::*real*. $0 < u \wedge u < 1\}$

definition
closed-segment :: $'a$::*real-vector* $\Rightarrow$ $'a \Rightarrow$ $'a$ *set* **where**
closed-segment a b = $\{(1 - u) *_R a + u *_R b \mid u$::*real*. $0 \leq u \wedge u \leq 1\}$

definition *between* = $(\lambda\ (a,b).$ *closed-segment a b*$)$

lemmas *segment* = *open-segment-def closed-segment-def*

definition *starlike s* $\longleftrightarrow$ $(\exists\, a{\in}s.\ \forall\, x{\in}s.$ *closed-segment a x* $\subseteq$ *s*$)$

lemma *midpoint-refl*: *midpoint x x* = *x*
unfolding *midpoint-def* **unfolding** *scaleR-right-distrib* **unfolding** *scaleR-left-distrib*[*THEN sym*] **by** *auto*

lemma *midpoint-sym*: *midpoint a b* = *midpoint b a* **unfolding** *midpoint-def* **by** (*auto simp add*: *scaleR-right-distrib*)

lemma *midpoint-eq-iff*: *midpoint a b* = *c* $\longleftrightarrow$ *a* + *b* = *c* + *c*
proof −
have *midpoint a b* = *c* $\longleftrightarrow$ *scaleR 2* (*midpoint a b*) = *scaleR 2 c*
by *simp*
thus *?thesis*
unfolding *midpoint-def scaleR-2* [*symmetric*] **by** *simp*
qed

lemma *dist-midpoint*:
fixes *a b* :: $'a$::*real-normed-vector* **shows**
dist a (*midpoint a b*) = (*dist a b*) / *2* (**is** *?t1*)
dist b (*midpoint a b*) = (*dist a b*) / *2* (**is** *?t2*)
dist (*midpoint a b*) *a* = (*dist a b*) / *2* (**is** *?t3*)
dist (*midpoint a b*) *b* = (*dist a b*) / *2* (**is** *?t4*)
proof−
have $*$: $\bigwedge x\ y{::}'a.\ 2 *_R x = - y \Longrightarrow$ *norm x* = (*norm y*) / *2* **unfolding** *equation-minus-iff* **by** *auto*
have $**$:$\bigwedge x\ y{::}'a.\ 2 *_R x = \ \ y \Longrightarrow$ *norm x* = (*norm y*) / *2* **by** *auto*
note *scaleR-right-distrib* [*simp*]
show *?t1* **unfolding** *midpoint-def dist-norm* **apply** (*rule* $**$)
by (*simp add*: *scaleR-right-diff-distrib*, *simp add*: *scaleR-2*)
show *?t2* **unfolding** *midpoint-def dist-norm* **apply** (*rule* $*$)
by (*simp add*: *scaleR-right-diff-distrib*, *simp add*: *scaleR-2*)
show *?t3* **unfolding** *midpoint-def dist-norm* **apply** (*rule* $*$)
by (*simp add*: *scaleR-right-diff-distrib*, *simp add*: *scaleR-2*)
show *?t4* **unfolding** *midpoint-def dist-norm* **apply** (*rule* $**$)
by (*simp add*: *scaleR-right-diff-distrib*, *simp add*: *scaleR-2*)
qed

lemma *midpoint-eq-endpoint*:

midpoint a b = a ⟷ a = b
midpoint a b = b ⟷ a = b
unfolding *midpoint-eq-iff* **by** *auto*

lemma *convex-contains-segment*:
convex s ⟷ (∀ a∈s. ∀ b∈s. closed-segment a b ⊆ s)
unfolding *convex-alt closed-segment-def* **by** *auto*

lemma *convex-imp-starlike*:
convex s ⟹ s ≠ {} ⟹ starlike s
unfolding *convex-contains-segment starlike-def* **by** *auto*

lemma *segment-convex-hull*:
closed-segment a b = convex hull {a,b} **proof**−
have ∗:⋀*x*. {*x*} ≠ {} **by** *auto*
have ∗∗:⋀*u v*. *u* + *v* = *1* ⟷ *u* = *1* − (*v*::*real*) **by** *auto*
show *?thesis* **unfolding** *segment convex-hull-insert*[*OF* ∗] *convex-hull-singleton*
apply(*rule set-ext*)
unfolding *mem-Collect-eq* **apply**(*rule*,*erule exE*)
apply(*rule-tac x=1 − u* **in** *exI*) **apply** *rule* **defer apply**(*rule-tac x=u* **in** *exI*)
defer
apply(*erule exE*, (*erule conjE*)*?*)+ **apply**(*rule-tac x=1 − u* **in** *exI*) **unfolding**
∗∗ **by** *auto* **qed**

lemma *convex-segment*: *convex* (*closed-segment a b*)
unfolding *segment-convex-hull* **by**(*rule convex-convex-hull*)

lemma *ends-in-segment*: *a* ∈ *closed-segment a b b* ∈ *closed-segment a b*
unfolding *segment-convex-hull* **apply**(*rule-tac*[!] *hull-subset*[*unfolded subset-eq*,
rule-format]) **by** *auto*

lemma *segment-furthest-le*:
fixes *a b x y* :: *real* ^ *'n*
assumes *x* ∈ *closed-segment a b* **shows** *norm*(*y* − *x*) ≤ *norm*(*y* − *a*) ∨ *norm*(*y*
− *x*) ≤ *norm*(*y* − *b*) **proof**−
obtain *z* **where** *z*∈{*a*, *b*} *norm* (*x* − *y*) ≤ *norm* (*z* − *y*) **using** *simplex-furthest-le*[*of*
{*a*, *b*} *y*]
using *assms*[*unfolded segment-convex-hull*] **by** *auto*
thus *?thesis* **by**(*auto simp add*:*norm-minus-commute*) **qed**

lemma *segment-bound*:
fixes *x a b* :: *real* ^ *'n*
assumes *x* ∈ *closed-segment a b*
shows *norm*(*x* − *a*) ≤ *norm*(*b* − *a*) *norm*(*x* − *b*) ≤ *norm*(*b* − *a*)
using *segment-furthest-le*[*OF assms*, *of a*]
using *segment-furthest-le*[*OF assms*, *of b*]
by (*auto simp add*:*norm-minus-commute*)

lemma *segment-refl*:*closed-segment a a* = {*a*} **unfolding** *segment* **by** (*auto simp*

add: *algebra-simps*)

lemma *between-mem-segment*: *between* (*a*,*b*) *x* ⟷ *x* ∈ *closed-segment a b*
unfolding *between-def mem-def* **by** *auto*

lemma *between*:*between* (*a*,*b*) (*x*::*real*^'*n*) ⟷ *dist a b* = (*dist a x*) + (*dist x b*)
proof(*cases a* = *b*)
case *True* **thus** *?thesis* **unfolding** *between-def split-conv mem-def*[*of x*, *symmetric*]
by(*auto simp add*:*segment-refl dist-commute*) **next**
case *False* **hence** *Fal*:*norm* (*a* − *b*) ≠ *0* **and** *Fal2*: *norm* (*a* − *b*) > *0* **by** *auto*
have ∗:⋀*u*. *a* − ((*1* − *u*) ∗*R a* + *u* ∗*R b*) = *u* ∗*R* (*a* − *b*) **by** (*auto simp add*: *algebra-simps*)
show *?thesis* **unfolding** *between-def split-conv mem-def*[*of x*, *symmetric*] *closed-segment-def mem-Collect-eq*
apply *rule* **apply**(*erule exE*, (*erule conjE*)+) **apply**(*subst dist-triangle-eq*)
proof−
fix *u* **assume** *as*:*x* = (*1* − *u*) ∗*R a* + *u* ∗*R b* *0* ≤ *u u* ≤ *1*
hence ∗:*a* − *x* = *u* ∗*R* (*a* − *b*) *x* − *b* = (*1* − *u*) ∗*R* (*a* − *b*)
unfolding *as*(*1*) **by**(*auto simp add*:*algebra-simps*)
show *norm* (*a* − *x*) ∗*R* (*x* − *b*) = *norm* (*x* − *b*) ∗*R* (*a* − *x*)
unfolding *norm-minus-commute*[*of x a*] ∗ *Cart-eq* **using** *as*(*2*,*3*)
by(*auto simp add*: *field-simps*)
next assume *as*:*dist a b* = *dist a x* + *dist x b*
have *norm* (*a* − *x*) / *norm* (*a* − *b*) ≤ *1* **unfolding** *divide-le-eq-1-pos*[*OF Fal2*] **unfolding** *as*[*unfolded dist-norm*] *norm-ge-zero* **by** *auto*
thus ∃ *u*. *x* = (*1* − *u*) ∗*R a* + *u* ∗*R b* ∧ *0* ≤ *u* ∧ *u* ≤ *1* **apply**(*rule-tac x*=*dist a x* / *dist a b* **in** *exI*)
unfolding *dist-norm Cart-eq* **apply**− **apply** *rule* **defer apply**(*rule*, *rule divide-nonneg-pos*) **prefer** *4* **proof** *rule*
fix *i*::'*n* **have** ((*1* − *norm* (*a* − *x*) / *norm* (*a* − *b*)) ∗*R a* + (*norm* (*a* − *x*) / *norm* (*a* − *b*)) ∗*R b*) $ *i* =
((*norm* (*a* − *b*) − *norm* (*a* − *x*)) ∗ (*a* $ *i*) + *norm* (*a* − *x*) ∗ (*b* $ *i*)) / *norm* (*a* − *b*)
using *Fal* **by**(*auto simp add*: *field-simps*)
also have ... = *x*$*i* **apply**(*rule divide-eq-imp*[*OF Fal*])
unfolding *as*[*unfolded dist-norm*] **using** *as*[*unfolded dist-triangle-eq Cart-eq*,*rule-format*, *of i*]
by(*auto simp add*:*field-simps*)
finally show *x* $ *i* = ((*1* − *norm* (*a* − *x*) / *norm* (*a* − *b*)) ∗*R a* + (*norm* (*a* − *x*) / *norm* (*a* − *b*)) ∗*R b*) $ *i* **by** *auto*
qed(*insert Fal2*, *auto*) **qed qed**

lemma *between-midpoint*: **fixes** *a*::*real*^'*n* **shows**
between (*a*,*b*) (*midpoint a b*) (**is** *?t1*)
between (*b*,*a*) (*midpoint a b*) (**is** *?t2*)
proof− **have** ∗:⋀*x y z*. *x* = (*1*/*2*::*real*) ∗*R z* ⟹ *y* = (*1*/*2*) ∗*R z* ⟹ *norm z* = *norm x* + *norm y* **by** *auto*
show *?t1 ?t2* **unfolding** *between midpoint-def dist-norm* **apply**(*rule-tac*[!] ∗)

by(*auto simp add*:*field-simps Cart-eq*) **qed**

lemma *between-mem-convex-hull*:
between (*a*,*b*) *x* ⟷ *x* ∈ *convex hull* {*a*,*b*}
unfolding *between-mem-segment segment-convex-hull* **..**

19.43 Shrinking towards the interior of a convex set.

lemma *mem-interior-convex-shrink*:
fixes *s* :: (*real* ^ -) *set*
assumes *convex s c* ∈ *interior s x* ∈ *s* 0 < *e e* ≤ *1*
shows *x* − *e* ∗*R* (*x* − *c*) ∈ *interior s*
proof− **obtain** *d* **where** *d*>*0* **and** *d*:*ball c d* ⊆ *s* **using** *assms*(*2*) **unfolding** *mem-interior* **by** *auto*
show *?thesis* **unfolding** *mem-interior* **apply**(*rule-tac x*=*e*∗*d* **in** *exI*)
apply(*rule*) **defer unfolding** *subset-eq Ball-def mem-ball* **proof**(*rule*,*rule*)
fix *y* **assume** *as*:*dist* (*x* − *e* ∗*R* (*x* − *c*)) *y* < *e* ∗ *d*
have ∗:*y* = (*1* − (*1* − *e*)) ∗*R* ((*1* / *e*) ∗*R* *y* − ((*1* − *e*) / *e*) ∗*R* *x*) + (*1* − *e*) ∗*R* *x* **using** ‹*e*>*0*› **by** (*auto simp add*: *scaleR-left-diff-distrib scaleR-right-diff-distrib*)
have *dist c* ((*1* / *e*) ∗*R* *y* − ((*1* − *e*) / *e*) ∗*R* *x*) = *abs*(*1*/*e*) ∗ *norm* (*e* ∗*R* *c* − *y* + (*1* − *e*) ∗*R* *x*)
unfolding *dist-norm* **unfolding** *norm-scaleR*[*THEN sym*] **apply**(*rule norm-eqI*) **using** ‹*e*>*0*›
by(*auto simp add*: *Cart-eq field-simps*)
also have ... = *abs*(*1*/*e*) ∗ *norm* (*x* − *e* ∗*R* (*x* − *c*) − *y*) **by**(*auto intro*!:*norm-eqI simp add*: *algebra-simps*)
also have ... < *d* **using** *as*[*unfolded dist-norm*] **and** ‹*e*>*0*›
by(*auto simp add*:*pos-divide-less-eq*[*OF* ‹*e*>*0*›] *mult-commute*)
finally show *y* ∈ *s* **apply**(*subst* ∗) **apply**(*rule assms*(*1*)[*unfolded convex-alt*,*rule-format*])
apply(*rule d*[*unfolded subset-eq*,*rule-format*]) **unfolding** *mem-ball* **using** *assms*(*3*−*5*) **by** *auto*
qed(*rule mult-pos-pos*, *insert* ‹*e*>*0*› ‹*d*>*0*›, *auto*) **qed**

lemma *mem-interior-closure-convex-shrink*:
fixes *s* :: (*real* ^ -) *set*
assumes *convex s c* ∈ *interior s x* ∈ *closure s* 0 < *e e* ≤ *1*
shows *x* − *e* ∗*R* (*x* − *c*) ∈ *interior s*
proof− **obtain** *d* **where** *d*>*0* **and** *d*:*ball c d* ⊆ *s* **using** *assms*(*2*) **unfolding** *mem-interior* **by** *auto*
have ∃ *y*∈*s*. *norm* (*y* − *x*) ∗ (*1* − *e*) < *e* ∗ *d* **proof**(*cases x*∈*s*)
case *True* **thus** *?thesis* **using** ‹*e*>*0*› ‹*d*>*0*› **by**(*rule-tac bexI*[**where** *x*=*x*], *auto intro*!: *mult-pos-pos*) **next**
case *False* **hence** *x*:*x islimpt s* **using** *assms*(*3*)[*unfolded closure-def*] **by** *auto*
show *?thesis* **proof**(*cases e*=*1*)
case *True* **obtain** *y* **where** *y*∈*s y* ≠ *x dist y x* < *1*
using *x*[*unfolded islimpt-approachable*,*THEN spec*[**where** *x*=*1*]] **by** *auto*
thus *?thesis* **apply**(*rule-tac x*=*y* **in** *bexI*) **unfolding** *True* **using** ‹*d*>*0*› **by** *auto* **next**
case *False* **hence** *0* < *e* ∗ *d* / (*1* − *e*) **and** ∗:*1* − *e* > *0*

using ‹$e{\leq}1$› ‹$e{>}0$› ‹$d{>}0$› **by**(*auto intro*!:*mult-pos-pos divide-pos-pos*)
then obtain y **where** $y{\in}s$ $y \neq x$ *dist* y $x < e * d$ / $(1 - e)$
using x[*unfolded islimpt-approachable*,*THEN spec*[**where** $x{=}e{*}d$ / $(1 - e)$]]
by *auto*
thus *?thesis* **apply**(*rule-tac* $x{=}y$ **in** *bexI*) **unfolding** *dist-norm* **using** *pos-less-divide-eq*[*OF* $*$] **by** *auto* **qed qed**
then obtain y **where** $y{\in}s$ **and** y:*norm* $(y - x) * (1 - e) < e * d$ **by** *auto*
def $z \equiv c + ((1 - e)$ / $e) *_R (x - y)$
have $*$:$x - e *_R (x - c) = y - e *_R (y - z)$ **unfolding** *z-def* **using** ‹$e{>}0$› **by**
(*auto simp add*: *scaleR-right-diff-distrib scaleR-right-distrib scaleR-left-diff-distrib*)
have $z{\in}$*interior s* **apply**(*rule subset-interior*[*OF d*,*unfolded subset-eq*,*rule-format*])
unfolding *interior-open*[*OF open-ball*] *mem-ball z-def dist-norm* **using** y **and** *assms*(*4*,*5*)
by(*auto simp add*:*field-simps norm-minus-commute*)
thus *?thesis* **unfolding** $*$ **apply** $-$ **apply**(*rule mem-interior-convex-shrink*)
using *assms*(*1*,*4*−*5*) ‹$y{\in}s$› **by** *auto* **qed**

19.44 Some obvious but surprisingly hard simplex lemmas.

lemma *simplex*:
assumes *finite s* $0 \notin s$
shows *convex hull* (*insert* 0 s) = $\{ y. (\exists u. (\forall x{\in}s.\ 0 \leq u\ x) \wedge$ *setsum* u $s \leq 1$ $\wedge$ *setsum* $(\lambda x.\ u\ x *_R x)$ $s = y)\}$
unfolding *convex-hull-finite*[*OF finite.insertI*[*OF assms*(*1*)]] **apply**(*rule set-ext*, *rule*) **unfolding** *mem-Collect-eq*
apply(*erule-tac*[!] *exE*) **apply**(*erule-tac*[!] *conjE*)+ **unfolding** *setsum-clauses*(*2*)[*OF assms*(*1*)]
apply(*rule-tac* $x{=}u$ **in** *exI*) **defer apply**(*rule-tac* $x{=}\lambda x.$ *if* $x = 0$ *then* $1 -$ *setsum* u s *else* $u\ x$ **in** *exI*) **using** *assms*(*2*)
unfolding *if-smult* **and** *setsum-delta-notmem*[*OF assms*(*2*)] **by** *auto*

lemma *std-simplex*:
convex hull (*insert* 0 $\{$ *basis* i | $i.\ i{\in}UNIV\}$) =
$\{x$::*real*^'*n* . $(\forall i.\ 0 \leq x\$i) \wedge$ *setsum* $(\lambda i.\ x\$i)$ *UNIV* $\leq 1\ \}$ (**is** *convex hull* (*insert* 0 *?p*) = *?s*)
proof− **let** *?D* = *UNIV*::'*n set*
have $0 \notin$ *?p* **by**(*auto simp add*: *basis-nonzero*)
have {(*basis* i)::*real*^'*n* |$i.\ i \in$ *?D*} = *basis* ‘ *?D* **by** *auto*
note *sumbas* = *this* *setsum-reindex*[*OF basis-inj*, *unfolded o-def*]
show *?thesis* **unfolding** *simplex*[*OF finite-stdbasis* ‹$0 \notin$ *?p*›] **apply**(*rule set-ext*)
unfolding *mem-Collect-eq* **apply** *rule*
apply(*erule exE*, (*erule conjE*)+) **apply**(*erule-tac*[*2*] *conjE*)+ **proof**−
fix x::*real*^'*n* **and** u **assume** *as*: $\forall x{\in}\{$*basis* i |$i.\ i \in$*?D*$\}.\ 0 \leq u\ x$ *setsum* u $\{$*basis* i |$i.\ i \in$ *?D*$\} \leq 1$ $(\sum x{\in}\{$*basis* i |$i.\ i \in$*?D*$\}.\ u\ x *_R x) = x$
have $*$:$\forall i.\ u$ (*basis* i) = $x\$i$ **using** *as*(*3*) **unfolding** *sumbas* **and** *basis-expansion-unique* [**where** '*a*=*real*, *unfolded smult-conv-scaleR*] **by** *auto*
hence $**$:*setsum* u $\{$*basis* i |$i.\ i \in$ *?D*$\}$ = *setsum* (*op* $\$$ x) *?D* **unfolding** *sumbas* **by**(*rule-tac setsum-cong*, *auto*)
show $(\forall i.\ 0 \leq x\ \$\ i) \wedge$ *setsum* (*op* $\$$ x) *?D* ≤ 1 **apply** $-$ **proof**(*rule*,*rule*)

fix *i*::*'n* **show** *0* ≤ *x*$*i* **unfolding** ∗[*rule-format*,*of i*,*THEN sym*] **apply**(*rule-tac as*(*1*)[*rule-format*]) **by** *auto*
qed(*insert as*(*2*)[*unfolded* ∗∗], *auto*)
next fix *x*::*real^'n* **assume** *as*:∀ *i*. *0* ≤ *x* $ *i setsum* (*op* $ *x*) *?D* ≤ *1*
show ∃ *u*. (∀ *x*∈{*basis i* |*i*. *i* ∈ *?D*}. *0* ≤ *u x*) ∧ *setsum u* {*basis i* |*i*. *i* ∈ *?D*} ≤ *1* ∧ (∑ *x*∈{*basis i* |*i*. *i* ∈ *?D*}. *u x* ∗*R x*) = *x*
apply(*rule-tac x*=λ*y*. *inner y x* **in** *exI*) **apply**(*rule*,*rule*) **unfolding** *mem-Collect-eq* **apply**(*erule exE*) **using** *as*(*1*) **apply**(*erule-tac x*=*i* **in** *allE*)
unfolding *sumbas* **using** *as*(*2*) **and** *basis-expansion-unique* [**where** *'a*=*real*, *unfolded smult-conv-scaleR*] **by**(*auto simp add*:*inner-basis*) **qed qed**

lemma *interior-std-simplex*:
interior (*convex hull* (*insert 0* { *basis i*| *i*. *i*∈*UNIV*})) =
{*x*::*real^'n*. (∀ *i*. *0* < *x*$*i*) ∧ *setsum* (λ*i*. *x*$*i*) *UNIV* < *1* }
apply(*rule set-ext*) **unfolding** *mem-interior std-simplex* **unfolding** *subset-eq mem-Collect-eq Ball-def mem-ball*
unfolding *Ball-def*[*symmetric*] **apply** *rule* **apply**(*erule exE*, (*erule conjE*)+) **defer apply**(*erule conjE*) **proof**−
fix *x*::*real^'n* **and** *e* **assume** *0*<*e* **and** *as*:∀ *xa*. *dist x xa* < *e* ⟶ (∀ *x*. *0* ≤ *xa* $ *x*) ∧ *setsum* (*op* $ *xa*) *UNIV* ≤ *1*
show (∀ *xa*. *0* < *x* $ *xa*) ∧ *setsum* (*op* $ *x*) *UNIV* < *1* **apply**(*rule*,*rule*) **proof**−
fix *i*::*'n* **show** *0* < *x* $ *i* **using** *as*[*THEN spec*[**where** *x*=*x* − (*e* / *2*) ∗*R basis i*]] **and** ‹*e*>*0*›
unfolding *dist-norm* **by**(*auto simp add*: *norm-basis elim*:*allE*[**where** *x*=*i*])
next guess *a* **using** *UNIV-witness*[**where** *'a*=*'n*] **..**
have ∗∗:*dist x* (*x* + (*e* / *2*) ∗*R basis a*) < *e* **using** ‹*e*>*0*› **and** *norm-basis*[*of a*]
unfolding *dist-norm* **by**(*auto intro*!: *mult-strict-left-mono-comm*)
have ⋀*i*. (*x* + (*e* / *2*) ∗*R basis a*) $ *i* = *x*$*i* + (*if i* = *a then e*/*2 else 0*) **by** *auto*
hence ∗:*setsum* (*op* $ (*x* + (*e* / *2*) ∗*R basis a*)) *UNIV* = *setsum* (λ*i*. *x*$*i* + (*if a* = *i then e*/*2 else 0*)) *UNIV* **by**(*rule-tac setsum-cong*, *auto*)
have *setsum* (*op* $ *x*) *UNIV* < *setsum* (*op* $ (*x* + (*e* / *2*) ∗*R basis a*)) *UNIV* **unfolding** ∗ *setsum-addf*
using ‹*0*<*e*› *dimindex-ge-1* **by**(*auto simp add*: *setsum-delta'*)
also have ... ≤ *1* **using** ∗∗ **apply**(*drule-tac as*[*rule-format*]) **by** *auto*
finally show *setsum* (*op* $ *x*) *UNIV* < *1* **by** *auto* **qed**
next
fix *x*::*real^'n* **assume** *as*:∀ *i*. *0* < *x* $ *i setsum* (*op* $ *x*) *UNIV* < *1*
guess *a* **using** *UNIV-witness*[**where** *'a*=*'b*] **..**
let *?d* = (*1* − *setsum* (*op* $ *x*) *UNIV*) / *real* (*CARD*(*'n*))
have *Min* ((*op* $ *x*) ‘ *UNIV*) > *0* **apply**(*rule Min-grI*) **using** *as*(*1*) *dimindex-ge-1* **by** *auto*
moreover have *?d* > *0* **apply**(*rule divide-pos-pos*) **using** *as*(*2*) **using** *dimindex-ge-1* **by**(*auto simp add*: *Suc-le-eq*)
ultimately show ∃ *e*>*0*. ∀ *y*. *dist x y* < *e* ⟶ (∀ *i*. *0* ≤ *y* $ *i*) ∧ *setsum* (*op* $ *y*) *UNIV* ≤ *1*
apply(*rule-tac x*=*min* (*Min* ((*op* $ *x*) ‘ *UNIV*)) *?D* **in** *exI*) **apply** *rule* **defer apply**(*rule*,*rule*) **proof**−

fix *y* **assume** *y*:*dist x y* < *min* (*Min* (*op* $ *x* ‘ *UNIV*)) *?d*
have *setsum* (*op* $ *y*) *UNIV* ≤ *setsum* (λ*i*. *x*$*i* + *?d*) *UNIV* **proof**(*rule setsum-mono*)
fix *i*::′*n* **have** *abs* (*y*$*i* − *x*$*i*) < *?d* **apply**(*rule le-less-trans*) **using** *component-le-norm*[*of y* − *x i*]
using *y*[*unfolded min-less-iff-conj dist-norm*, *THEN conjunct2*] **by**(*auto simp add*: *norm-minus-commute*)
thus *y* $ *i* ≤ *x* $ *i* + *?d* **by** *auto* **qed**
also have ... ≤ *1* **unfolding** *setsum-addf setsum-constant card-enum real-eq-of-nat* **using** *dimindex-ge-1* **by**(*auto simp add*: *Suc-le-eq*)
finally show (∀ *i*. *0* ≤ *y* $ *i*) ∧ *setsum* (*op* $ *y*) *UNIV* ≤ *1* **apply**− **proof**(*rule*,*rule*)
fix *i*::′*n* **have** *norm* (*x* − *y*) < *x*$*i* **using** *y*[*unfolded min-less-iff-conj dist-norm*, *THEN conjunct1*]
by *auto*
thus *0* ≤ *y*$*i* **using** *component-le-norm*[*of x* − *y i*] **and** *as*(*1*)[*rule-format*, *of i*] **by** *auto*
qed *auto* **qed** *auto* **qed**

lemma *interior-std-simplex-nonempty*: **obtains** *a*::*real*^′*n* **where**
a ∈ *interior*(*convex hull* (*insert 0* {*basis i* | *i* . *i* ∈ *UNIV*})) **proof**−
let *?D* = *UNIV*::′*n set* **let** *?a* = *setsum* (λ*b*::*real*^′*n*. *inverse* (*2* ∗ *real CARD*(′*n*)) ∗*R* *b*) {(*basis i*) | *i*. *i* ∈ *?D*}
have ∗:{*basis i* :: *real* ^ ′*n* | *i*. *i* ∈ *?D*} = *basis* ‘ *?D* **by** *auto*
{ fix *i* **have** *?a* $ *i* = *inverse* (*2* ∗ *real CARD*(′*n*))
unfolding *setsum-component vector-smult-component* **and** ∗ **and** *setsum-reindex*[*OF basis-inj*] **and** *o-def*
apply(*rule trans*[*of* - *setsum* (λ*j*. *if i* = *j then inverse* (*2* ∗ *real CARD*(′*n*)) *else 0*) *?D*]) **apply**(*rule setsum-cong2*)
unfolding *setsum-delta*′[*OF finite-UNIV*[**where** ′*a*=′*n*]] **and** *real-dimindex-ge-1*[**where** ′*n*=′*n*] **by**(*auto simp add*: *basis-component*[*of i*]) **}**
note ∗∗ = *this*
show *?thesis* **apply**(*rule that*[*of ?a*]) **unfolding** *interior-std-simplex mem-Collect-eq* **proof**(*rule*,*rule*)
fix *i*::′*n* **show** *0* < *?a* $ *i* **unfolding** ∗∗ **using** *dimindex-ge-1* **by**(*auto simp add*: *Suc-le-eq*) **next**
have *setsum* (*op* $ *?a*) *?D* = *setsum* (λ*i*. *inverse* (*2* ∗ *real CARD*(′*n*))) *?D* **by**(*rule setsum-cong2*, *rule* ∗∗)
also have ... < *1* **unfolding** *setsum-constant card-enum real-eq-of-nat divide-inverse*[*THEN sym*] **by** (*auto simp add*:*field-simps*)
finally show *setsum* (*op* $ *?a*) *?D* < *1* **by** *auto* **qed qed**

end

20 Brouwer-Fixpoint: Results connected with topological dimension.

theory *Brouwer-Fixpoint*
imports *Convex-Euclidean-Space*
begin

lemma *brouwer-compactness-lemma*:
assumes *compact s continuous-on s f* ¬ (∃ *x*∈*s*. (*f x* = (*0*::*real*^'*n*)))
obtains *d* **where** *0* < *d* ∀ *x*∈*s*. *d* ≤ *norm*(*f x*) **proof**(*cases s*={}) **case** *False*
have *continuous-on s* (*norm* ∘ *f*) **by**(*rule continuous-on-intros continuous-on-norm assms*(*2*))+
then obtain *x* **where** *x*:*x*∈*s* ∀ *y*∈*s*. (*norm* ∘ *f*) *x* ≤ (*norm* ∘ *f*) *y*
using *continuous-attains-inf*[*OF assms*(*1*), *of norm* ∘ *f*] **and** *False* **unfolding** *o-def* **by** *auto*
have (*norm* ∘ *f*) *x* > *0* **using** *assms*(*3*) **and** *x*(*1*) **by** *auto*
thus *?thesis* **apply**(*rule that*) **using** *x*(*2*) **unfolding** *o-def* **by** *auto* **qed**(*rule that*[*of 1*], *auto*)

lemma *kuhn-labelling-lemma*:
assumes (∀ *x*::*real*^-. *P x* ⟶ *P* (*f x*)) ∀ *x*. *P x* ⟶ (∀ *i*. *Q i* ⟶ *0* ≤ *x*$*i* ∧ *x*$*i* ≤ *1*)
shows ∃ *l*. (∀ *x i*. *l x i* ≤ (*1*::*nat*)) ∧
(∀ *x i*. *P x* ∧ *Q i* ∧ (*x*$*i* = *0*) ⟶ (*l x i* = *0*)) ∧
(∀ *x i*. *P x* ∧ *Q i* ∧ (*x*$*i* = *1*) ⟶ (*l x i* = *1*)) ∧
(∀ *x i*. *P x* ∧ *Q i* ∧ (*l x i* = *0*) ⟶ *x*$*i* ≤ *f*(*x*)$*i*) ∧
(∀ *x i*. *P x* ∧ *Q i* ∧ (*l x i* = *1*) ⟶ *f*(*x*)$*i* ≤ *x*$*i*) **proof**−
have *and-forall-thm*:⋀*P Q*. (∀ *x*. *P x*) ∧ (∀ *x*. *Q x*) ⟷ (∀ *x*. *P x* ∧ *Q x*) **by** *auto*
have ∗:∀ *x y*::*real*. *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1* ⟶ (*x* ≠ *1* ∧ *x* ≤ *y* ∨ *x* ≠ *0* ∧ *y* ≤ *x*) **by** *auto*
show *?thesis* **unfolding** *and-forall-thm* **apply**(*subst choice-iff*[*THEN sym*])+ **proof**(*rule*,*rule*) **case** *goal1*
let *?R* = λ*y*. (*P x* ∧ *Q xa* ∧ *x* $ *xa* = *0* ⟶ *y* = (*0*::*nat*)) ∧
(*P x* ∧ *Q xa* ∧ *x* $ *xa* = *1* ⟶ *y* = *1*) ∧ (*P x* ∧ *Q xa* ∧ *y* = *0* ⟶ *x* $ *xa* ≤ *f x* $ *xa*) ∧ (*P x* ∧ *Q xa* ∧ *y* = *1* ⟶ *f x* $ *xa* ≤ *x* $ *xa*)
{ **assume** *P x Q xa* **hence** *0* ≤ *f x* $ *xa* ∧ *f x* $ *xa* ≤ *1* **using** *assms*(*2*)[*rule-format*,*of f x xa*]
apply(*drule-tac assms*(*1*)[*rule-format*]) **by** *auto* }
hence *?R 0* ∨ *?R 1* **by** *auto* **thus** *?case* **by** *auto* **qed qed**

20.1 The key "counting" observation, somewhat abstracted.

lemma *setsum-Un-disjoint'*:**assumes** *finite A finite B A* ∩ *B* = {} *A* ∪ *B* = *C*
shows *setsum g C* = *setsum g A* + *setsum g B*
using *setsum-Un-disjoint*[*OF assms*(*1*−*3*)] **and** *assms*(*4*) **by** *auto*

lemma *kuhn-counting-lemma*: **assumes** *finite faces finite simplices*
∀ *f*∈*faces*. *bnd f* ⟶ (*card* {*s* ∈ *simplices*. *face f s*} = *1*)

$\forall f{\in}faces.\ \neg\ bnd\ f\ \longrightarrow\ (card\ \{s\ \in\ simplices.\ face\ f\ s\}\ =\ 2)$
$\forall\ s{\in}simplices.\ compo\ s\ \longrightarrow\ (card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\}\ =\ 1)$
$\forall\ s{\in}simplices.\ \neg\ compo\ s\ \longrightarrow\ (card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\}\ =\ 0)\ \vee$
$(card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\}\ =\ 2)$
$odd(card\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ bnd\ f\})$
shows $odd(card\ \{s\ \in\ simplices.\ compo\ s\})$ **proof**−
have $\bigwedge x.\ \{f{\in}faces.\ compo'\ f\ \wedge\ bnd\ f\ \wedge\ face\ f\ x\}\ \cup\ \{f{\in}faces.\ compo'\ f\ \wedge\ \neg bnd\ f\ \wedge\ face\ f\ x\}\ =\ \{f{\in}faces.\ compo'\ f\ \wedge\ face\ f\ x\}$
$\bigwedge x.\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ bnd\ f\ \wedge\ face\ f\ x\}\ \cap\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ \neg\ bnd\ f\ \wedge\ face\ f\ x\}\ =\ \{\}$ **by** *auto*
hence *lem1*:$setsum\ (\lambda s.\ (card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\}))\ simplices\ =$
$setsum\ (\lambda s.\ card\ \{f\ \in\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ bnd\ f\}.\ face\ f\ s\})\ simplices\ +$
$setsum\ (\lambda s.\ card\ \{f\ \in\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ \neg\ (bnd\ f)\}.\ face\ f\ s\})\ simplices$
unfolding $setsum\text{-}addf[THEN\ sym]$ **apply**− **apply**$(rule\ setsum\text{-}cong2)$
using $assms(1)$ **by**$(auto\ simp\ add{:}\ card\text{-}Un\text{-}Int,\ auto\ simp\ add{:}conj\text{-}commute)$
have *lem2*:$setsum\ (\lambda j.\ card\ \{f\ \in\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ bnd\ f\}.\ face\ f\ j\})\ simplices\ =$
$1\ *\ card\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ bnd\ f\}$
$setsum\ (\lambda j.\ card\ \{f\ \in\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ \neg\ bnd\ f\}.\ face\ f\ j\})\ simplices\ =$
$2\ *\ card\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ \neg\ bnd\ f\}$
apply$(rule\text{-}tac[!]\ setsum\text{-}multicount)$ **using** *assms* **by** *auto*
have *lem3*:$setsum\ (\lambda s.\ card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\})\ simplices\ =$
$setsum\ (\lambda s.\ card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\})\ \{s\ \in\ simplices.\ \ compo\ s\}+$
$setsum\ (\lambda s.\ card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\})\ \{s\ \in\ simplices.\ \neg\ compo\ s\}$
apply$(rule\ setsum\text{-}Un\text{-}disjoint')$ **using** $assms(2)$ **by** *auto*
have *lem4*:$setsum\ (\lambda s.\ card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\})\ \{s\ \in\ simplices.\ compo\ s\}$
$=\ setsum\ (\lambda s.\ 1)\ \{s\ \in\ simplices.\ compo\ s\}$
apply$(rule\ setsum\text{-}cong2)$ **using** $assms(5)$ **by** *auto*
have *lem5*: $setsum\ (\lambda s.\ card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\})\ \{s\ \in\ simplices.\ \neg\ compo\ s\}\ =$
$setsum\ (\lambda s.\ card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\})$
$\{s\ \in\ simplices.\ (\neg\ compo\ s)\ \wedge\ (card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\}\ =\ 0)\}\ +$
$setsum\ (\lambda s.\ card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\})$
$\{s\ \in\ simplices.\ (\neg\ compo\ s)\ \wedge\ (card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\}\ =\ 2)\}$
apply$(rule\ setsum\text{-}Un\text{-}disjoint')$ **using** $assms(2,6)$ **by** *auto*
have $*{:}int\ (\sum s{\in}\{s\ \in\ simplices.\ compo\ s\}.\ card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\})\ =$
$int\ (card\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ bnd\ f\}\ +\ 2\ *\ card\ \{f\ \in\ faces.\ compo'\ f\ \wedge\ \neg\ bnd\ f\})\ -$
$int\ (card\ \{s\ \in\ simplices.\ \neg\ compo\ s\ \wedge\ card\ \{f\ \in\ faces.\ face\ f\ s\ \wedge\ compo'\ f\}\ =\ 2\}\ *\ 2)$
using $lem1[unfolded\ lem3\ lem2\ lem5]$ **by** *auto*
have *even-minus-odd*:$\bigwedge x\ y.\ even\ x\ \Longrightarrow\ odd\ (y{::}int)\ \Longrightarrow\ odd\ (x\ -\ y)$ **using** *assms* **by** *auto*
have *odd-minus-even*:$\bigwedge x\ y.\ odd\ x\ \Longrightarrow\ even\ (y{::}int)\ \Longrightarrow\ odd\ (x\ -\ y)$ **using**

assms **by** *auto*
show *?thesis* **unfolding** *even-nat-def* **unfolding** *card-eq-setsum* **and** *lem4*[*THEN sym*] **and** ∗[*unfolded card-eq-setsum*]
unfolding *card-eq-setsum*[*THEN sym*] **apply** (*rule odd-minus-even*) **unfolding** *zadd-int*[*THEN sym*] **apply**(*rule odd-plus-even*)
apply(*rule assms*(*7*)[*unfolded even-nat-def*]) **unfolding** *int-mult* **by** *auto* **qed**

20.2 The odd/even result for faces of complete vertices, generalized.

lemma *card-1-exists*: *card s = 1* ⟷ (∃!*x*. *x* ∈ *s*) **unfolding** *One-nat-def*
apply *rule* **apply**(*drule card-eq-SucD*) **defer apply**(*erule ex1E*) **proof**−
fix *x* **assume** *as*:*x* ∈ *s* ∀ *y*. *y* ∈ *s* ⟶ *y* = *x*
have ∗:*s* = *insert x* {} **apply**− **apply**(*rule set-ext*,*rule*) **unfolding** *singleton-iff*
apply(*rule as*(*2*)[*rule-format*]) **using** *as*(*1*) **by** *auto*
show *card s* = *Suc 0* **unfolding** ∗ **using** *card-insert* **by** *auto* **qed** *auto*

lemma *card-2-exists*: *card s = 2* ⟷ (∃ *x*∈*s*. ∃ *y*∈*s*. *x* ≠ *y* ∧ (∀ *z*∈*s*. (*z* = *x*) ∨ (*z* = *y*))) **proof**
assume *card s = 2* **then obtain** *x y* **where** *obt*:*s* = {*x*, *y*} *x*≠*y* **unfolding** *numeral-2-eq-2* **apply** − **apply**(*erule exE conjE*|*drule card-eq-SucD*)+ **by** *auto*
show ∃ *x*∈*s*. ∃ *y*∈*s*. *x* ≠ *y* ∧ (∀ *z*∈*s*. *z* = *x* ∨ *z* = *y*) **using** *obt* **by** *auto* **next**
assume ∃ *x*∈*s*. ∃ *y*∈*s*. *x* ≠ *y* ∧ (∀ *z*∈*s*. *z* = *x* ∨ *z* = *y*) **then guess** *x* **.. from** *this*(*2*) **guess** *y* **..**
with ‹*x*∈*s*› **have** ∗:*s* = {*x*, *y*} *x*≠*y* **by** *auto*
from *this*(*2*) **show** *card s* = *2* **unfolding** ∗ **by** *auto* **qed**

lemma *image-lemma-0*: **assumes** *card* {*a*∈*s*. *f* ‘ (*s* − {*a*}) = *t* − {*b*}} = *n*
shows *card* {*s*′. ∃ *a*∈*s*. (*s*′ = *s* − {*a*}) ∧ (*f* ‘ *s*′ = *t* − {*b*})} = *n* **proof**−
have ∗:{*s*′. ∃ *a*∈*s*. (*s*′ = *s* − {*a*}) ∧ (*f* ‘ *s*′ = *t* − {*b*})} = (λ*a*. *s* − {*a*}) ‘ {*a*∈*s*. *f* ‘ (*s* − {*a*}) = *t* − {*b*}} **by** *auto*
show *?thesis* **unfolding** ∗ **unfolding** *assms*[*THEN sym*] **apply**(*rule card-image*) **unfolding** *inj-on-def*
apply(*rule*,*rule*,*rule*) **unfolding** *mem-Collect-eq* **by** *auto* **qed**

lemma *image-lemma-1*: **assumes** *finite s finite t card s = card t f* ‘ *s = t b* ∈ *t*
shows *card* {*s*′. ∃ *a*∈*s*. *s*′ = *s* − {*a*} ∧ *f* ‘ *s*′ = *t* − {*b*}} = *1* **proof**−
obtain *a* **where** *a*:*b* = *f a a*∈*s* **using** *assms*(*4*−*5*) **by** *auto*
have *inj*:*inj-on f s* **apply**(*rule eq-card-imp-inj-on*) **using** *assms*(*1*−*4*) **by** *auto*
have ∗:{*a* ∈ *s*. *f* ‘ (*s* − {*a*}) = *t* − {*b*}} = {*a*} **apply**(*rule set-ext*) **unfolding** *singleton-iff*
apply(*rule*,*rule inj*[*unfolded inj-on-def*,*rule-format*]) **unfolding** *a* **using** *a*(*2*) **and** *assms* **and** *inj*[*unfolded inj-on-def*] **by** *auto*
show *?thesis* **apply**(*rule image-lemma-0*) **unfolding** ∗ **by** *auto* **qed**

lemma *image-lemma-2*: **assumes** *finite s finite t card s = card t f* ‘ *s* ⊆ *t f* ‘ *s* ≠ *t b* ∈ *t*
shows (*card* {*s*′. ∃ *a*∈*s*. (*s*′ = *s* − {*a*}) ∧ *f* ‘ *s*′ = *t* − {*b*}} = *0*) ∨
(*card* {*s*′. ∃ *a*∈*s*. (*s*′ = *s* − {*a*}) ∧ *f* ‘ *s*′ = *t* − {*b*}} = *2*) **proof**(*cases*

$\{a{\in}s.\ f\ `\ (s - \{a\}) = t - \{b\}\} = \{\})$
case *True* **thus** *?thesis* **apply**−**apply**(*rule disjI1*, *rule image-lemma-0*) **using** *assms*(*1*) **by** *auto*
next let *?M* = $\{a{\in}s.\ f\ `\ (s - \{a\}) = t - \{b\}\}$
case *False* **then obtain** *a* **where** *a*∈*?M* **by** *auto* **hence** *a*:*a*∈*s f* ` (*s* − {*a*}) = *t* − {*b*} **by** *auto*
have *f a* ∈ *t* − {*b*} **using** *a* **and** *assms* **by** *auto*
hence ∃ *c* ∈ *s* − {*a*}. *f a* = *f c* **unfolding** *image-iff*[*symmetric*] **and** *a* **by** *auto*
then obtain *c* **where** *c*:*c* ∈ *s a* ≠ *c f a* = *f c* **by** *auto*
hence ∗:*f* ` (*s* − {*c*}) = *f* ` (*s* − {*a*}) **apply**−**apply**(*rule set-ext*,*rule*) **proof**−
fix *x* **assume** *x* ∈ *f* ` (*s* − {*a*}) **then obtain** *y* **where** *y*:*f y* = *x y*∈*s*− {*a*} **by** *auto*
thus *x* ∈ *f* ` (*s* − {*c*}) **unfolding** *image-iff* **apply**(*rule-tac x*=*if y* = *c then a else y* **in** *bexI*) **using** *c a* **by** *auto* **qed** *auto*
have *c*∈*?M* **unfolding** *mem-Collect-eq* **and** ∗ **using** *a* **and** *c*(*1*) **by** *auto*
show *?thesis* **apply**(*rule disjI2*, *rule image-lemma-0*) **unfolding** *card-2-exists*
apply(*rule bexI*[*OF* - ‹*a*∈*?M*›], *rule bexI*[*OF* - ‹*c*∈*?M*›],*rule*,*rule* ‹*a*≠*c*›) **proof**(*rule*,*unfold mem-Collect-eq*,*erule conjE*)
fix *z* **assume** *as*:*z* ∈ *s f* ` (*s* − {*z*}) = *t* − {*b*}
have *inj*:*inj-on f* (*s* − {*z*}) **apply**(*rule eq-card-imp-inj-on*) **unfolding** *as* **using** *as*(*1*) **and** *assms* **by** *auto*
show *z* = *a* ∨ *z* = *c* **proof**(*rule ccontr*)
assume ¬ (*z* = *a* ∨ *z* = *c*) **thus** *False* **using** *inj*[*unfolded inj-on-def*,*THEN bspec*[**where** *x*=*a*],*THEN bspec*[**where** *x*=*c*]]
using ‹*a*∈*s*› ‹*c*∈*s*› ‹*f a* = *f c*› ‹*a*≠*c*› **by** *auto* **qed qed qed**

20.3 Combine this with the basic counting lemma.

lemma *kuhn-complete-lemma*:
assumes *finite simplices*
∀ *f s*. *face f s* ⟷ (∃ *a*∈*s*. *f* = *s* − {*a*}) ∀ *s*∈*simplices*. *card s* = *n* + *2* ∀ *s*∈*simplices*. (*rl* ` *s*) ⊆ {*0*..*n*+*1*}
∀ *f*∈ {*f*. ∃ *s*∈*simplices*. *face f s*}. *bnd f* ⟶ (*card* {*s*∈*simplices*. *face f s*} = *1*)
∀ *f*∈ {*f*. ∃ *s*∈*simplices*. *face f s*}. ¬*bnd f* ⟶ (*card* {*s*∈*simplices*. *face f s*} = *2*)
odd(*card* {*f*∈{*f*. ∃ *s*∈*simplices*. *face f s*}. *rl* ` *f* = {*0*..*n*} ∧ *bnd f*})
shows *odd* (*card* {*s*∈*simplices*. (*rl* ` *s* = {*0*..*n*+*1*})})
apply(*rule kuhn-counting-lemma*) **defer apply**(*rule assms*)+ **prefer** *3* **apply**(*rule assms*) **proof**(*rule-tac*[*1*−*2*] *ballI impI*)+
have ∗:{*f*. ∃ *s*∈*simplices*. ∃ *a*∈*s*. *f* = *s* − {*a*}} = (⋃ *s*∈*simplices*. {*f*. ∃ *a*∈*s*. (*f* = *s* − {*a*})}) **by** *auto*
have ∗∗: ∀ *s*∈*simplices*. *card s* = *n* + *2* ∧ *finite s* **using** *assms*(*3*) **by** (*auto intro*: *card-ge-0-finite*)
show *finite* {*f*. ∃ *s*∈*simplices*. *face f s*} **unfolding** *assms*(*2*)[*rule-format*] **and** ∗
apply(*rule finite-UN-I*[*OF assms*(*1*)]) **using** ∗∗ **by** *auto*
have ∗:⋀ *P f s*. *s*∈*simplices* ⟹ (*f* ∈ {*f*. ∃ *s*∈*simplices*. ∃ *a*∈*s*. *f* = *s* − {*a*}}) ∧ (∃ *a*∈*s*. (*f* = *s* − {*a*})) ∧ *P f* ⟷ (∃ *a*∈*s*. (*f* = *s* − {*a*}) ∧ *P f*) **by** *auto*
fix *s* **assume** *s*:*s*∈*simplices* **let** *?S* = {*f* ∈ {*f*. ∃ *s*∈*simplices*. *face f s*}. *face f s* ∧ *rl* ` *f* = {*0*..*n*}}
have {*0*..*n* + *1*} − {*n* + *1*} = {*0*..*n*} **by** *auto*

hence $S{:}?S = \{s'.\ \exists\, a{\in}s.\ s' = s - \{a\} \wedge rl\ `\ s' = \{0..n+1\} - \{n+1\}\}$ **apply**− **apply**(*rule set-ext*)
unfolding *assms*(*2*)[*rule-format*] *mem-Collect-eq* **and** ∗[*OF s*, *unfolded mem-Collect-eq*, **where** $P{=}\lambda x.\ rl\ `\ x = \{0..n\}$] **by** *auto*
show $rl\ `\ s = \{0..n{+}1\} \Longrightarrow card\ ?S = 1\ rl\ `\ s \neq \{0..n{+}1\} \Longrightarrow card\ ?S = 0 \vee card\ ?S = 2$ **unfolding** *S*
apply(*rule-tac*[!] *image-lemma-1 image-lemma-2*) **using** ∗∗ *assms*(*4*) **and** *s* **by** *auto* **qed**

20.4 We use the following notion of ordering rather than pointwise indexing.

definition $kle\ n\ x\ y \longleftrightarrow (\exists\, k{\subseteq}\{1..n{::}nat\}.\ (\forall j.\ y(j) = x(j) + (if\ j \in k\ then\ (1{::}nat)\ else\ 0)))$

lemma *kle-refl*[*intro*]: *kle n x x* **unfolding** *kle-def* **by** *auto*

lemma *kle-antisym*: $kle\ n\ x\ y \wedge kle\ n\ y\ x \longleftrightarrow (x = y)$
unfolding *kle-def* **apply** *rule* **apply**(*rule ext*) **by** *auto*

lemma *pointwise-minimal-pointwise-maximal*: **fixes** *s*::(*nat*⇒*nat*) *set*
assumes *finite s* $s \neq \{\}$ $\forall\, x{\in}s.\ \forall\, y{\in}s.\ (\forall j.\ x\ j \leq y\ j) \vee (\forall j.\ y\ j \leq x\ j)$
shows $\exists\, a{\in}s.\ \forall\, x{\in}s.\ \forall j.\ a\ j \leq x\ j\ \exists\, a{\in}s.\ \forall\, x{\in}s.\ \forall j.\ x\ j \leq a\ j$
using *assms* **unfolding** *atomize-conj* **apply**− **proof**(*induct s rule*:*finite-induct*)
fix *x* **and** *F*::(*nat*⇒*nat*) *set* **assume** *as*:*finite F x* $\notin F$
$[\![F \neq \{\};\ \forall\, x{\in}F.\ \forall\, y{\in}F.\ (\forall j.\ x\ j \leq y\ j) \vee (\forall j.\ y\ j \leq x\ j)]\!]$
$\Longrightarrow (\exists\, a{\in}F.\ \forall\, x{\in}F.\ \forall j.\ a\ j \leq x\ j) \wedge (\exists\, a{\in}F.\ \forall\, x{\in}F.\ \forall j.\ x\ j \leq a\ j)$ *insert x F* $\neq \{\}$
$\forall\, xa{\in}insert\ x\ F.\ \forall\, y{\in}insert\ x\ F.\ (\forall j.\ xa\ j \leq y\ j) \vee (\forall j.\ y\ j \leq xa\ j)$
show $(\exists\, a{\in}insert\ x\ F.\ \forall\, x{\in}insert\ x\ F.\ \forall j.\ a\ j \leq x\ j) \wedge (\exists\, a{\in}insert\ x\ F.\ \forall\, x{\in}insert\ x\ F.\ \forall j.\ x\ j \leq a\ j)$ **proof**(*cases F*={})
case *True* **thus** *?thesis* **apply**−**apply**(*rule*,*rule-tac*[!] *x*=*x* **in** *bexI*) **by** *auto*
next
case *False* **obtain** *a b* **where** *a*:$a{\in}insert\ x\ F\ \forall\, x{\in}F.\ \forall j.\ a\ j \leq x\ j$ **and**
b:$b{\in}insert\ x\ F\ \forall\, x{\in}F.\ \forall j.\ x\ j \leq b\ j$ **using** *as*(*3*)[*OF False*] **using** *as*(*5*) **by** *auto*
have $\exists\, a{\in}insert\ x\ F.\ \forall\, x{\in}insert\ x\ F.\ \forall j.\ a\ j \leq x\ j$
using *as*(*5*)[*rule-format*,*OF a*(*1*) *insertI1*] **apply**− **proof**(*erule disjE*)
assume $\forall j.\ a\ j \leq x\ j$ **thus** *?thesis* **apply**(*rule-tac x*=*a* **in** *bexI*) **using** *a* **by** *auto* **next**
assume $\forall j.\ x\ j \leq a\ j$ **thus** *?thesis* **apply**(*rule-tac x*=*x* **in** *bexI*) **apply**(*rule*,*rule*) **using** *a* **apply** −
apply(*erule-tac x*=*xa* **in** *ballE*) **apply**(*erule-tac x*=*j* **in** *allE*)+ **by** *auto*
qed moreover
have $\exists\, b{\in}insert\ x\ F.\ \forall\, x{\in}insert\ x\ F.\ \forall j.\ x\ j \leq b\ j$
using *as*(*5*)[*rule-format*,*OF b*(*1*) *insertI1*] **apply**− **proof**(*erule disjE*)
assume $\forall j.\ x\ j \leq b\ j$ **thus** *?thesis* **apply**(*rule-tac x*=*b* **in** *bexI*) **using** *b* **by** *auto* **next**
assume $\forall j.\ b\ j \leq x\ j$ **thus** *?thesis* **apply**(*rule-tac x*=*x* **in** *bexI*) **ap-**

ply(*rule*,*rule*) **using** *b* **apply** −
apply(*erule-tac x=xa* **in** *ballE*) **apply**(*erule-tac x=j* **in** *allE*)+ **by** *auto*
qed
ultimately show *?thesis* **by** *auto* **qed qed**(*auto*)

lemma *kle-imp-pointwise*: *kle n x y* ⟹ (∀ *j*. *x j* ≤ *y j*) **unfolding** *kle-def* **by** *auto*

lemma *pointwise-antisym*: **fixes** *x*::*nat* ⇒ *nat*
shows (∀ *j*. *x j* ≤ *y j*) ∧ (∀ *j*. *y j* ≤ *x j*) ⟷ (*x* = *y*)
apply(*rule*, *rule ext*,*erule conjE*) **apply**(*erule-tac x=xa* **in** *allE*)+ **by** *auto*

lemma *kle-trans*: **assumes** *kle n x y kle n y z kle n x z* ∨ *kle n z x* **shows** *kle n x z*
using *assms* **apply**− **apply**(*erule disjE*) **apply** *assumption* **proof**− **case** *goal1*
hence *x=z* **apply**− **apply**(*rule ext*) **apply**(*drule kle-imp-pointwise*)+
apply(*erule-tac x=xa* **in** *allE*)+ **by** *auto* **thus** *?case* **by** *auto* **qed**

lemma *kle-strict*: **assumes** *kle n x y x* ≠ *y*
shows ∀ *j*. *x j* ≤ *y j* ∃ *k*. 1 ≤ *k* ∧ *k* ≤ *n* ∧ *x*(*k*) < *y*(*k*)
apply(*rule kle-imp-pointwise*[*OF assms*(*1*)]) **proof**−
guess *k* **using** *assms*(*1*)[*unfolded kle-def*] **.. note** *k* = *this*
show ∃ *k*. 1 ≤ *k* ∧ *k* ≤ *n* ∧ *x*(*k*) < *y*(*k*) **proof**(*cases k={}*)
case *True* **hence** *x=y* **apply**−**apply**(*rule ext*) **using** *k* **by** *auto*
thus *?thesis* **using** *assms*(*2*) **by** *auto* **next**
case *False* **hence** (*SOME k'*. *k'* ∈ *k*) ∈ *k* **apply**−**apply**(*rule someI-ex*) **by** *auto*
thus *?thesis* **apply**(*rule-tac x=SOME k'*. *k'* ∈ *k* **in** *exI*) **using** *k* **by** *auto* **qed**
qed

lemma *kle-minimal*: **assumes** *finite s s* ≠ {} ∀ *x*∈*s*. ∀ *y*∈*s*. *kle n x y* ∨ *kle n y x*
shows ∃ *a*∈*s*. ∀ *x*∈*s*. *kle n a x* **proof**−
have ∃ *a*∈*s*. ∀ *x*∈*s*. ∀ *j*. *a j* ≤ *x j* **apply**(*rule pointwise-minimal-pointwise-maximal*(*1*)[*OF assms*(*1*−*2*)])
apply(*rule*,*rule*) **apply**(*drule-tac assms*(*3*)[*rule-format*],*assumption*) **using** *kle-imp-pointwise* **by** *auto*
then guess *a* **.. note** *a=this* **show** *?thesis* **apply**(*rule-tac x=a* **in** *bexI*) **proof**
fix *x* **assume** *x*∈*s*
show *kle n a x* **using** *assms*(*3*)[*rule-format*,*OF a*(*1*) ‹*x*∈*s*›] **apply**− **proof**(*erule disjE*)
assume *kle n x a* **hence** *x* = *a* **apply**− **unfolding** *pointwise-antisym*[*THEN sym*]
apply(*drule kle-imp-pointwise*) **using** *a*(*2*)[*rule-format*,*OF* ‹*x*∈*s*›] **by** *auto*
thus *?thesis* **using** *kle-refl* **by** *auto* **qed qed**(*insert a*, *auto*) **qed**

lemma *kle-maximal*: **assumes** *finite s s* ≠ {} ∀ *x*∈*s*. ∀ *y*∈*s*. *kle n x y* ∨ *kle n y x*
shows ∃ *a*∈*s*. ∀ *x*∈*s*. *kle n x a* **proof**−
have ∃ *a*∈*s*. ∀ *x*∈*s*. ∀ *j*. *a j* ≥ *x j* **apply**(*rule pointwise-minimal-pointwise-maximal*(*2*)[*OF assms*(*1*−*2*)])

apply(*rule*,*rule*) **apply**(*drule-tac assms*(*3*)[*rule-format*],*assumption*) **using** *kle-imp-pointwise* **by** *auto*
then guess *a* **.. note** *a*=*this* **show** *?thesis* **apply**(*rule-tac x*=*a* **in** *bexI*) **proof**
fix *x* **assume** *x*∈*s*
show *kle n x a* **using** *assms*(*3*)[*rule-format*,*OF a*(*1*) ‹*x*∈*s*›] **apply**− **proof**(*erule disjE*)
assume *kle n a x* **hence** *x* = *a* **apply**− **unfolding** *pointwise-antisym*[*THEN sym*]
apply(*drule kle-imp-pointwise*) **using** *a*(*2*)[*rule-format*,*OF* ‹*x*∈*s*›] **by** *auto*
thus *?thesis* **using** *kle-refl* **by** *auto* **qed qed**(*insert a*, *auto*) **qed**

lemma *kle-strict-set*: **assumes** *kle n x y x* ≠ *y*
shows *1* ≤ *card* {*k*∈{*1*..*n*}. *x k* < *y k*} **proof**−
guess *i* **using** *kle-strict*(*2*)[*OF assms*] **..**
hence *card* {*i*} ≤ *card* {*k*∈{*1*..*n*}. *x k* < *y k*} **apply**− **apply**(*rule card-mono*) **by** *auto*
thus *?thesis* **by** *auto* **qed**

lemma *kle-range-combine*:
assumes *kle n x y kle n y z kle n x z* ∨ *kle n z x*
m1 ≤ *card* {*k*∈{*1*..*n*}. *x k* < *y k*}
m2 ≤ *card* {*k*∈{*1*..*n*}. *y k* < *z k*}
shows *kle n x z* ∧ *m1* + *m2* ≤ *card* {*k*∈{*1*..*n*}. *x k* < *z k*}
apply(*rule*,*rule kle-trans*[*OF assms*(*1*−*3*)]) **proof**−
have ⋀*j*. *x j* < *y j* ⟹ *x j* < *z j* **apply**(*rule less-le-trans*) **using** *kle-imp-pointwise*[*OF assms*(*2*)] **by** *auto* **moreover**
have ⋀*j*. *y j* < *z j* ⟹ *x j* < *z j* **apply**(*rule le-less-trans*) **using** *kle-imp-pointwise*[*OF assms*(*1*)] **by** *auto* **ultimately**
have ∗:{*k*∈{*1*..*n*}. *x k* < *y k*} ∪ {*k*∈{*1*..*n*}. *y k* < *z k*} = {*k*∈{*1*..*n*}. *x k* < *z k*} **by** *auto*
have ∗∗:{*k* ∈ {*1*..*n*}. *x k* < *y k*} ∩ {*k* ∈ {*1*..*n*}. *y k* < *z k*} = {} **unfolding** *disjoint-iff-not-equal*
apply(*rule*,*rule*,*unfold mem-Collect-eq*,*rule ccontr*) **apply**(*erule conjE*)+ **proof**−
fix *i j* **assume** *as*:*i* ∈ {*1*..*n*} *x i* < *y i j* ∈ {*1*..*n*} *y j* < *z j* ¬ *i* ≠ *j*
guess *kx* **using** *assms*(*1*)[*unfolded kle-def*] **.. note** *kx*=*this*
have *x i* < *y i* **using** *as* **by** *auto* **hence** *i* ∈ *kx* **using** *as*(*1*) *kx* **apply**(*rule-tac ccontr*) **by** *auto*
hence *x i* + *1* = *y i* **using** *kx* **by** *auto* **moreover**
guess *ky* **using** *assms*(*2*)[*unfolded kle-def*] **.. note** *ky*=*this*
have *y i* < *z i* **using** *as* **by** *auto* **hence** *i* ∈ *ky* **using** *as*(*1*) *ky* **apply**(*rule-tac ccontr*) **by** *auto*
hence *y i* + *1* = *z i* **using** *ky* **by** *auto* **ultimately**
have *z i* = *x i* + *2* **by** *auto*
thus *False* **using** *assms*(*3*) **unfolding** *kle-def* **by**(*auto simp add*: *split-if-eq1*) **qed**
have *fin*:⋀*P*. *finite* {*x*∈{*1*..*n*::*nat*}. *P x*} **by** *auto*
have *m1* + *m2* ≤ *card* {*k*∈{*1*..*n*}. *x k* < *y k*} + *card* {*k*∈{*1*..*n*}. *y k* < *z k*} **using** *assms*(*4*−*5*) **by** *auto*
also have ... ≤ *card* {*k*∈{*1*..*n*}. *x k* < *z k*} **unfolding** *card-Un-Int*[*OF fin fin*]

unfolding $*$ $**$ **by** *auto*
finally show *m1 + m2* $\leq$ *card* $\{k \in \{1..n\}.\ x\ k < z\ k\}$ **by** *auto* **qed**

lemma *kle-range-combine-l*:
assumes *kle n x y kle n y z kle n x z* $\vee$ *kle n z x m* $\leq$ *card* $\{k \in \{1..n\}.\ y(k) < z(k)\}$
shows *kle n x z* $\wedge$ *m* $\leq$ *card* $\{k \in \{1..n\}.\ x(k) < z(k)\}$
using *kle-range-combine*[*OF assms*(*1*−*3*) - *assms*(*4*), *of 0*] **by** *auto*

lemma *kle-range-combine-r*:
assumes *kle n x y kle n y z kle n x z* $\vee$ *kle n z x m* $\leq$ *card* $\{k \in \{1..n\}.\ x\ k < y\ k\}$
shows *kle n x z* $\wedge$ *m* $\leq$ *card* $\{k \in \{1..n\}.\ x(k) < z(k)\}$
using *kle-range-combine*[*OF assms*(*1*−*3*) *assms*(*4*), *of 0*] **by** *auto*

lemma *kle-range-induct*:
assumes *card s* = *Suc m* $\forall x \in s.\ \forall y \in s.$ *kle n x y* $\vee$ *kle n y x*
shows $\exists x \in s.\ \exists y \in s.$ *kle n x y* $\wedge$ *m* $\leq$ *card* $\{k \in \{1..n\}.\ x\ k < y\ k\}$ **proof**−
have *finite s s*$\neq\{\}$ **using** *assms*(*1*) **by** (*auto intro*: *card-ge-0-finite*)
thus *?thesis* **using** *assms* **apply**− **proof**(*induct m arbitrary*: *s*)
case *0* **thus** *?case* **using** *kle-refl* **by** *auto* **next**
case (*Suc m*) **then obtain** *a* **where** *a*:*a*$\in$*s* $\forall x \in s.$ *kle n a x* **using** *kle-minimal*[*of s n*] **by** *auto*
show *?case* **proof**(*cases s* $\subseteq \{a\}$) **case** *False*
hence *card* $(s - \{a\})$ = *Suc m s* − $\{a\}$ $\neq \{\}$ **using** *card-Diff-singleton*[*OF - a*(*1*)] *Suc*(*4*) ‹*finite s*› **by** *auto*
then obtain *x b* **where** *xb*:*x*$\in$*s* − $\{a\}$ *b*$\in$*s* − $\{a\}$ *kle n x b m* $\leq$ *card* $\{k \in \{1..n\}.\ x\ k < b\ k\}$
using *Suc*(*1*)[*of s* − $\{a\}$] **using** *Suc*(*5*) ‹*finite s*› **by** *auto*
have *1* $\leq$ *card* $\{k \in \{1..n\}.\ a\ k < x\ k\}$ *m* $\leq$ *card* $\{k \in \{1..n\}.\ x\ k < b\ k\}$
apply(*rule kle-strict-set*) **apply**(*rule a*(*2*)[*rule-format*]) **using** *a* **and** *xb* **by** *auto*
thus *?thesis* **apply**(*rule-tac x*=*a* **in** *bexI*, *rule-tac x*=*b* **in** *bexI*)
using *kle-range-combine*[*OF a*(*2*)[*rule-format*] *xb*(*3*) *Suc*(*5*)[*rule-format*], *of 1 m*] **using** *a*(*1*) *xb*(*1*−*2*) **by** *auto* **next**
case *True* **hence** *s* = $\{a\}$ **using** *Suc*(*3*) **by** *auto* **hence** *card s* = *1* **by** *auto*
hence *False* **using** *Suc*(*4*) ‹*finite s*› **by** *auto* **thus** *?thesis* **by** *auto* **qed qed qed**

lemma *kle-Suc*: *kle n x y* $\Longrightarrow$ *kle* $(n + 1)$ *x y*
unfolding *kle-def* **apply**(*erule exE*) **apply**(*rule-tac x*=*k* **in** *exI*) **by** *auto*

lemma *kle-trans-1*: **assumes** *kle n x y* **shows** *x j* $\leq$ *y j y j* $\leq$ *x j* + *1*
using *assms*[*unfolded kle-def*] **by** *auto*

lemma *kle-trans-2*: **assumes** *kle n a b kle n b c* $\forall j.$ *c j* $\leq$ *a j* + *1* **shows** *kle n a c* **proof**−
guess *kk1* **using** *assms*(*1*)[*unfolded kle-def*] **.. note** *kk1*=*this*
guess *kk2* **using** *assms*(*2*)[*unfolded kle-def*] **.. note** *kk2*=*this*

show *?thesis* **unfolding** *kle-def* **apply**(*rule-tac x=kk1 ∪ kk2* **in** *exI*) **apply**(*rule*) **defer proof**
fix *i* **show** *c i = a i + (if i ∈ kk1 ∪ kk2 then 1 else 0)* **proof**(*cases i∈kk1 ∪ kk2*)
case *True* **hence** *c i ≥ a i + (if i ∈ kk1 ∪ kk2 then 1 else 0)*
unfolding *kk1*[*THEN conjunct2*,*rule-format*,*of i*] *kk2*[*THEN conjunct2*,*rule-format*,*of i*] **by** *auto*
moreover have *c i ≤ a i + (if i ∈ kk1 ∪ kk2 then 1 else 0)* **using** *True assms(3)* **by** *auto*
ultimately show *?thesis* **by** *auto* **next**
case *False* **thus** *?thesis* **using** *kk1 kk2* **by** *auto* **qed qed**(*insert kk1 kk2*, *auto*) **qed**

lemma *kle-between-r*: **assumes** *kle n a b kle n b c kle n a x kle n c x* **shows** *kle n b x*
apply(*rule kle-trans-2*[*OF assms*(*2*,*4*)]) **proof have** ∗:⋀*c b x*::*nat. x ≤ c + 1 ⟹ c ≤ b ⟹ x ≤ b + 1* **by** *auto*
fix *j* **show** *x j ≤ b j + 1* **apply**(*rule* ∗)**using** *kle-trans-1*[*OF assms*(*1*),*of j*] *kle-trans-1*[*OF assms*(*3*), *of j*] **by** *auto* **qed**

lemma *kle-between-l*: **assumes** *kle n a b kle n b c kle n x a kle n x c* **shows** *kle n x b*
apply(*rule kle-trans-2*[*OF assms*(*3*,*1*)]) **proof have** ∗:⋀*c b x*::*nat. c ≤ x + 1 ⟹ b ≤ c ⟹ b ≤ x + 1* **by** *auto*
fix *j* **show** *b j ≤ x j + 1* **apply**(*rule* ∗) **using** *kle-trans-1*[*OF assms*(*2*),*of j*] *kle-trans-1*[*OF assms*(*4*), *of j*] **by** *auto* **qed**

lemma *kle-adjacent*:
assumes ∀ *j*. *b j = (if j = k then a*(*j*) *+ 1 else a j) kle n a x kle n x b*
shows *(x = a) ∨ (x = b)* **proof**(*cases x k = a k*)
case *True* **show** *?thesis* **apply**(*rule disjI1*,*rule ext*) **proof**− **fix** *j*
have *x j ≤ a j* **using** *kle-imp-pointwise*[*OF assms*(*3*),*THEN spec*[**where** *x=j*]]
unfolding *assms*(*1*)[*rule-format*] **apply**−**apply**(*cases j=k*) **using** *True* **by** *auto*
thus *x j = a j* **using** *kle-imp-pointwise*[*OF assms*(*2*),*THEN spec*[**where** *x=j*]] **by** *auto* **qed next**
case *False* **show** *?thesis* **apply**(*rule disjI2*,*rule ext*) **proof**− **fix** *j*
have *x j ≥ b j* **using** *kle-imp-pointwise*[*OF assms*(*2*),*THEN spec*[**where** *x=j*]]
unfolding *assms*(*1*)[*rule-format*] **apply**−**apply**(*cases j=k*) **using** *False* **by** *auto*
thus *x j = b j* **using** *kle-imp-pointwise*[*OF assms*(*3*),*THEN spec*[**where** *x=j*]] **by** *auto* **qed qed**

20.5 kuhn's notion of a simplex (a reformulation to avoid so much indexing).

definition *ksimplex p n* (*s*::(*nat ⇒ nat*) *set*) ⟷
(*card s = n + 1 ∧*

$(\forall x{\in}s.\ \forall j.\ x(j) \leq p) \wedge$
$(\forall x{\in}s.\ \forall j.\ j{\notin}\{1..n\} \longrightarrow (x\ j = p)) \wedge$
$(\forall x{\in}s.\ \forall y{\in}s.\ kle\ n\ x\ y \vee kle\ n\ y\ x))$

lemma *ksimplexI*:*card s = n + 1* $\Longrightarrow$ $\forall x{\in}s.\ \forall j.\ x\ j \leq p \Longrightarrow \forall x{\in}s.\ \forall j.\ j \notin \{1..n\} \longrightarrow x\ j = p \Longrightarrow \forall x{\in}s.\ \forall y{\in}s.\ kle\ n\ x\ y \vee kle\ n\ y\ x \Longrightarrow ksimplex\ p\ n\ s$
unfolding *ksimplex-def* **by** *auto*

lemma *ksimplex-eq*: *ksimplex p n* (*s*::(*nat* ⇒ *nat*) *set*) ⟷
(card s = n + 1 ∧ finite s ∧
(∀ x∈s. ∀ j. x(j) ≤ p) ∧
(∀ x∈s. ∀ j. j∉{1..n} ⟶ (x j = p)) ∧
(∀ x∈s. ∀ y∈s. kle n x y ∨ kle n y x))
unfolding *ksimplex-def* **by** (*auto intro*: *card-ge-0-finite*)

lemma *ksimplex-extrema*: **assumes** *ksimplex p n s* **obtains** *a b* **where** *a ∈ s b ∈ s*
∀ x∈s. kle n a x ∧ kle n x b ∀ i. b(i) = (if i ∈ {1..n} then a(i) + 1 else a(i))
proof(*cases n=0*)
case *True* **obtain** *x* **where** ∗:*s = {x}* **using** *assms*[*unfolded ksimplex-eq True*,*THEN conjunct1*]
unfolding *add-0-left card-1-exists* **by** *auto*
show *?thesis* **apply**(*rule that*[*of x x*]) **unfolding** ∗ *True* **by** *auto* **next**
note *assm = assms*[*unfolded ksimplex-eq*]
case *False* **have** *s≠{}* **using** *assm* **by** *auto*
obtain *a* **where** *a*:*a∈s ∀ x∈s. kle n a x* **using** ‹*s≠{}*› *assm* **using** *kle-minimal*[*of s n*] **by** *auto*
obtain *b* **where** *b*:*b∈s ∀ x∈s. kle n x b* **using** ‹*s≠{}*› *assm* **using** *kle-maximal*[*of s n*] **by** *auto*
obtain *c d* **where** *c-d*:*c∈s d∈s kle n c d n ≤ card {k ∈ {1..n}. c k < d k}*
using *kle-range-induct*[*of s n n*] **using** *assm* **by** *auto*
have *kle n c b ∧ n ≤ card {k ∈ {1..n}. c k < b k}* **apply**(*rule kle-range-combine-r*[**where** *y=d*]) **using** *c-d a b* **by** *auto*
hence *kle n a b ∧ n ≤ card {k∈{1..n}. a(k) < b(k)}* **apply−apply**(*rule kle-range-combine-l*[**where** *y=c*]) **using** *a* ‹*c∈s*› ‹*b∈s*› **by** *auto*
moreover have *card {1..n} ≥ card {k∈{1..n}. a(k) < b(k)}* **apply**(*rule card-mono*) **by** *auto*
ultimately have ∗:*{k∈{1 .. n}. a k < b k} = {1..n}* **apply−** **apply**(*rule card-subset-eq*) **by** *auto*
show *?thesis* **apply**(*rule that*[*OF a(1) b(1)*]) **defer apply**(*subst* ∗[*THEN sym*]) **unfolding** *mem-Collect-eq* **proof**
guess *k* **using** *a*(*2*)[*rule-format*,*OF b(1)*,*unfolded kle-def*] **.. note** *k=this*
fix *i* **show** *b i = (if i ∈ {1..n} ∧ a i < b i then a i + 1 else a i)* **proof**(*cases i ∈ {1..n}*)
case *True* **thus** *?thesis* **unfolding** *k*[*THEN conjunct2*,*rule-format*] **by** *auto* **next**
case *False* **have** *a i = p* **using** *assm* **and** *False* ‹*a∈s*› **by** *auto*
moreover have *b i = p* **using** *assm* **and** *False* ‹*b∈s*› **by** *auto*
ultimately show *?thesis* **by** *auto* **qed qed**(*insert a(2) b(2) assm*,*auto*) **qed**

lemma *ksimplex-extrema-strong*:
assumes *ksimplex p n s* $n \neq 0$ **obtains** *a b* **where** $a \in s$ $b \in s$ $a \neq b$
$\forall x{\in}s.\ kle\ n\ a\ x \wedge kle\ n\ x\ b\ \forall i.\ b(i) = (if\ i \in \{1..n\}\ then\ a(i) + 1\ else\ a(i))$
proof−
obtain *a b* **where** *ab*:$a \in s$ $b \in s$ $\forall x{\in}s.\ kle\ n\ a\ x \wedge kle\ n\ x\ b\ \forall i.\ b(i) = (if\ i \in \{1..n\}\ then\ a(i) + 1\ else\ a(i))$
apply(*rule ksimplex-extrema*[*OF assms*(*1*)]) **by** *auto*
have $a \neq b$ **apply**(*rule ccontr*) **unfolding** *not-not* **apply**(*drule cong*[*of - - 1 1*]) **using** *ab*(*4*) *assms*(*2*) **by** *auto*
thus *?thesis* **apply**(*rule-tac that*[*of a b*]) **using** *ab* **by** *auto* **qed**

lemma *ksimplexD*:
assumes *ksimplex p n s*
shows *card s* $= n + 1$ *finite s card s* $= n + 1$ $\forall x{\in}s.\ \forall j.\ x\ j \leq p$ $\forall x{\in}s.\ \forall j.\ j \notin \{1..n\} \longrightarrow x\ j = p$
$\forall x{\in}s.\ \forall y{\in}s.\ kle\ n\ x\ y \vee kle\ n\ y\ x$ **using** *assms* **unfolding** *ksimplex-eq* **by** *auto*

lemma *ksimplex-successor*:
assumes *ksimplex p n s a* $\in s$
shows $(\forall x{\in}s.\ kle\ n\ x\ a) \vee (\exists y{\in}s.\ \exists k{\in}\{1..n\}.\ \forall j.\ y(j) = (if\ j = k\ then\ a(j) + 1\ else\ a(j)))$
proof(*cases* $\forall x{\in}s.\ kle\ n\ x\ a$) **case** *True* **thus** *?thesis* **by** *auto* **next note** *assm* = *ksimplexD*[*OF assms*(*1*)]
case *False* **then obtain** *b* **where** *b*:$b{\in}s$ $\neg\ kle\ n\ b\ a$ $\forall x{\in}\{x \in s.\ \neg\ kle\ n\ x\ a\}.\ kle\ n\ b\ x$
using *kle-minimal*[*of* $\{x{\in}s.\ \neg\ kle\ n\ x\ a\}$ *n*] **and** *assm* **by** *auto*
hence ∗∗:$1 \leq card\ \{k{\in}\{1..n\}.\ a\ k < b\ k\}$ **apply**− **apply**(*rule kle-strict-set*) **using** *assm*(*6*) **and** ‹*a*∈*s*› **by**(*auto simp add*:*kle-refl*)

let *?kle1* $= \{x \in s.\ \neg\ kle\ n\ x\ a\}$ **have** *card ?kle1* > 0 **apply**(*rule ccontr*) **using** *assm*(*2*) **and** *False* **by** *auto*
hence *sizekle1*: *card ?kle1* $=$ *Suc* (*card ?kle1* $- 1$) **using** *assm*(*2*) **by** *auto*
obtain *c d* **where** *c-d*: $c \in s$ $\neg\ kle\ n\ c\ a$ $d \in s$ $\neg\ kle\ n\ d\ a$ *kle n c d card ?kle1* $- 1 \leq card\ \{k \in \{1..n\}.\ c\ k < d\ k\}$
using *kle-range-induct*[*OF sizekle1*, *of n*] **using** *assm* **by** *auto*

let *?kle2* $= \{x \in s.\ kle\ n\ x\ a\}$
have *card ?kle2* > 0 **apply**(*rule ccontr*) **using** *assm*(*6*)[*rule-format*,*of a a*] **and** ‹*a*∈*s*› **and** *assm*(*2*) **by** *auto*
hence *sizekle2*:*card ?kle2* $=$ *Suc* (*card ?kle2* $- 1$) **using** *assm*(*2*) **by** *auto*
obtain *e f* **where** *e-f*: $e \in s$ *kle n e a* $f \in s$ *kle n f a kle n e f card ?kle2* $- 1 \leq card\ \{k \in \{1..n\}.\ e\ k < f\ k\}$
using *kle-range-induct*[*OF sizekle2*, *of n*] **using** *assm* **by** *auto*

have *card* $\{k{\in}\{1..n\}.\ a\ k < b\ k\} = 1$ **proof**(*rule ccontr*) **case** *goal1*
hence *as*:*card* $\{k{\in}\{1..n\}.\ a\ k < b\ k\} \geq 2$ **using** ∗∗ **by** *auto*
have ∗:*finite ?kle2 finite ?kle1 ?kle2* $\cup$ *?kle1* $= s$ *?kle2* $\cap$ *?kle1* $= \{\}$ **using** *assm*(*2*) **by** *auto*

have (*card ?kle2 − 1*) + *2* + (*card ?kle1 − 1*) = *card ?kle2* + *card ?kle1* **using** *sizekle1 sizekle2* **by** *auto*
also have ... = *n* + *1* **unfolding** *card-Un-Int*[*OF* ∗(*1*−*2*)] ∗(*3*−) **using** *assm*(*3*) **by** *auto*
finally have *n*:(*card ?kle2 − 1*) + (*2* + (*card ?kle1 − 1*)) = *n* + *1* **by** *auto*
have *kle n e a* ∧ *card* {*x* ∈ *s*. *kle n x a*} − *1* ≤ *card* {*k* ∈ {*1*..*n*}. *e k* < *a k*}
apply(*rule kle-range-combine-r*[**where** *y*=*f*]) **using** *e-f* **using** ‹*a*∈*s*› *assm*(*6*) **by** *auto*
moreover have *kle n b d* ∧ *card* {*x* ∈ *s*. ¬ *kle n x a*} − *1* ≤ *card* {*k* ∈ {*1*..*n*}. *b k* < *d k*}
apply(*rule kle-range-combine-l*[**where** *y*=*c*]) **using** *c-d* **using** *assm*(*6*) **and** *b* **by** *auto*
hence *kle n a d* ∧ *2* + (*card* {*x* ∈ *s*. ¬ *kle n x a*} − *1*) ≤ *card* {*k* ∈ {*1*..*n*}. *a k* < *d k*} **apply**−
apply(*rule kle-range-combine*[**where** *y*=*b*]) **using** *as* **and** *b assm*(*6*) ‹*a*∈*s*› ‹*d*∈*s*› **apply**− **by** *blast*+
ultimately have *kle n e d* ∧ (*card ?kle2 − 1*) + (*2* + (*card ?kle1 − 1*)) ≤ *card* {*k*∈{*1*..*n*}. *e k* < *d k*} **apply**−
apply(*rule kle-range-combine*[**where** *y*=*a*]) **using** *assm*(*6*)[*rule-format*,*OF* ‹*e*∈*s*› ‹*d*∈*s*›] **apply** − **by** *blast*+
moreover have *card* {*k* ∈ {*1*..*n*}. *e k* < *d k*} ≤ *card* {*1*..*n*} **apply**(*rule card-mono*) **by** *auto*
ultimately show *False* **unfolding** *n* **by** *auto* **qed**
then guess *k* **unfolding** *card-1-exists* **..** **note** *k*=*this*[*unfolded mem-Collect-eq*]

show *?thesis* **apply**(*rule disjI2*) **apply**(*rule-tac x*=*b* **in** *bexI*,*rule-tac x*=*k* **in** *bexI*) **proof**
fix *j*::*nat* **have** *kle n a b* **using** *b* **and** *assm*(*6*)[*rule-format*, *OF* ‹*a*∈*s*› ‹*b*∈*s*›] **by** *auto*
then guess *kk* **unfolding** *kle-def* **..** **note** *kk-raw*=*this* **note** *kk*=*this*[*THEN conjunct2*,*rule-format*]
have *kkk*:*k*∈*kk* **apply**(*rule ccontr*) **using** *k*(*1*) **unfolding** *kk* **by** *auto*
show *b j* = (*if j* = *k then a j* + *1 else a j*) **proof**(*cases j*∈*kk*)
case *True* **hence** *j*=*k* **apply**−**apply**(*rule k*(*2*)[*rule-format*]) **using** *kk-raw kkk* **by** *auto*
thus *?thesis* **unfolding** *kk* **using** *kkk* **by** *auto* **next**
case *False* **hence** *j*≠*k* **using** *k*(*2*)[*rule-format*, *of j k*] **using** *kk-raw kkk* **by** *auto*
thus *?thesis* **unfolding** *kk* **using** *kkk* **using** *False* **by** *auto* **qed** **qed**(*insert k*(*1*) ‹*b*∈*s*›, *auto*) **qed**

lemma *ksimplex-predecessor*:
assumes *ksimplex p n s a* ∈ *s*
shows (∀ *x*∈*s*. *kle n a x*) ∨ (∃ *y*∈*s*. ∃ *k*∈{*1*..*n*}. ∀ *j*. *a*(*j*) = (*if j* = *k then y*(*j*) + *1 else y*(*j*)))
proof(*cases* ∀ *x*∈*s*. *kle n a x*) **case** *True* **thus** *?thesis* **by** *auto* **next** **note** *assm* = *ksimplexD*[*OF assms*(*1*)]
case *False* **then obtain** *b* **where** *b*:*b*∈*s* ¬ *kle n a b* ∀ *x*∈{*x* ∈ *s*. ¬ *kle n a x*}. *kle n x b*

using *kle-maximal*[*of* {*x*∈*s*. ¬ *kle n a x*} *n*] **and** *assm* **by** *auto*
hence ∗∗:*1* ≤ *card* {*k*∈{*1*..*n*}. *a k* > *b k*} **apply**− **apply**(*rule kle-strict-set*) **using** *assm*(*6*) **and** ‹*a*∈*s*› **by**(*auto simp add*:*kle-refl*)

let *?kle1* = {*x* ∈ *s*. ¬ *kle n a x*} **have** *card ?kle1* > *0* **apply**(*rule ccontr*) **using** *assm*(*2*) **and** *False* **by** *auto*
hence *sizekle1*:*card ?kle1* = *Suc* (*card ?kle1* − *1*) **using** *assm*(*2*) **by** *auto*
obtain *c d* **where** *c-d*: *c* ∈ *s* ¬ *kle n a c d* ∈ *s* ¬ *kle n a d kle n d c card ?kle1* − *1* ≤ *card* {*k* ∈ {*1*..*n*}. *c k* > *d k*}
using *kle-range-induct*[*OF sizekle1*, *of n*] **using** *assm* **by** *auto*

let *?kle2* = {*x* ∈ *s*. *kle n a x*}
have *card ?kle2* > *0* **apply**(*rule ccontr*) **using** *assm*(*6*)[*rule-format*,*of a a*] **and** ‹*a*∈*s*› **and** *assm*(*2*) **by** *auto*
hence *sizekle2*:*card ?kle2* = *Suc* (*card ?kle2* − *1*) **using** *assm*(*2*) **by** *auto*
obtain *e f* **where** *e-f*: *e* ∈ *s kle n a e f* ∈ *s kle n a f kle n f e card ?kle2* − *1* ≤ *card* {*k* ∈ {*1*..*n*}. *e k* > *f k*}
using *kle-range-induct*[*OF sizekle2*, *of n*] **using** *assm* **by** *auto*

have *card* {*k*∈{*1*..*n*}. *a k* > *b k*} = *1* **proof**(*rule ccontr*) **case** *goal1*
hence *as*:*card* {*k*∈{*1*..*n*}. *a k* > *b k*} ≥ *2* **using** ∗∗ **by** *auto*
have ∗:*finite ?kle2 finite ?kle1 ?kle2* ∪ *?kle1* = *s ?kle2* ∩ *?kle1* = {} **using** *assm*(*2*) **by** *auto*
have (*card ?kle2* − *1*) + *2* + (*card ?kle1* − *1*) = *card ?kle2* + *card ?kle1* **using** *sizekle1 sizekle2* **by** *auto*
also have ... = *n* + *1* **unfolding** *card-Un-Int*[*OF* ∗(*1*−*2*)] ∗(*3*−) **using** *assm*(*3*) **by** *auto*
finally have *n*:(*card ?kle1* − *1*) + *2* + (*card ?kle2* − *1*) = *n* + *1* **by** *auto*
have *kle n a e* ∧ *card* {*x* ∈ *s*. *kle n a x*} − *1* ≤ *card* {*k* ∈ {*1*..*n*}. *e k* > *a k*}
apply(*rule kle-range-combine-l*[**where** *y*=*f*]) **using** *e-f* **using** ‹*a*∈*s*› *assm*(*6*) **by** *auto*
moreover have *kle n d b* ∧ *card* {*x* ∈ *s*. ¬ *kle n a x*} − *1* ≤ *card* {*k* ∈ {*1*..*n*}. *b k* > *d k*}
apply(*rule kle-range-combine-r*[**where** *y*=*c*]) **using** *c-d* **using** *assm*(*6*) **and** *b* **by** *auto*
hence *kle n d a* ∧ (*card* {*x* ∈ *s*. ¬ *kle n a x*} − *1*) + *2* ≤ *card* {*k* ∈ {*1*..*n*}. *a k* > *d k*} **apply**−
apply(*rule kle-range-combine*[**where** *y*=*b*]) **using** *as* **and** *b assm*(*6*) ‹*a*∈*s*› ‹*d*∈*s*› **by** *blast*+
ultimately have *kle n d e* ∧ (*card ?kle1* − *1* + *2*) + (*card ?kle2* − *1*) ≤ *card* {*k*∈{*1*..*n*}. *e k* > *d k*} **apply**−
apply(*rule kle-range-combine*[**where** *y*=*a*]) **using** *assm*(*6*)[*rule-format*,*OF* ‹*e*∈*s*› ‹*d*∈*s*›] **apply** − **by** *blast*+
moreover have *card* {*k* ∈ {*1*..*n*}. *e k* > *d k*} ≤ *card* {*1*..*n*} **apply**(*rule card-mono*) **by** *auto*
ultimately show *False* **unfolding** *n* **by** *auto* **qed**
then guess *k* **unfolding** *card-1-exists* **..** **note** *k*=*this*[*unfolded mem-Collect-eq*]

show *?thesis* **apply**(*rule disjI2*) **apply**(*rule-tac x*=*b* **in** *bexI*,*rule-tac x*=*k* **in**

bexI) **proof**
fix *j*::*nat* **have** *kle n b a* **using** *b* **and** *assm*(*6*)[*rule-format*, *OF* ‹*a*∈*s*› ‹*b*∈*s*›]
by *auto*
then guess *kk* **unfolding** *kle-def* **..** **note** *kk-raw*=*this* **note** *kk*=*this*[*THEN conjunct2*,*rule-format*]
have *kkk*:*k*∈*kk* **apply**(*rule ccontr*) **using** *k*(*1*) **unfolding** *kk* **by** *auto*
show *a j* = (*if j* = *k then b j* + *1 else b j*) **proof**(*cases j*∈*kk*)
case *True* **hence** *j*=*k* **apply**−**apply**(*rule k*(*2*)[*rule-format*]) **using** *kk-raw kkk* **by** *auto*
thus *?thesis* **unfolding** *kk* **using** *kkk* **by** *auto* **next**
case *False* **hence** *j*≠*k* **using** *k*(*2*)[*rule-format*, *of j k*] **using** *kk-raw kkk* **by** *auto*
thus *?thesis* **unfolding** *kk* **using** *kkk* **using** *False* **by** *auto* **qed** **qed**(*insert k*(*1*) ‹*b*∈*s*›, *auto*) **qed**

20.6 The lemmas about simplices that we need.

lemma *card-funspace'*: **assumes** *finite s finite t card s* = *m card t* = *n*
shows *card* {*f*. (∀ *x*∈*s*. *f x* ∈ *t*) ∧ (∀ *x*∈*UNIV* − *s*. *f x* = *d*)} = *n* ^ *m* (**is** *card* (*?M s*) = -)
using *assms* **apply** − **proof**(*induct m arbitrary*: *s*)
have ∗:{*f*. ∀ *x*. *f x* = *d*} = {λ*x*. *d*} **apply**(*rule set-ext*,*rule*)**unfolding** *mem-Collect-eq* **apply**(*rule*,*rule ext*) **by** *auto*
case *0* **thus** *?case* **by**(*auto simp add*: ∗) **next**
case (*Suc m*) **guess** *a* **using** *card-eq-SucD*[*OF Suc*(*4*)] **..** **then guess** *s0*
apply(*erule-tac exE*) **apply**(*erule conjE*)+ **.** **note** *as0* = *this*
have ∗∗:*card s0* = *m* **using** *as0* **using** *Suc*(*2*) *Suc*(*4*) **by** *auto*
let *?l* = (λ(*b*,*g*) *x*. *if x* = *a then b else g x*) **have** ∗:*?M* (*insert a s0*) = *?l* ‘ {(*b*,*g*). *b*∈*t* ∧ *g*∈*?M s0*}
apply(*rule set-ext*,*rule*) **unfolding** *mem-Collect-eq image-iff* **apply**(*erule conjE*)
apply(*rule-tac x*=(*x a*, λ*y*. *if y*∈*s0 then x y else d*) **in** *bexI*) **apply**(*rule ext*)
prefer *3* **apply** *rule* **defer**
apply(*erule bexE*,*rule*) **unfolding** *mem-Collect-eq* **apply**(*erule splitE*)+ **apply**(*erule conjE*)+ **proof**−
fix *x xa xb xc y* **assume** *as*:*x* = (λ(*b*, *g*) *x*. *if x* = *a then b else g x*) *xa xb* ∈ *UNIV* − *insert a s0 xa* = (*xc*, *y*) *xc* ∈ *t*
∀ *x*∈*s0*. *y x* ∈ *t* ∀ *x*∈*UNIV* − *s0*. *y x* = *d* **thus** *x xb* = *d* **unfolding** *as* **by** *auto* **qed** *auto*
have *inj*:*inj-on ?l* {(*b*,*g*). *b*∈*t* ∧ *g*∈*?M s0*} **unfolding** *inj-on-def* **apply**(*rule*,*rule*,*rule*) **unfolding** *mem-Collect-eq* **apply**(*erule splitE conjE*)+ **proof**−
case *goal1* **note** *as* = *this*(*1*,*4*−)[*unfolded goal1 split-conv*]
have *xa* = *xb* **using** *as*(*1*)[*THEN cong*[*of - - a*]] **by** *auto*
moreover have *ya* = *yb* **proof**(*rule ext*) **fix** *x* **show** *ya x* = *yb x* **proof**(*cases x* = *a*)
case *False* **thus** *?thesis* **using** *as*(*1*)[*THEN cong*[*of - - x x*]] **by** *auto* **next**
case *True* **thus** *?thesis* **using** *as*(*5*,*7*) **using** *as0*(*2*) **by** *auto* **qed** **qed**
ultimately show *?case* **unfolding** *goal1* **by** *auto* **qed**
have *finite s0* **using** ‹*finite s*› **unfolding** *as0* **by** *simp*
show *?case* **unfolding** *as0* ∗ *card-image*[*OF inj*] **using** *assms*

unfolding *SetCompr-Sigma-eq* **apply**−
unfolding *card-cartesian-product*
using *Suc(1)*[*OF* ‹*finite s0*› ‹*finite t*› ∗∗ ‹*card t = n*›] **by** *auto*
qed

lemma *card-funspace*: **assumes** *finite s finite t*
shows *card* {*f*. (∀ *x*∈*s*. *f x* ∈ *t*) ∧ (∀ *x*∈*UNIV* − *s*. *f x* = *d*)} = (*card t*) ^ (*card s*)
using *assms* **by** (*auto intro*: *card-funspace'*)

lemma *finite-funspace*: **assumes** *finite s finite t*
shows *finite* {*f*. (∀ *x*∈*s*. *f x* ∈ *t*) ∧ (∀ *x*∈*UNIV* − *s*. *f x* = *d*)} (**is** *finite ?S*)
proof (*cases card t* > *0*)
case *True*
have *card ?S* = (*card t*) ^ (*card s*)
using *assms* **by** (*auto intro*!: *card-funspace*)
thus *?thesis* **using** *True* **by** (*auto intro*: *card-ge-0-finite*)
next
case *False* **hence** *t* = {} **using** ‹*finite t*› **by** *auto*
show *?thesis*
proof (*cases s* = {})
have ∗:{*f*. ∀ *x*. *f x* = *d*} = {λ*x*. *d*} **by** (*auto intro*: *ext*)
case *True* **thus** *?thesis* **using** ‹*t* = {}› **by** (*auto simp*: ∗)
next
case *False* **thus** *?thesis* **using** ‹*t* = {}› **by** *simp*
qed
qed

lemma *finite-simplices*: *finite* {*s*. *ksimplex p n s*}
apply(*rule finite-subset*[*of* - {*s*. *s*⊆{*f*. (∀ *i*∈{*1*..*n*}. *f i* ∈ {*0*..*p*}) ∧ (∀ *i*∈*UNIV*−{*1*..*n*}. *f i* = *p*)}}])
unfolding *ksimplex-def* **defer apply**(*rule finite-Collect-subsets*) **apply**(*rule finite-funspace*)
by *auto*

lemma *simplex-top-face*: **assumes** *0*<*p* ∀ *x*∈*f*. *x* (*n* + *1*) = *p*
shows (∃ *s a*. *ksimplex p* (*n* + *1*) *s* ∧ *a* ∈ *s* ∧ (*f* = *s* − {*a*})) ⟷ *ksimplex p n f* (**is** *?ls* = *?rs*) **proof**
assume *?ls* **then guess** *s* **..** **then guess** *a* **apply**−**apply**(*erule exE*,(*erule conjE*)+) **. note** *sa*=*this*
show *?rs* **unfolding** *ksimplex-def sa*(*3*) **apply**(*rule*) **defer apply** *rule* **defer apply**(*rule*,*rule*,*rule*,*rule*) **defer apply**(*rule*,*rule*) **proof**−
fix *x y* **assume** *as*:*x* ∈*s* − {*a*} *y* ∈*s* − {*a*} **have** *xyp*:*x* (*n* + *1*) = *y* (*n* + *1*)
using *as*(*1*)[*unfolded sa*(*3*)[*THEN sym*], *THEN assms*(*2*)[*rule-format*]]
using *as*(*2*)[*unfolded sa*(*3*)[*THEN sym*], *THEN assms*(*2*)[*rule-format*]] **by** *auto*
show *kle n x y* ∨ *kle n y x* **proof**(*cases kle* (*n* + *1*) *x y*)
case *True* **then guess** *k* **unfolding** *kle-def* **..** **note** *k*=*this* **hence** ∗:*n*+*1* ∉ *k* **using** *xyp* **by** *auto*
have ¬ (∃ *x*∈*k*. *x*∉{*1*..*n*}) **apply**(*rule ccontr*) **unfolding** *not-not* **apply**(*erule*

bexE) **proof**−
fix *x* **assume** *as*:$x \in k\ x \notin \{1..n\}$ **have** $x \neq n{+}1$ **using** *as* **and** $*$ **by** *auto*
thus *False* **using** *as* **and** *k*[*THEN conjunct1*,*unfolded subset-eq*] **by** *auto*
qed
thus *?thesis* **apply**−**apply**(*rule disjI1*) **unfolding** *kle-def* **using** *k* **apply**(*rule-tac x*=*k* **in** *exI*) **by** *auto* **next**
case *False* **hence** *kle* $(n + 1)\ y\ x$ **using** *ksimplexD*(*6*)[*OF sa*(*1*),*rule-format*, *of x y*] **using** *as* **by** *auto*
then guess *k* **unfolding** *kle-def* **.. note** *k*=*this* **hence** $*$:$n{+}1 \notin k$ **using** *xyp* **by** *auto*
hence ¬ $(\exists x{\in}k.\ x{\notin}\{1..n\})$ **apply**−**apply**(*rule ccontr*) **unfolding** *not-not* **apply**(*erule bexE*) **proof**−
fix *x* **assume** *as*:$x \in k\ x \notin \{1..n\}$ **have** $x \neq n{+}1$ **using** *as* **and** $*$ **by** *auto*
thus *False* **using** *as* **and** *k*[*THEN conjunct1*,*unfolded subset-eq*] **by** *auto*
qed
thus *?thesis* **apply**−**apply**(*rule disjI2*) **unfolding** *kle-def* **using** *k* **apply**(*rule-tac x*=*k* **in** *exI*) **by** *auto* **qed next**
fix *x j* **assume** *as*:$x{\in}s - \{a\}\ j{\notin}\{1..n\}$
thus $x\,j = p$ **using** *as*(*1*)[*unfolded sa*(*3*)[*THEN sym*], *THEN assms*(*2*)[*rule-format*]]
apply(*cases* $j = n{+}1$) **using** *sa*(*1*)[*unfolded ksimplex-def*] **by** *auto* **qed**(*insert sa ksimplexD*[*OF sa*(*1*)], *auto*) **next**
assume *?rs* **note** *rs*=*ksimplexD*[*OF this*] **guess** *a b* **apply**(*rule ksimplex-extrema*[*OF* ‹*?rs*›]) **. note** *ab* = *this*
def $c \equiv \lambda i.$ *if* $i = (n + 1)$ *then* $p - 1$ *else a i*
have *c*∉*f* **apply**(*rule ccontr*) **unfolding** *not-not* **apply**(*drule assms*(*2*)[*rule-format*]) **unfolding** *c-def* **using** *assms*(*1*) **by** *auto*
thus *?ls* **apply**(*rule-tac x*=*insert c f* **in** *exI*,*rule-tac x*=*c* **in** *exI*) **unfolding** *ksimplex-def conj-assoc*
apply(*rule conjI*) **defer apply**(*rule conjI*) **defer apply**(*rule conjI*) **defer apply**(*rule conjI*) **defer**
proof(*rule-tac*[*3*−*5*] *ballI allI*)+
fix *x j* **assume** *x*:$x \in$ *insert c f* **thus** $x\ j \leq p$ **proof** (*cases x*=*c*)
case *True* **show** *?thesis* **unfolding** *True c-def* **apply**(*cases j*=*n*+*1*) **using** *ab*(*1*) **and** *rs*(*4*) **by** *auto*
qed(*insert x rs*(*4*), *auto simp add*:*c-def*)
show $j \notin \{1..n + 1\} \longrightarrow x\ j = p$ **apply**(*cases x*=*c*) **using** *x ab*(*1*) *rs*(*5*) **unfolding** *c-def* **by** *auto*
{ fix *z* **assume** *z*:$z \in$ *insert c f* **hence** *kle* $(n + 1)\ c\ z$ **apply**(*cases* $z = c$) **proof**−
case *False* **hence** *z*∈*f* **using** *z* **by** *auto*
then guess *k* **apply**(*drule-tac ab*(*3*)[*THEN bspec*[**where** *x*=*z*], *THEN conjunct1*]) **unfolding** *kle-def* **apply**(*erule exE*) **.**
thus *kle* $(n + 1)\ c\ z$ **unfolding** *kle-def* **apply**(*rule-tac x*=*insert* $(n + 1)$ *k* **in** *exI*) **unfolding** *c-def*
using *ab* **using** *rs*(*5*)[*rule-format*,*OF ab*(*1*),*of* $n + 1$] *assms*(*1*) **by** *auto* **qed** *auto* **} note** $* = this$
fix *y* **assume** *y*:$y \in$ *insert c f* **show** *kle* $(n + 1)\ x\ y \vee$ *kle* $(n + 1)\ y\ x$ **proof**(*cases* $x = c \vee y = c$)
case *False* **hence** $**$:$x{\in}f\ y{\in}f$ **using** *x y* **by** *auto*

show *?thesis* **using** *rs(6)*[*rule-format*,*OF* **]* **by**(*auto dest*: *kle-Suc*) **qed**(*insert* * *x y*, *auto*)
qed(*insert rs*, *auto*) **qed**

lemma *ksimplex-fix-plane*:
assumes $a \in s$ $j \in \{1..n::nat\}$ $\forall x \in s - \{a\}.\ x\ j = q\ a0 \in s\ a1 \in s$
$\forall i.\ a1\ i = ((if\ i \in \{1..n\}\ then\ a0\ i + 1\ else\ a0\ i)::nat)$
shows $(a = a0) \vee (a = a1)$ **proof**− **have** *:$\bigwedge P\ A\ x\ y.\ \forall x \in A.\ P\ x \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow P\ x \wedge P\ y$ **by** *auto*
show *?thesis* **apply**(*rule ccontr*) **using** *[*OF assms(3), of a0 a1*] **unfolding** *assms(6)*[*THEN spec*[**where** *x=j*]]
using *assms(1−2,4−5)* **by** *auto* **qed**

lemma *ksimplex-fix-plane-0*:
assumes $a \in s$ $j \in \{1..n::nat\}$ $\forall x \in s - \{a\}.\ x\ j = 0\ a0 \in s\ a1 \in s$
$\forall i.\ a1\ i = ((if\ i \in \{1..n\}\ then\ a0\ i + 1\ else\ a0\ i)::nat)$
shows $a = a1$ **apply**(*rule ccontr*) **using** *ksimplex-fix-plane*[*OF assms*]
using *assms(3)*[*THEN bspec*[**where** *x=a1*]] **using** *assms(2,5)*
unfolding *assms(6)*[*THEN spec*[**where** *x=j*]] **by** *simp*

lemma *ksimplex-fix-plane-p*:
assumes *ksimplex p n s* $a \in s$ $j \in \{1..n\}$ $\forall x \in s - \{a\}.\ x\ j = p\ a0 \in s\ a1 \in s$
$\forall i.\ a1\ i = (if\ i \in \{1..n\}\ then\ a0\ i + 1\ else\ a0\ i)$
shows $a = a0$ **proof**(*rule ccontr*) **note** *s = ksimplexD*[*OF assms(1),rule-format*]
assume *as*:$a \neq a0$ **hence** *:$a0 \in s - \{a\}$ **using** *assms(5)* **by** *auto*
hence $a1 = a$ **using** *ksimplex-fix-plane*[*OF assms(2−)*] **by** *auto*
thus *False* **using** *as* **using** *assms(3,5)* **and** *assms(7)*[*rule-format*,*of j*]
unfolding *assms(4)*[*rule-format*,*OF* *] **using** *s(4)*[*OF assms(6), of j*] **by** *auto*
qed

lemma *ksimplex-replace-0*:
assumes *ksimplex p n s* $a \in s$ $n \neq 0$ $j \in \{1..n\}$ $\forall x \in s - \{a\}.\ x\ j = 0$
shows $card\ \{s'.\ ksimplex\ p\ n\ s' \wedge (\exists b \in s'.\ s' - \{b\} = s - \{a\})\} = 1$ **proof**−
have *:$\bigwedge s'\ a\ a'.\ s' - \{a'\} = s - \{a\} \Longrightarrow a' = a \Longrightarrow a' \in s' \Longrightarrow a \in s \Longrightarrow (s' = s)$ **by** *auto*
have **:$\bigwedge s'\ a'.\ ksimplex\ p\ n\ s' \Longrightarrow a' \in s' \Longrightarrow s' - \{a'\} = s - \{a\} \Longrightarrow s' = s$ **proof**− **case** *goal1*
guess *a0 a1* **apply**(*rule ksimplex-extrema-strong*[*OF assms(1,3)*]) **. note** *exta = this*[*rule-format*]
have *a*:$a = a1$ **apply**(*rule ksimplex-fix-plane-0*[*OF assms(2,4−5)*]) **using** *exta(1−2,5)* **by** *auto* **moreover**
guess *b0 b1* **apply**(*rule ksimplex-extrema-strong*[*OF goal1(1) assms(3)*]) **. note** *extb = this*[*rule-format*]
have *a'*:$a' = b1$ **apply**(*rule ksimplex-fix-plane-0*[*OF goal1(2) assms(4), of b0*]) **unfolding** *goal1(3)* **using** *assms extb goal1* **by** *auto* **moreover**
have $b0 = a0$ **unfolding** *kle-antisym*[*THEN sym, of b0 a0 n*] **using** *exta extb* **using** *goal1(3)* **unfolding** *a a'* **by** *blast*
hence $b1 = a1$ **apply**−**apply**(*rule ext*) **unfolding** *exta(5) extb(5)* **by** *auto* **ultimately**

show $s' = s$ **apply**−**apply**(*rule* ∗[*of* - *a1 b1*]) **using** *exta*(*1*−*2*) *extb*(*1*−*2*) *goal1* **by** *auto* **qed**
show *?thesis* **unfolding** *card-1-exists* **apply**−**apply**(*rule ex1I*[*of* - *s*])
unfolding *mem-Collect-eq* **defer apply**(*erule conjE bexE*)+ **apply**(*rule-tac* a'=*b* **in** ∗∗) **using** *assms*(*1*,*2*) **by** *auto* **qed**

lemma *ksimplex-replace-1*:
assumes *ksimplex p n s a* ∈ *s n* ≠ *0 j*∈{*1*..*n*} ∀ *x*∈*s* − {*a*}. *x j* = *p*
shows *card* {s'. *ksimplex p n s'* ∧ (∃ *b*∈s'. s' − {*b*} = *s* − {*a*})} = *1* **proof**−
have *lem*:⋀*a a' s'*. s' − {a'} = *s* − {*a*} ⟹ a' = *a* ⟹ a' ∈ s' ⟹ *a* ∈ *s* ⟹ s' = *s* **by** *auto*
have *lem*:⋀s' a'. *ksimplex p n s'* ⟹ a'∈s' ⟹ s' − {a'} = *s* − {*a*} ⟹ s' = *s* **proof**− **case** *goal1*
guess *a0 a1* **apply**(*rule ksimplex-extrema-strong*[*OF assms*(*1*,*3*)]) **. note** *exta* = *this*[*rule-format*]
have *a*:*a* = *a0* **apply**(*rule ksimplex-fix-plane-p*[*OF assms*(*1*−*2*,*4*−*5*) *exta*(*1*,*2*)]) **unfolding** *exta* **by** *auto* **moreover**
guess *b0 b1* **apply**(*rule ksimplex-extrema-strong*[*OF goal1*(*1*) *assms*(*3*)]) **. note** *extb* = *this*[*rule-format*]
have a':a' = *b0* **apply**(*rule ksimplex-fix-plane-p*[*OF goal1*(*1*−*2*) *assms*(*4*), *of* - *b1*]) **unfolding** *goal1 extb* **using** *extb*(*1*,*2*) *assms*(*5*) **by** *auto*
moreover have ∗:*b1* = *a1* **unfolding** *kle-antisym*[*THEN sym*, *of b1 a1 n*] **using** *exta extb* **using** *goal1*(*3*) **unfolding** *a a'* **by** *blast* **moreover**
have *a0* = *b0* **apply**(*rule ext*) **proof**− **case** *goal1* **show** *a0 x* = *b0 x*
using ∗[*THEN cong*, *of x x*] **unfolding** *exta extb* **apply**−**apply**(*cases* *x*∈{*1*..*n*}) **by** *auto* **qed**
ultimately show s' = *s* **apply**−**apply**(*rule lem*[*OF goal1*(*3*) - *goal1*(*2*) *assms*(*2*)]) **by** *auto* **qed**
show *?thesis* **unfolding** *card-1-exists* **apply**(*rule ex1I*[*of* - *s*]) **unfolding** *mem-Collect-eq* **apply**(*rule*,*rule assms*(*1*))
apply(*rule-tac x*=*a* **in** *bexI*) **prefer** *3* **apply**(*erule conjE bexE*)+ **apply**(*rule-tac* a'=*b* **in** *lem*) **using** *assms*(*1*−*2*) **by** *auto* **qed**

lemma *ksimplex-replace-2*:
assumes *ksimplex p n s a* ∈ *s n* ≠ *0* ~(∃ *j*∈{*1*..*n*}. ∀ *x*∈*s* − {*a*}. *x j* = *0*) ~(∃ *j*∈{*1*..*n*}. ∀ *x*∈*s* − {*a*}. *x j* = *p*)
shows *card* {s'. *ksimplex p n s'* ∧ (∃ *b*∈s'. s' − {*b*} = *s* − {*a*})} = *2* (**is** *card ?A* = *2*) **proof**−
have *lem1*:⋀*a a' s s'*. s' − {a'} = *s* − {*a*} ⟹ a' = *a* ⟹ a' ∈ s' ⟹ *a* ∈ *s* ⟹ s' = *s* **by** *auto*
have *lem2*:⋀*a b*. *a*∈*s* ⟹ *b*≠*a* ⟹ *s* ≠ *insert b* (*s* − {*a*}) **proof case** *goal1*
hence *a*∈*insert b* (*s* − {*a*}) **by** *auto* **hence** *a*∈ *s* − {*a*} **unfolding** *insert-iff* **using** *goal1* **by** *auto*
thus *False* **by** *auto* **qed**
guess *a0 a1* **apply**(*rule ksimplex-extrema-strong*[*OF assms*(*1*,*3*)]) **. note** *a0a1*=*this*
{ assume *a*=*a0*
have ∗:⋀*P Q*. (*P* ∨ *Q*) ⟹ ¬ *P* ⟹ *Q* **by** *auto*
have ∃ *x*∈*s*. ¬ *kle n x a0* **apply**(*rule-tac x*=*a1* **in** *bexI*) **proof assume** *as*:*kle n a1 a0*

show *False* **using** *kle-imp-pointwise*[*OF as*,*THEN spec*[**where** *x*=*1*]] **unfolding** *a0a1*(*5*)[*THEN spec*[**where** *x*=*1*]]
using *assms*(*3*) **by** *auto* **qed**(*insert a0a1*,*auto*)
hence ∃ *y*∈*s*. ∃ *k*∈{*1*..*n*}. ∀ *j*. *y j* = (*if j* = *k then a0 j* + *1 else a0 j*)
apply(*rule-tac* ∗[*OF ksimplex-successor*[*OF assms*(*1*−*2*),*unfolded* ‹*a*=*a0*›]])
by *auto*
then guess *a2* **.. from** *this*(*2*) **guess** *k* **.. note** *k*=*this* **note** *a2*=‹*a2*∈*s*›
def *a3* ≡ λ*j*. *if j* = *k then a1 j* + *1 else a1 j*
have *a3* ∉ *s* **proof assume** *a3*∈*s* **hence** *kle n a3 a1* **using** *a0a1*(*4*) **by** *auto*
thus *False* **apply**(*drule-tac kle-imp-pointwise*) **unfolding** *a3-def*
apply(*erule-tac x*=*k* **in** *allE*) **by** *auto* **qed**
hence *a3* ≠ *a0 a3* ≠ *a1* **using** *a0a1* **by** *auto*
have *a2* ≠ *a0* **using** *k*(*2*)[*THEN spec*[**where** *x*=*k*]] **by** *auto*
have *lem3*:⋀*x*. *x*∈(*s* − {*a0*}) ⟹ *kle n a2 x* **proof**(*rule ccontr*) **case** *goal1*
hence *as*:*x*∈*s x*≠*a0* **by** *auto*
have *kle n a2 x* ∨ *kle n x a2* **using** *ksimplexD*(*6*)[*OF assms*(*1*)] **and** *as* ‹*a2*∈*s*› **by** *auto* **moreover**
have *kle n a0 x* **using** *a0a1*(*4*) *as* **by** *auto*
ultimately have *x* = *a0* ∨ *x* = *a2* **apply**−**apply**(*rule kle-adjacent*[*OF k*(*2*)])
using *goal1*(*2*) **by** *auto*
hence *x* = *a2* **using** *as* **by** *auto* **thus** *False* **using** *goal1*(*2*) **using** *kle-refl*
by *auto* **qed**
let *?s* = *insert a3* (*s* − {*a0*}) **have** *ksimplex p n ?s* **apply**(*rule ksimplexI*)
proof(*rule-tac*[*2*−] *ballI*,*rule-tac*[*4*] *ballI*)
show *card ?s* = *n* + *1* **using** *ksimplexD*(*2*−*3*)[*OF assms*(*1*)]
using ‹*a3*≠*a0*› ‹*a3*∉*s*› ‹*a0*∈*s*› **by**(*auto simp add*:*card-insert-if*)
fix *x* **assume** *x*:*x* ∈ *insert a3* (*s* − {*a0*})
show ∀ *j*. *x j* ≤ *p* **proof**(*rule*,*cases x* = *a3*)
fix *j* **case** *False* **thus** *x j*≤*p* **using** *x ksimplexD*(*4*)[*OF assms*(*1*)] **by** *auto*
next
fix *j* **case** *True* **show** *x j*≤*p* **unfolding** *True* **proof**(*cases j*=*k*)
case *False* **thus** *a3 j* ≤*p* **unfolding** *True a3-def* **using** ‹*a1*∈*s*› *ksimplexD*(*4*)[*OF assms*(*1*)] **by** *auto* **next**
guess *a4* **using** *assms*(*5*)[*unfolded bex-simps ball-simps*,*rule-format*,*OF k*(*1*)] **.. note** *a4*=*this*
have *a2 k* ≤ *a4 k* **using** *lem3*[*OF a4*(*1*)[*unfolded* ‹*a*=*a0*›],*THEN kle-imp-pointwise*] **by** *auto*
also have ... < *p* **using** *ksimplexD*(*4*)[*OF assms*(*1*),*rule-format*,*of a4 k*] **using** *a4* **by** *auto*
finally have ∗:*a0 k* + *1* < *p* **unfolding** *k*(*2*)[*rule-format*] **by** *auto*
case *True* **thus** *a3 j* ≤*p* **unfolding** *a3-def* **unfolding** *a0a1*(*5*)[*rule-format*]
using *k*(*1*) *k*(*2*)*assms*(*5*) **using** ∗ **by** *simp* **qed qed**
show ∀ *j*. *j* ∉ {*1*..*n*} ⟶ *x j* = *p* **proof**(*rule*,*rule*,*cases x*=*a3*) **fix** *j*::*nat* **assume** *j*:*j*∉{*1*..*n*}
{ case *False* **thus** *x j* = *p* **using** *j x ksimplexD*(*5*)[*OF assms*(*1*)] **by** *auto* **}**
case *True* **show** *x j* = *p* **unfolding** *True a3-def* **using** *j k*(*1*)
using *ksimplexD*(*5*)[*OF assms*(*1*),*rule-format*,*OF* ‹*a1*∈*s*› *j*] **by** *auto* **qed**
fix *y* **assume** *y*:*y*∈*insert a3* (*s* − {*a0*})
have *lem4*:⋀*x*. *x*∈*s* ⟹ *x*≠*a0* ⟹ *kle n x a3* **proof**− **case** *goal1*

guess *kk* **using** *a0a1(4)*[*rule-format*,*OF* ‹*x*∈*s*›,*THEN conjunct2*,*unfolded kle-def*]
apply−**apply**(*erule exE*,*erule conjE*) **. note** *kk*=*this*
have *k*∉*kk* **proof assume** *k*∈*kk*
hence *a1 k* = *x k* + *1* **using** *kk* **by** *auto*
hence *a0 k* = *x k* **unfolding** *a0a1(5)*[*rule-format*] **using** *k(1)* **by** *auto*
hence *a2 k* = *x k* + *1* **unfolding** *k(2)*[*rule-format*] **by** *auto* **moreover**
have *a2 k* ≤ *x k* **using** *lem3*[*of x*,*THEN kle-imp-pointwise*] *goal1* **by** *auto*

ultimately show *False* **by** *auto* **qed**
thus *?case* **unfolding** *kle-def* **apply**(*rule-tac x*=*insert k kk* **in** *exI*) **using** *kk(1)*
unfolding *a3-def kle-def kk(2)*[*rule-format*] **using** *k(1)* **by** *auto* **qed**
show *kle n x y* ∨ *kle n y x* **proof**(*cases y*=*a3*)
case *True* **show** *?thesis* **unfolding** *True* **apply**(*cases x*=*a3*) **defer apply**(*rule disjI1*,*rule lem4*)
using *x* **by** *auto* **next**
case *False* **show** *?thesis* **proof**(*cases x*=*a3*) **case** *True* **show** *?thesis* **unfolding** *True*
apply(*rule disjI2*,*rule lem4*) **using** *y False* **by** *auto* **next**
case *False* **thus** *?thesis* **apply**(*rule-tac ksimplexD(6)*[*OF assms(1)*,*rule-format*])

using *x y* ‹*y*≠*a3*› **by** *auto* **qed qed qed**
hence *insert a3* (*s* − {*a0*}) ∈ *?A* **unfolding** *mem-Collect-eq* **apply**−**apply**(*rule*,*assumption*)
apply(*rule-tac x*=*a3* **in** *bexI*) **unfolding** ‹*a*=*a0*› **using** ‹*a3*∉*s*› **by** *auto* **moreover**
have *s* ∈ *?A* **using** *assms(1,2)* **by** *auto* **ultimately have** *?A* ⊇ {*s*, *insert a3* (*s* − {*a0*})} **by** *auto*
moreover have *?A* ⊆ {*s*, *insert a3* (*s* − {*a0*})} **apply**(*rule*) **unfolding** *mem-Collect-eq* **proof**(*erule conjE*)
fix *s′* **assume** *as*:*ksimplex p n s′* **and** ∃ *b*∈*s′*. *s′* − {*b*} = *s* − {*a*}
from *this(2)* **guess** *a′* **.. note** *a′*=*this*
guess *a-min a-max* **apply**(*rule ksimplex-extrema-strong*[*OF as assms(3)*]) **. note** *min-max*=*this*
have ∗:∀ *x*∈*s′* − {*a′*}. *x k* = *a2 k* **unfolding** *a′* **proof fix** *x* **assume** *x*:*x*∈*s*−{*a*}
hence *kle n a2 x* **apply**−**apply**(*rule lem3*) **using** ‹*a*=*a0*› **by** *auto*
hence *a2 k* ≤ *x k* **apply**(*drule-tac kle-imp-pointwise*) **by** *auto* **moreover**
have *x k* ≤ *a2 k* **unfolding** *k(2)*[*rule-format*] **using** *a0a1(4)*[*rule-format*,*of x*,*THEN conjunct1*]
unfolding *kle-def* **using** *x* **by** *auto* **ultimately show** *x k* = *a2 k* **by** *auto* **qed**
have ∗∗:*a′*=*a-min* ∨ *a′*=*a-max* **apply**(*rule ksimplex-fix-plane*[*OF a′(1) k(1)* ∗]) **using** *min-max* **by** *auto*
show *s′* ∈ {*s*, *insert a3* (*s* − {*a0*})} **proof**(*cases a′*=*a-min*)
case *True* **have** *a-max* = *a1* **unfolding** *kle-antisym*[*THEN sym*,*of a-max a1 n*] **apply**(*rule*)
apply(*rule a0a1(4)*[*rule-format*,*THEN conjunct2*]) **defer proof**(*rule min-max(4)*[*rule-format*,*THEN conjunct2*])

show *a1∈s′* **using** *a′* **unfolding** ‹*a=a0*› **using** *a0a1* **by** *auto*
show *a-max* ∈ *s* **proof**(*rule ccontr*) **assume** *a-max∉s*
hence *a-max* = *a′* **using** *a′ min-max* **by** *auto*
thus *False* **unfolding** *True* **using** *min-max* **by** *auto* **qed** **qed**
hence ∀ *i*. *a-max i* = *a1 i* **by** *auto*
hence *a′* = *a* **unfolding** *True* ‹*a=a0*› **apply**−**apply**(*subst expand-fun-eq*,*rule*)
apply(*erule-tac x=x* **in** *allE*) **unfolding** *a0a1*(*5*)[*rule-format*] *min-max*(*5*)[*rule-format*]
proof− **case** *goal1* **thus** *?case* **apply**(*cases x∈*{*1..n*}) **by** *auto* **qed**
hence *s′* = *s* **apply**−**apply**(*rule lem1*[*OF a′*(*2*)]) **using** ‹*a∈s*› ‹*a′∈s′*› **by** *auto*
thus *?thesis* **by** *auto* **next**
case *False* **hence** *as*:*a′* = *a-max* **using** ∗∗ **by** *auto*
have *a-min* = *a2* **unfolding** *kle-antisym*[*THEN sym*, *of - - n*] **apply** *rule*
apply(*rule min-max*(*4*)[*rule-format*,*THEN conjunct1*]) **defer** **proof**(*rule lem3*)
show *a-min* ∈ *s* − {*a0*} **unfolding** *a′*(*2*)[*THEN sym*,*unfolded* ‹*a=a0*›]
unfolding *as* **using** *min-max*(*1−3*) **by** *auto*
have *a2* ≠ *a* **unfolding** ‹*a=a0*› **using** *k*(*2*)[*rule-format*,*of k*] **by** *auto*
hence *a2* ∈ *s* − {*a*} **using** *a2* **by** *auto* **thus** *a2* ∈ *s′* **unfolding** *a′*(*2*)[*THEN sym*] **by** *auto* **qed**
hence ∀ *i*. *a-min i* = *a2 i* **by** *auto*
hence *a′* = *a3* **unfolding** *as* ‹*a=a0*› **apply**−**apply**(*subst expand-fun-eq*,*rule*)
apply(*erule-tac x=x* **in** *allE*) **unfolding** *a0a1*(*5*)[*rule-format*] *min-max*(*5*)[*rule-format*]
unfolding *a3-def k*(*2*)[*rule-format*] **unfolding** *a0a1*(*5*)[*rule-format*]
proof− **case** *goal1*
show *?case* **unfolding** *goal1* **apply**(*cases x∈*{*1..n*}) **defer** **apply**(*cases x=k*)
using ‹*k∈*{*1..n*}› **by** *auto* **qed**
hence *s′* = *insert a3* (*s* − {*a0*}) **apply**−**apply**(*rule lem1*) **defer** **apply** *assumption*
apply(*rule a′*(*1*)) **unfolding** *a′* ‹*a=a0*› **using** ‹*a3∉s*› **by** *auto*
thus *?thesis* **by** *auto* **qed** **qed**
ultimately have ∗:*?A* = {*s*, *insert a3* (*s* − {*a0*})} **by** *blast*
have *s* ≠ *insert a3* (*s* − {*a0*}) **using** ‹*a3∉s*› **by** *auto*
hence *?thesis* **unfolding** ∗ **by** *auto* **}** **moreover**
{ assume *a=a1*
have ∗:⋀*P Q*. (*P* ∨ *Q*) ⟹ ¬ *P* ⟹ *Q* **by** *auto*
have ∃ *x∈s*. ¬ *kle n a1 x* **apply**(*rule-tac x=a0* **in** *bexI*) **proof assume** *as*:*kle n a1 a0*
show *False* **using** *kle-imp-pointwise*[*OF as*,*THEN spec*[**where** *x=1*]] **unfolding** *a0a1*(*5*)[*THEN spec*[**where** *x=1*]]
using *assms*(*3*) **by** *auto* **qed**(*insert a0a1*,*auto*)
hence ∃ *y∈s*. ∃ *k∈*{*1..n*}. ∀ *j*. *a1 j* = (*if j* = *k then y j* + *1 else y j*)
apply(*rule-tac* ∗[*OF ksimplex-predecessor*[*OF assms*(*1−2*),*unfolded* ‹*a=a1*›]]) **by** *auto*
then guess *a2* .. **from** *this*(*2*) **guess** *k* .. **note** *k=this* **note** *a2=*‹*a2∈s*›
def *a3* ≡ λ*j*. *if j* = *k then a0 j* − *1 else a0 j*
have *a2* ≠ *a1* **using** *k*(*2*)[*THEN spec*[**where** *x=k*]] **by** *auto*
have *lem3*:⋀*x*. *x∈*(*s* − {*a1*}) ⟹ *kle n x a2* **proof**(*rule ccontr*) **case** *goal1*

hence *as*:$x \in s$ $x \neq a1$ **by** *auto*
have *kle n a2 x* $\vee$ *kle n x a2* **using** *ksimplexD*(*6*)[*OF assms*(*1*)] **and** *as* ‹*a2*∈*s*› **by** *auto* **moreover**
have *kle n x a1* **using** *a0a1*(*4*) *as* **by** *auto*
ultimately have $x = a2 \vee x = a1$ **apply**−**apply**(*rule kle-adjacent*[*OF k*(*2*)]) **using** *goal1*(*2*) **by** *auto*
hence $x = a2$ **using** *as* **by** *auto* **thus** *False* **using** *goal1*(*2*) **using** *kle-refl* **by** *auto* **qed**
have *a0 k* $\neq$ *0* **proof**−
guess *a4* **using** *assms*(*4*)[*unfolded bex-simps ball-simps*,*rule-format*,*OF* ‹*k*∈{*1*..*n*}›] **..** **note** *a4*=*this*
have *a4 k* $\leq$ *a2 k* **using** *lem3*[*OF a4*(*1*)[*unfolded* ‹*a*=*a1*›],*THEN kle-imp-pointwise*] **by** *auto*
moreover have *a4 k* $>$ *0* **using** *a4* **by** *auto* **ultimately have** *a2 k* $>$ *0* **by** *auto*
hence *a1 k* $>$ *1* **unfolding** *k*(*2*)[*rule-format*] **by** *simp*
thus *?thesis* **unfolding** *a0a1*(*5*)[*rule-format*] **using** *k*(*1*) **by** *simp* **qed**
hence *lem4*:$\forall j.$ *a0 j* = (*if j*=*k then a3 j* + *1 else a3 j*) **unfolding** *a3-def* **by** *simp*
have ¬ *kle n a0 a3* **apply**(*rule ccontr*) **unfolding** *not-not* **apply**(*drule kle-imp-pointwise*)
unfolding *lem4*[*rule-format*] **apply**(*erule-tac x*=*k* **in** *allE*) **by** *auto*
hence *a3* $\notin$ *s* **using** *a0a1*(*4*) **by** *auto*
hence *a3* $\neq$ *a1 a3* $\neq$ *a0* **using** *a0a1* **by** *auto*
let *?s* = *insert a3* (*s* − {*a1*}) **have** *ksimplex p n ?s* **apply**(*rule ksimplexI*) **proof**(*rule-tac*[*2*−] *ballI*,*rule-tac*[*4*] *ballI*)
show *card ?s* = *n*+*1* **using** *ksimplexD*(*2*−*3*)[*OF assms*(*1*)]
using ‹*a3*≠*a0*› ‹*a3*∉*s*› ‹*a1*∈*s*› **by**(*auto simp add*:*card-insert-if*)
fix *x* **assume** *x*:$x \in$ *insert a3* (*s* − {*a1*})
show $\forall j.$ *x j* $\leq$ *p* **proof**(*rule*,*cases x* = *a3*)
fix *j* **case** *False* **thus** *x j*≤*p* **using** *x ksimplexD*(*4*)[*OF assms*(*1*)] **by** *auto*
next
fix *j* **case** *True* **show** *x j*≤*p* **unfolding** *True* **proof**(*cases j*=*k*)
case *False* **thus** *a3 j* ≤*p* **unfolding** *True a3-def* **using** ‹*a0*∈*s*› *ksimplexD*(*4*)[*OF assms*(*1*)] **by** *auto* **next**
guess *a4* **using** *assms*(*5*)[*unfolded bex-simps ball-simps*,*rule-format*,*OF k*(*1*)] **..** **note** *a4*=*this*
case *True* **have** *a3 k* $\leq$ *a0 k* **unfolding** *lem4*[*rule-format*] **by** *auto*
also have ... $\leq$ *p* **using** *ksimplexD*(*4*)[*OF assms*(*1*),*rule-format*,*of a0 k*] *a0a1* **by** *auto*
finally show *a3 j* $\leq$ *p* **unfolding** *True* **by** *auto* **qed** **qed**
show $\forall j.$ *j* $\notin$ {*1*..*n*} $\longrightarrow$ *x j* = *p* **proof**(*rule*,*rule*,*cases x*=*a3*) **fix** *j*::*nat* **assume** *j*:*j*∉{*1*..*n*}
{ **case** *False* **thus** *x j* = *p* **using** *j x ksimplexD*(*5*)[*OF assms*(*1*)] **by** *auto* }
case *True* **show** *x j* = *p* **unfolding** *True a3-def* **using** *j k*(*1*)
using *ksimplexD*(*5*)[*OF assms*(*1*),*rule-format*,*OF* ‹*a0*∈*s*› *j*] **by** *auto* **qed**
fix *y* **assume** *y*:*y*∈*insert a3* (*s* − {*a1*})
have *lem4*:$\bigwedge x.$ *x*∈*s* $\Longrightarrow$ *x*≠*a1* $\Longrightarrow$ *kle n a3 x* **proof**− **case** *goal1* **hence** ∗:*x*∈*s* − {*a1*} **by** *auto*
have *kle n a3 a2* **proof**− **have** *kle n a0 a1* **using** *a0a1* **by** *auto* **then**

guess *kk* **unfolding** *kle-def* **..**
thus *?thesis* **unfolding** *kle-def* **apply**(*rule-tac x=kk* **in** *exI*) **unfolding** *lem4*[*rule-format*] *k*(*2*)[*rule-format*]
apply(*rule*)**defer proof**(*rule*) **case** *goal1* **thus** *?case* **apply**−**apply**(*erule conjE*)
apply(*erule-tac*[!] *x=j* **in** *allE*) **apply**(*cases j∈kk*) **apply**(*case-tac*[!] *j=k*) **by** *auto* **qed** *auto* **qed moreover**
have *kle n a3 a0* **unfolding** *kle-def lem4*[*rule-format*] **apply**(*rule-tac x={k}* **in** *exI*) **using** *k*(*1*) **by** *auto*
ultimately show *?case* **apply**−**apply**(*rule kle-between-l*[*of - a0 - a2*]) **using** *lem3*[*OF ∗*]
using *a0a1*(*4*)[*rule-format*,*OF goal1*(*1*)] **by** *auto* **qed**
show *kle n x y ∨ kle n y x* **proof**(*cases y=a3*)
case *True* **show** *?thesis* **unfolding** *True* **apply**(*cases x=a3*) **defer apply**(*rule disjI2*,*rule lem4*)
using *x* **by** *auto* **next**
case *False* **show** *?thesis* **proof**(*cases x=a3*) **case** *True* **show** *?thesis* **unfolding** *True*
apply(*rule disjI1*,*rule lem4*) **using** *y False* **by** *auto* **next**
case *False* **thus** *?thesis* **apply**(*rule-tac ksimplexD*(*6*)[*OF assms*(*1*),*rule-format*])

using *x y* ‹*y≠a3*› **by** *auto* **qed qed qed**
hence *insert a3* (*s* − {*a1*}) ∈ *?A* **unfolding** *mem-Collect-eq* **apply**−**apply**(*rule*,*assumption*)
apply(*rule-tac x=a3* **in** *bexI*) **unfolding** ‹*a=a1*› **using** ‹*a3*∉*s*› **by** *auto* **moreover**
have *s* ∈ *?A* **using** *assms*(*1*,*2*) **by** *auto* **ultimately have** *?A* ⊇ {*s*, *insert a3* (*s* − {*a1*})} **by** *auto*
moreover have *?A* ⊆ {*s*, *insert a3* (*s* − {*a1*})} **apply**(*rule*) **unfolding** *mem-Collect-eq* **proof**(*erule conjE*)
fix *s′* **assume** *as*:*ksimplex p n s′* **and** ∃ *b*∈*s′*. *s′* − {*b*} = *s* − {*a*}
from *this*(*2*) **guess** *a′* **.. note** *a′=this*
guess *a-min a-max* **apply**(*rule ksimplex-extrema-strong*[*OF as assms*(*3*)]) **. note** *min-max=this*
have ∗:∀ *x*∈*s′* − {*a′*}. *x k* = *a2 k* **unfolding** *a′* **proof fix** *x* **assume** *x*:*x*∈*s*−{*a*}
hence *kle n x a2* **apply**−**apply**(*rule lem3*) **using** ‹*a=a1*› **by** *auto*
hence *x k* ≤ *a2 k* **apply**(*drule-tac kle-imp-pointwise*) **by** *auto* **moreover**
{ have *a2 k* ≤ *a0 k* **using** *k*(*2*)[*rule-format*,*of k*] **unfolding** *a0a1*(*5*)[*rule-format*] **using** *k*(*1*) **by** *simp*
also have ... ≤ *x k* **using** *a0a1*(*4*)[*rule-format*,*of x*,*THEN conjunct1*,*THEN kle-imp-pointwise*] *x* **by** *auto*
finally have *a2 k* ≤ *x k* **. } ultimately show** *x k* = *a2 k* **by** *auto* **qed**
have ∗∗:*a′=a-min ∨ a′=a-max* **apply**(*rule ksimplex-fix-plane*[*OF a′*(*1*) *k*(*1*) ∗]) **using** *min-max* **by** *auto*
have *a2* ≠ *a1* **proof assume** *as*:*a2* = *a1*
show *False* **using** *k*(*2*) **unfolding** *as* **apply**(*erule-tac x=k* **in** *allE*) **by** *auto* **qed**
hence *a2′*:*a2* ∈ *s′* − {*a′*} **unfolding** *a′* **using** *a2* **unfolding** ‹*a=a1*› **by** *auto*
show *s′* ∈ {*s*, *insert a3* (*s* − {*a1*})} **proof**(*cases a′=a-min*)

case *True* **have** *a-max* ∈ *s* − {*a1*} **using** *min-max* **unfolding** *a′*(*2*)[*unfolded* ‹*a*=*a1*›,*THEN sym*] *True* **by** *auto*
hence *a-max* = *a2* **unfolding** *kle-antisym*[*THEN sym*,*of a-max a2 n*] **apply**−**apply**(*rule*)
apply(*rule lem3*,*assumption*) **apply**(*rule min-max*(*4*)[*rule-format*,*THEN conjunct2*]) **using** *a2′* **by** *auto*
hence *a-max*:∀ *i*. *a-max i* = *a2 i* **by** *auto*
have ∗:∀ *j*. *a2 j* = (*if j*∈{*1..n*} *then a3 j* + *1 else a3 j*)
using *k*(*2*) **unfolding** *lem4*[*rule-format*] *a0a1*(*5*)[*rule-format*] **apply**−**apply**(*rule*,*erule-tac x*=*j* **in** *allE*)
proof− **case** *goal1* **thus** *?case* **apply**(*cases j*∈{*1..n*},*case-tac*[!] *j*=*k*) **by** *auto* **qed**
have ∀ *i*. *a-min i* = *a3 i* **using** *a-max* **apply**−**apply**(*rule*,*erule-tac x*=*i* **in** *allE*)
unfolding *min-max*(*5*)[*rule-format*] ∗[*rule-format*] **proof**− **case** *goal1*
thus *?case* **apply**(*cases i*∈{*1..n*}) **by** *auto* **qed** **hence** *a-min* = *a3* **unfolding** *expand-fun-eq* **.**
hence *s′* = *insert a3* (*s* − {*a1*}) **using** *a′* **unfolding** ‹*a*=*a1*› *True* **by** *auto*
thus *?thesis* **by** *auto* **next**
case *False* **hence** *as*:*a′*=*a-max* **using** ∗∗ **by** *auto*
have *a-min* = *a0* **unfolding** *kle-antisym*[*THEN sym*,*of* - - *n*] **apply**(*rule*)
apply(*rule min-max*(*4*)[*rule-format*,*THEN conjunct1*]) **defer** **apply**(*rule a0a1*(*4*)[*rule-format*,*THEN conjunct1*]) **proof**−
have *a-min* ∈ *s* − {*a1*} **using** *min-max*(*1*,*3*) **unfolding** *a′*(*2*)[*THEN sym*,*unfolded* ‹*a*=*a1*›] *as* **by** *auto*
thus *a-min* ∈ *s* **by** *auto* **have** *a0* ∈ *s* − {*a1*} **using** *a0a1*(*1*−*3*) **by** *auto*
thus *a0* ∈ *s′*
unfolding *a′*(*2*)[*THEN sym*,*unfolded* ‹*a*=*a1*›] **by** *auto* **qed**
hence ∀ *i*. *a-max i* = *a1 i* **unfolding** *a0a1*(*5*)[*rule-format*] *min-max*(*5*)[*rule-format*] **by** *auto*
hence *s′* = *s* **apply**−**apply**(*rule lem1*[*OF a′*(*2*)]) **using** ‹*a*∈*s*› ‹*a′*∈*s′*› **unfolding** *as* ‹*a*=*a1*› **unfolding** *expand-fun-eq* **by** *auto*
thus *?thesis* **by** *auto* **qed** **qed**
ultimately have ∗:*?A* = {*s*, *insert a3* (*s* − {*a1*})} **by** *blast*
have *s* ≠ *insert a3* (*s* − {*a1*}) **using** ‹*a3*∉*s*› **by** *auto*
hence *?thesis* **unfolding** ∗ **by** *auto* **}** **moreover**
{ assume *as*:*a*≠*a0 a*≠*a1* **have** ¬ (∀ *x*∈*s*. *kle n a x*) **proof case** *goal1*
have *a*=*a0* **unfolding** *kle-antisym*[*THEN sym*,*of* - - *n*] **apply**(*rule*)
using *goal1 a0a1 assms*(*2*) **by** *auto* **thus** *False* **using** *as* **by** *auto* **qed**
hence ∃ *y*∈*s*. ∃ *k*∈{*1..n*}. ∀ *j*. *a j* = (*if j* = *k then y j* + *1 else y j*) **using** *ksimplex-predecessor*[*OF assms*(*1*−*2*)] **by** *blast*
then guess *u* **..** **from** *this*(*2*) **guess** *k* **..** **note** *k* = *this*[*rule-format*] **note** *u* = ‹*u*∈*s*›
have ¬ (∀ *x*∈*s*. *kle n x a*) **proof case** *goal1*
have *a*=*a1* **unfolding** *kle-antisym*[*THEN sym*,*of* - - *n*] **apply**(*rule*)
using *goal1 a0a1 assms*(*2*) **by** *auto* **thus** *False* **using** *as* **by** *auto* **qed**
hence ∃ *y*∈*s*. ∃ *k*∈{*1..n*}. ∀ *j*. *y j* = (*if j* = *k then a j* + *1 else a j*) **using** *ksimplex-successor*[*OF assms*(*1*−*2*)] **by** *blast*
then guess *v* **..** **from** *this*(*2*) **guess** *l* **..** **note** *l* = *this*[*rule-format*] **note** *v* =

‹$v{\in}s$›
def $a' \equiv \lambda j.$ *if* $j = l$ *then* $u\ j + 1$ *else* $u\ j$
have *kl*:$k \neq l$ **proof assume** $k{=}l$ **have** $*{:}\bigwedge P.$ (*if* P *then* (1::*nat*) *else* 0) $\neq 2$ **by** *auto*
thus *False* **using** *ksimplexD*(6)[*OF assms*(1),*rule-format*,*OF* $u\ v$] **unfolding** *kle-def*
unfolding $l(2)\ k(2)$ ‹$k{=}l$› **apply**−**apply**(*erule disjE*)**apply**(*erule-tac*[!] *exE conjE*)+
apply(*erule-tac*[!] $x{=}l$ **in** *allE*)+ **by**(*auto simp add*: $*$) **qed**
hence *aa'*:$a'{\neq}a$ **apply**−**apply** *rule* **unfolding** *expand-fun-eq* **unfolding** a'*-def* $k(2)$
apply(*erule-tac* $x{=}l$ **in** *allE*) **by** *auto*
have $a' \notin s$ **apply**(*rule*) **apply**(*drule ksimplexD*(6)[*OF assms*(1),*rule-format*,*OF* ‹$a{\in}s$›]) **proof**(*cases kle n a a'*)
case *goal2* **hence** *kle n a' a* **by** *auto* **thus** *False* **apply**(*drule-tac kle-imp-pointwise*)
apply(*erule-tac* $x{=}l$ **in** *allE*) **unfolding** a'*-def* $k(2)$ **using** *kl* **by** *auto* **next**
case *True* **thus** *False* **apply**(*drule-tac kle-imp-pointwise*)
apply(*erule-tac* $x{=}k$ **in** *allE*) **unfolding** a'*-def* $k(2)$ **using** *kl* **by** *auto* **qed**
have *kle-uv*:*kle n u a kle n u a' kle n a v kle n a' v* **unfolding** *kle-def* **apply**−
apply(*rule-tac*[1] $x{=}\{k\}$ **in** *exI*,*rule-tac*[2] $x{=}\{l\}$ **in** *exI*)
apply(*rule-tac*[3] $x{=}\{l\}$ **in** *exI*,*rule-tac*[4] $x{=}\{k\}$ **in** *exI*)
unfolding $l(2)\ k(2)$ a'*-def* **using** $l(1)\ k(1)$ **by** *auto*
have *uxv*:$\bigwedge x.$ *kle n u x* $\Longrightarrow$ *kle n x v* $\Longrightarrow (x = u) \vee (x = a) \vee (x = a') \vee (x = v)$
proof− **case** *goal1* **thus** *?case* **proof**(*cases* $x\ k = u\ k$, *case-tac*[!] $x\ l = u\ l$)
assume *as*:$x\ l = u\ l\ x\ k = u\ k$
have $x = u$ **unfolding** *expand-fun-eq*
using *goal1*(2)[*THEN kle-imp-pointwise*,*unfolded* $l(2)$] **unfolding** $k(2)$ **apply**−
using *goal1*(1)[*THEN kle-imp-pointwise*] **apply**−**apply** *rule* **apply**(*erule-tac* $x{=}xa$ **in** *allE*)+ **proof**− **case** *goal1*
thus *?case* **apply**(*cases* $x{=}l$) **apply**(*case-tac*[!] $x{=}k$) **using** *as* **by** *auto* **qed**
thus *?case* **by** *auto* **next**
assume *as*:$x\ l \neq u\ l\ x\ k = u\ k$
have $x = a'$ **unfolding** *expand-fun-eq* **unfolding** a'*-def*
using *goal1*(2)[*THEN kle-imp-pointwise*] **unfolding** $l(2)\ k(2)$ **apply**−
using *goal1*(1)[*THEN kle-imp-pointwise*] **apply**−**apply** *rule* **apply**(*erule-tac* $x{=}xa$ **in** *allE*)+ **proof**− **case** *goal1*
thus *?case* **apply**(*cases* $x{=}l$) **apply**(*case-tac*[!] $x{=}k$) **using** *as* **by** *auto* **qed**
thus *?case* **by** *auto* **next**
assume *as*:$x\ l = u\ l\ x\ k \neq u\ k$
have $x = a$ **unfolding** *expand-fun-eq*
using *goal1*(2)[*THEN kle-imp-pointwise*] **unfolding** $l(2)\ k(2)$ **apply**−
using *goal1*(1)[*THEN kle-imp-pointwise*] **apply**−**apply** *rule* **apply**(*erule-tac* $x{=}xa$ **in** *allE*)+ **proof**− **case** *goal1*
thus *?case* **apply**(*cases* $x{=}l$) **apply**(*case-tac*[!] $x{=}k$) **using** *as* **by** *auto* **qed**
thus *?case* **by** *auto* **next**
assume *as*:$x\ l \neq u\ l\ x\ k \neq u\ k$
have $x = v$ **unfolding** *expand-fun-eq*

using *goal1*(*2*)[*THEN kle-imp-pointwise*] **unfolding** *l*(*2*) *k*(*2*) **apply**−
using *goal1*(*1*)[*THEN kle-imp-pointwise*] **apply**−**apply** *rule* **apply**(*erule-tac x*=*xa* **in** *allE*)+ **proof**− **case** *goal1*
thus *?case* **apply**(*cases x*=*l*) **apply**(*case-tac*[!] *x*=*k*) **using** *as* ‹*k*≠*l*› **by** *auto* **qed thus** *?case* **by** *auto* **qed qed**
have *uv*:*kle n u v* **apply**(*rule kle-trans*[*OF kle-uv*(*1*,*3*)]) **using** *ksimplexD*(*6*)[*OF assms*(*1*)] **using** *u v* **by** *auto*
have *lem3*:⋀*x*. *x*∈*s* ⟹ *kle n v x* ⟹ *kle n a*′ *x* **apply**(*rule kle-between-r*[*of - u - v*])
prefer *3* **apply**(*rule kle-trans*[*OF uv*]) **defer apply**(*rule ksimplexD*(*6*)[*OF assms*(*1*),*rule-format*])
using *kle-uv* ‹*u*∈*s*› **by** *auto*
have *lem4*:⋀*x*. *x*∈*s* ⟹ *kle n x u* ⟹ *kle n x a*′ **apply**(*rule kle-between-l*[*of - u - v*])
prefer *4* **apply**(*rule kle-trans*[*OF - uv*]) **defer apply**(*rule ksimplexD*(*6*)[*OF assms*(*1*),*rule-format*])
using *kle-uv* ‹*v*∈*s*› **by** *auto*
have *lem5*:⋀*x*. *x*∈*s* ⟹ *x*≠*a* ⟹ *kle n x a*′ ∨ *kle n a*′ *x* **proof**− **case** *goal1* **thus** *?case*
proof(*cases kle n v x* ∨ *kle n x u*) **case** *True* **thus** *?thesis* **using** *goal1* **by**(*auto intro*:*lem3 lem4*) **next**
case *False* **hence** ∗:*kle n u x kle n x v* **using** *ksimplexD*(*6*)[*OF assms*(*1*)] **using** *goal1* ‹*u*∈*s*› ‹*v*∈*s*› **by** *auto*
show *?thesis* **using** *uxv*[*OF* ∗] **using** *kle-uv* **using** *goal1* **by** *auto* **qed qed**
have *ksimplex p n* (*insert a*′ (*s* − {*a*})) **apply**(*rule ksimplexI*) **proof**(*rule-tac*[*2*−] *ballI*,*rule-tac*[*4*] *ballI*)
show *card* (*insert a*′ (*s* − {*a*})) = *n* + *1* **using** *ksimplexD*(*2*−*3*)[*OF assms*(*1*)]
using ‹*a*′≠*a*› ‹*a*′∉*s*› ‹*a*∈*s*› **by**(*auto simp add*:*card-insert-if*)
fix *x* **assume** *x*:*x* ∈ *insert a*′ (*s* − {*a*})
show ∀*j*. *x j* ≤ *p* **proof**(*rule*,*cases x* = *a*′)
fix *j* **case** *False* **thus** *x j*≤*p* **using** *x ksimplexD*(*4*)[*OF assms*(*1*)] **by** *auto* **next**
fix *j* **case** *True* **show** *x j*≤*p* **unfolding** *True* **proof**(*cases j*=*l*)
case *False* **thus** *a*′ *j* ≤*p* **unfolding** *True a*′*-def* **using** ‹*u*∈*s*› *ksimplexD*(*4*)[*OF assms*(*1*)] **by** *auto* **next**
case *True* **have** ∗:*a l* = *u l v l* = *a l* + *1* **using** *k*(*2*)[*of l*] *l*(*2*)[*of l*] ‹*k*≠*l*› **by** *auto*
have *u l* + *1* ≤ *p* **unfolding** ∗[*THEN sym*] **using** *ksimplexD*(*4*)[*OF assms*(*1*)] **using** ‹*v*∈*s*› **by** *auto*
thus *a*′ *j* ≤*p* **unfolding** *a*′*-def True* **by** *auto* **qed qed**
show ∀*j*. *j* ∉ {*1*..*n*} ⟶ *x j* = *p* **proof**(*rule*,*rule*,*cases x*=*a*′) **fix** *j*::*nat* **assume** *j*:*j*∉{*1*..*n*}
{ case *False* **thus** *x j* = *p* **using** *j x ksimplexD*(*5*)[*OF assms*(*1*)] **by** *auto* **}**
case *True* **show** *x j* = *p* **unfolding** *True a*′*-def* **using** *j l*(*1*)
using *ksimplexD*(*5*)[*OF assms*(*1*),*rule-format*,*OF* ‹*u*∈*s*› *j*] **by** *auto* **qed**
fix *y* **assume** *y*:*y*∈*insert a*′ (*s* − {*a*})
show *kle n x y* ∨ *kle n y x* **proof**(*cases y*=*a*′)
case *True* **show** *?thesis* **unfolding** *True* **apply**(*cases x*=*a*′) **defer apply**(*rule lem5*) **using** *x* **by** *auto* **next**

case *False* **show** *?thesis* **proof**(*cases x=a′*) **case** *True* **show** *?thesis* **unfolding** *True*
using *lem5*[*of y*] **using** *y* **by** *auto* **next**
case *False* **thus** *?thesis* **apply**(*rule-tac ksimplexD*(*6*)[*OF assms*(*1*),*rule-format*])

using *x y* ‹*y≠a′*› **by** *auto* **qed qed qed**
hence *insert a′* (*s* − {*a*}) ∈ *?A* **unfolding** *mem-Collect-eq* **apply**−**apply**(*rule*,*assumption*)
apply(*rule-tac x=a′* **in** *bexI*) **using** *aa′* ‹*a′*∉*s*› **by** *auto* **moreover**
have *s* ∈ *?A* **using** *assms*(*1*,*2*) **by** *auto* **ultimately have** *?A* ⊇ {*s*, *insert a′* (*s* − {*a*})} **by** *auto*
moreover have *?A* ⊆ {*s*, *insert a′* (*s* − {*a*})} **apply**(*rule*) **unfolding** *mem-Collect-eq* **proof**(*erule conjE*)
fix *s′* **assume** *as*:*ksimplex p n s′* **and** ∃ *b*∈*s′*. *s′* − {*b*} = *s* − {*a*}
from *this*(*2*) **guess** *a″* **.. note** *a″*=*this*
have *u*≠*v* **unfolding** *expand-fun-eq* **unfolding** *l*(*2*) *k*(*2*) **by** *auto*
hence *uv′*:¬ *kle n v u* **using** *uv* **using** *kle-antisym* **by** *auto*
have *u*≠*a v*≠*a* **unfolding** *expand-fun-eq k*(*2*) *l*(*2*) **by** *auto*
hence *uvs′*:*u*∈*s′ v*∈*s′* **using** ‹*u*∈*s*› ‹*v*∈*s*› **using** *a″* **by** *auto*
have *lem6*:*a* ∈ *s′* ∨ *a′* ∈ *s′* **proof**(*cases* ∀ *x*∈*s′*. *kle n x u* ∨ *kle n v x*)
case *False* **then guess** *w* **unfolding** *ball-simps* **.. note** *w*=*this*
hence *kle n u w kle n w v* **using** *ksimplexD*(*6*)[*OF as*] *uvs′* **by** *auto*
hence *w* = *a′* ∨ *w* = *a* **using** *uxv*[*of w*] *uvs′ w* **by** *auto* **thus** *?thesis* **using** *w* **by** *auto* **next**
case *True* **have** ¬ (∀ *x*∈*s′*. *kle n x u*) **unfolding** *ball-simps* **apply**(*rule-tac x=v* **in** *bexI*)
using *uv* ‹*u*≠*v*› **unfolding** *kle-antisym*[*of n u v*,*THEN sym*] **using** ‹*v*∈*s′*› **by** *auto*
hence ∃ *y*∈*s′*. ∃ *k*∈{*1..n*}. ∀ *j*. *y j* = (*if j* = *k then u j* + *1 else u j*) **using** *ksimplex-successor*[*OF as* ‹*u*∈*s′*›] **by** *blast*
then guess *w* **.. note** *w*=*this* **from** *this*(*2*) **guess** *kk* **.. note** *kk*=*this*[*rule-format*]
have ¬ *kle n w u* **apply**−**apply**(*rule*,*drule kle-imp-pointwise*)
apply(*erule-tac x=kk* **in** *allE*) **unfolding** *kk* **by** *auto*
hence ∗:*kle n v w* **using** *True*[*rule-format*,*OF w*(*1*)] **by** *auto*
hence *False* **proof**(*cases kk≠l*) **case** *True* **thus** *False* **using** ∗[*THEN kle-imp-pointwise*, *unfolded l*(*2*) *kk k*(*2*)]
apply(*erule-tac x=l* **in** *allE*) **using** ‹*k*≠*l*› **by** *auto* **next**
case *False* **hence** *kk*≠*k* **using** ‹*k*≠*l*› **by** *auto* **thus** *False* **using** ∗[*THEN kle-imp-pointwise*, *unfolded l*(*2*) *kk k*(*2*)]
apply(*erule-tac x=k* **in** *allE*) **using** ‹*k*≠*l*› **by** *auto* **qed**
thus *?thesis* **by** *auto* **qed**
thus *s′* ∈ {*s*, *insert a′* (*s* − {*a*})} **proof**(*cases a*∈*s′*)
case *True* **hence** *s′* = *s* **apply**−**apply**(*rule lem1*[*OF a″*(*2*)]) **using** *a″* ‹*a*∈*s*› **by** *auto*
thus *?thesis* **by** *auto* **next case** *False* **hence** *a′*∈*s′* **using** *lem6* **by** *auto*
hence *s′* = *insert a′* (*s* − {*a*}) **apply**−**apply**(*rule lem1*[*of - a″ - a′*])
unfolding *a″*(*2*)[*THEN sym*] **using** *a″* **using** ‹*a′*∉*s*› **by** *auto*
thus *?thesis* **by** *auto* **qed qed**
ultimately have ∗:*?A* = {*s*, *insert a′* (*s* − {*a*})} **by** *blast*
have *s* ≠ *insert a′* (*s* − {*a*}) **using** ‹*a′*∉*s*› **by** *auto*

hence *?thesis* **unfolding** $*$ **by** *auto* **}**
ultimately show *?thesis* **by** *auto* **qed**

20.7 Hence another step towards concreteness.

lemma *kuhn-simplex-lemma*:
assumes $\forall s.\ ksimplex\ p\ (n + 1)\ s \longrightarrow (rl\ `\ s \subseteq \{0..n+1\})$
$odd\ (card\{f.\ \exists s\ a.\ ksimplex\ p\ (n + 1)\ s \wedge a \in s \wedge (f = s - \{a\}) \wedge$
$(rl\ `\ f = \{0\ ..\ n\}) \wedge ((\exists j{\in}\{1..n+1\}.\forall x{\in}f.\ x\ j = 0) \vee (\exists j{\in}\{1..n+1\}.\forall x{\in}f.\ x\ j = p))\})$
shows $odd(card\ \{s{\in}\{s.\ ksimplex\ p\ (n + 1)\ s\}.\ rl\ `\ s = \{0..n+1\}\ \})$ **proof**−
have $*{:}\bigwedge x\ y.\ x = y \Longrightarrow odd\ (card\ x) \Longrightarrow odd\ (card\ y)$ **by** *auto*
have $*{:}odd(card\ \{f{\in}\{f.\ \exists s{\in}\{s.\ ksimplex\ p\ (n + 1)\ s\}.\ (\exists a{\in}s.\ f = s - \{a\})\}.$
$(rl\ `\ f = \{0..n\}) \wedge$
$((\exists j{\in}\{1..n+1\}.\ \forall x{\in}f.\ x\ j = 0) \vee$
$(\exists j{\in}\{1..n+1\}.\ \forall x{\in}f.\ x\ j = p))\})$ **apply**$(rule\ *[OF\ -\ assms(2)])$ **by** *auto*
show *?thesis* **apply**$(rule\ kuhn\text{-}complete\text{-}lemma[OF\ finite\text{-}simplices])$ **prefer** *6* **apply**$(rule\ *)$ **apply**$(rule,rule,rule)$
apply$(subst\ ksimplex\text{-}def)$ **defer apply**$(rule,rule\ assms(1)[rule\text{-}format])$ **unfolding** *mem-Collect-eq* **apply** *assumption*
apply *default*+ **unfolding** *mem-Collect-eq* **apply**$(erule\ disjE\ bexE)$+ **defer apply**$(erule\ disjE\ bexE)$+ **defer**
apply *default*+ **unfolding** *mem-Collect-eq* **apply**$(erule\ disjE\ bexE)$+ **unfolding** *mem-Collect-eq* **proof**−
fix *f s a* **assume** *as*:$ksimplex\ p\ (n + 1)\ s\ a{\in}s\ f = s - \{a\}$
let $?S = \{s.\ ksimplex\ p\ (n + 1)\ s \wedge (\exists a{\in}s.\ f = s - \{a\})\}$
have $S{:}?S = \{s'.\ ksimplex\ p\ (n + 1)\ s' \wedge (\exists b{\in}s'.\ s' - \{b\} = s - \{a\})\}$ **unfolding** *as* **by** *blast*
{ fix *j* **assume** $j{:}j \in \{1..n + 1\}\ \forall x{\in}f.\ x\ j = 0$ **thus** $card\ \{s.\ ksimplex\ p\ (n + 1)\ s \wedge (\exists a{\in}s.\ f = s - \{a\})\} = 1$ **unfolding** *S*
apply−**apply**$(rule\ ksimplex\text{-}replace\text{-}0)$ **apply**$(rule\ as)$+ **unfolding** *as* **by** *auto* **}**
{ fix *j* **assume** $j{:}j \in \{1..n + 1\}\ \forall x{\in}f.\ x\ j = p$ **thus** $card\ \{s.\ ksimplex\ p\ (n + 1)\ s \wedge (\exists a{\in}s.\ f = s - \{a\})\} = 1$ **unfolding** *S*
apply−**apply**$(rule\ ksimplex\text{-}replace\text{-}1)$ **apply**$(rule\ as)$+ **unfolding** *as* **by** *auto* **}**
show $\neg\ ((\exists j{\in}\{1..n+1\}.\ \forall x{\in}f.\ x\ j = 0) \vee (\exists j{\in}\{1..n+1\}.\ \forall x{\in}f.\ x\ j = p)) \Longrightarrow card\ ?S = 2$
unfolding *S* **apply**$(rule\ ksimplex\text{-}replace\text{-}2)$ **apply**$(rule\ as)$+ **unfolding** *as* **by** *auto* **qed** *auto* **qed**

20.8 Reduced labelling.

definition *reduced label* $(n{::}nat)\ (x{::}nat{\Rightarrow}nat) =$
$(SOME\ k.\ k \leq n \wedge (\forall i.\ 1{\leq}i \wedge i{<}k{+}1 \longrightarrow label\ x\ i = 0) \wedge (k = n \vee label\ x\ (k + 1) \neq (0{::}nat)))$

lemma *reduced-labelling*: **shows** $reduced\ label\ n\ x \leq n$ (**is** *?t1*) **and**
$\forall i.\ 1{\leq}i \wedge i < reduced\ label\ n\ x + 1 \longrightarrow (label\ x\ i = 0)$ (**is** *?t2*)

(*reduced label n x = n*) ∨ (*label x* (*reduced label n x + 1*) ≠ *0*) (**is** *?t3*) **proof**−
have *num-WOP*:⋀*P k*. *P* (*k*::*nat*) ⟹ ∃ *n*. *P n* ∧ (∀ *m*<*n*. ¬ *P m*)
apply(*drule ex-has-least-nat*[**where** *m*=λ*x*. *x*]) **apply**(*erule exE*,*rule-tac x*=*x* **in** *exI*) **by** *auto*
have ∗:*n* ≤ *n* ∧ (*label x* (*n* + *1*) ≠ *0* ∨ *n* = *n*) **by** *auto*
then guess *N* **apply**(*drule-tac num-WOP*[*of* λ*j*. *j*≤*n* ∧ (*label x* (*j*+*1*) ≠ *0* ∨ *n* = *j*)]) **apply**(*erule exE*) **. note** *N*=*this*
have *N'*:*N* ≤ *n* ∀ *i*. *1* ≤ *i* ∧ *i* < *N* + *1* ⟶ *label x i* = *0 N* = *n* ∨ *label x* (*N* + *1*) ≠ *0* **defer proof**(*rule*,*rule*)
fix *i* **assume** *i*:*1*≤*i* ∧ *i*<*N*+*1* **thus** *label x i* = *0* **using** *N*[*THEN conjunct2*,*THEN spec*[**where** *x*=*i* − *1*]] **using** *N* **by** *auto* **qed**(*insert N*, *auto*)
show *?t1 ?t2 ?t3* **unfolding** *reduced-def* **apply**(*rule-tac*[!] *someI2-ex*) **using** *N'* **by**(*auto intro*!: *exI*[**where** *x*=*N*]) **qed**

lemma *reduced-labelling-unique*: **fixes** *x*::*nat* ⇒ *nat*
assumes *r* ≤ *n* ∀ *i*. *1* ≤ *i* ∧ *i* < *r* + *1* ⟶ (*label x i* = *0*) (*r* = *n*) ∨ (*label x* (*r* + *1*) ≠ *0*)
shows *reduced label n x* = *r* **apply**(*rule le-antisym*) **apply**(*rule-tac*[!] *ccontr*) **unfolding** *not-le*
using *reduced-labelling*[*of label n x*] **using** *assms* **by** *auto*

lemma *reduced-labelling-0*: **assumes** *j*∈{*1*..*n*} *label x j* = *0* **shows** *reduced label n x* ≠ *j* − *1*
using *reduced-labelling*[*of label n x*] **using** *assms* **by** *fastsimp*

lemma *reduced-labelling-1*: **assumes** *j*∈{*1*..*n*} *label x j* ≠ *0* **shows** *reduced label n x* < *j*
using *assms* **and** *reduced-labelling*[*of label n x*] **apply**(*erule-tac x*=*j* **in** *allE*) **by** *auto*

lemma *reduced-labelling-Suc*:
assumes *reduced lab* (*n* + *1*) *x* ≠ *n* + *1* **shows** *reduced lab* (*n* + *1*) *x* = *reduced lab n x*
apply(*subst eq-commute*) **apply**(*rule reduced-labelling-unique*)
using *reduced-labelling*[*of lab n*+*1 x*] **and** *assms* **by** *auto*

lemma *complete-face-top*:
assumes ∀ *x*∈*f*. ∀ *j*∈{*1*..*n*+*1*}. *x j* = *0* ⟶ *lab x j* = *0*
∀ *x*∈*f*. ∀ *j*∈{*1*..*n*+*1*}. *x j* = *p* ⟶ *lab x j* = *1*
shows ((*reduced lab* (*n* + *1*)) ‘ *f* = {*0*..*n*}) ∧ ((∃ *j*∈{*1*..*n*+*1*}. ∀ *x*∈*f*. *x j* = *0*) ∨ (∃ *j*∈{*1*..*n*+*1*}. ∀ *x*∈*f*. *x j* = *p*)) ⟷
((*reduced lab* (*n* + *1*)) ‘ *f* = {*0*..*n*}) ∧ (∀ *x*∈*f*. *x* (*n* + *1*) = *p*) (**is** *?l* = *?r*)
proof
assume *?l* (**is** *?as* ∧ (*?a* ∨ *?b*)) **thus** *?r* **apply**−**apply**(*rule*,*erule conjE*,*assumption*) **proof**(*cases ?a*)
case *True* **then guess** *j* **.. note** *j*=*this* {**fix** *x* **assume** *x*:*x*∈*f*
have *reduced lab* (*n*+*1*) *x* ≠ *j* − *1* **using** *j* **apply**−**apply**(*rule reduced-labelling-0*) **defer apply**(*rule assms*(*1*)[*rule-format*]) **using** *x* **by** *auto* }
moreover have *j* − *1* ∈ {*0*..*n*} **using** *j* **by** *auto*

then guess y **unfolding** ‹*?l*›[*THEN conjunct1*,*THEN sym*] **and** *image-iff* **..**
note $y = this$
ultimately have *False* **by** *auto* **thus** $\forall x{\in}f.\ x\ (n + 1) = p$ **by** *auto* **next**

case *False* **hence** *?b* **using** ‹*?l*› **by** *blast* **then guess** j **.. note** $j{=}this$ **{fix** x
assume $x{:}x{\in}f$
have *reduced lab* $(n{+}1)\ x < j$ **using** j **apply−apply**(*rule reduced-labelling-1*)
using *assms*(*2*)[*rule-format*,*of x j*] **and** x **by** *auto* **}** **note** $* = this$
have $j = n + 1$ **proof**(*rule ccontr*) **case** *goal1* **hence** $j < n + 1$ **using** j **by**
auto **moreover**
have $n \in \{0..n\}$ **by** *auto* **then guess** y **unfolding** ‹*?l*›[*THEN conjunct1*,*THEN*
sym] *image-iff* **..**
ultimately show *False* **using** $*$[*of y*] **by** *auto* **qed**
thus $\forall x{\in}f.\ x\ (n + 1) = p$ **using** j **by** *auto* **qed qed**(*auto*)

20.9 Hence we get just about the nice induction.

lemma *kuhn-induction*:
assumes $0 < p\ \forall x.\ \forall j{\in}\{1..n{+}1\}.\ (\forall j.\ x\ j \le p) \wedge (x\ j = 0) \longrightarrow (lab\ x\ j = 0)$
$\forall x.\ \forall j{\in}\{1..n{+}1\}.\ (\forall j.\ x\ j \le p) \wedge (x\ j = p) \longrightarrow (lab\ x\ j = 1)$
$odd\ (card\ \{f.\ ksimplex\ p\ n\ f \wedge ((reduced\ lab\ n)\ `\ f = \{0..n\})\})$
shows $odd\ (card\ \{s.\ ksimplex\ p\ (n{+}1)\ s \wedge((reduced\ lab\ (n{+}1))\ `\ s = \{0..n{+}1\})\})$
proof−
have $*{:}\bigwedge s\ t.\ odd\ (card\ s) \Longrightarrow s = t \Longrightarrow odd\ (card\ t)\ \bigwedge s\ f.\ (\bigwedge x.\ f\ x \le n + 1\ )$
$\Longrightarrow f\ `\ s \subseteq \{0..n{+}1\}$ **by** *auto*
show *?thesis* **apply**(*rule kuhn-simplex-lemma*[*unfolded mem-Collect-eq*]) **ap-**
ply(*rule*,*rule*,*rule* $*$,*rule reduced-labelling*)
apply(*rule* $*$(*1*)[*OF assms*(*4*)]) **apply**(*rule set-ext*) **unfolding** *mem-Collect-eq*
apply(*rule*,*erule conjE*) **defer apply**(*rule*) **proof−**
fix f **assume** *as*:*ksimplex p n f reduced lab n* ‘ $f = \{0..n\}$
have $*{:}\forall x{\in}f.\ \forall j{\in}\{1..n + 1\}.\ x\ j = 0 \longrightarrow lab\ x\ j = 0\ \forall x{\in}f.\ \forall j{\in}\{1..n + 1\}.$
$x\ j = p \longrightarrow lab\ x\ j = 1$
using *assms*(*2−3*) **using** *as*(*1*)[*unfolded ksimplex-def*] **by** *auto*
have *allp*:$\forall x{\in}f.\ x\ (n + 1) = p$ **using** *assms*(*2*) **using** *as*(*1*)[*unfolded ksimplex-def*]
by *auto*
{ fix x **assume** $x{\in}f$ **hence** *reduced lab* $(n + 1)\ x < n + 1$ **apply−apply**(*rule*
reduced-labelling-1)
defer using *assms*(*3*) **using** *as*(*1*)[*unfolded ksimplex-def*] **by** *auto*
hence *reduced lab* $(n + 1)\ x =$ *reduced lab n x* **apply−apply**(*rule reduced-labelling-Suc*)
using *reduced-labelling*(*1*) **by** *auto* **}**
hence *reduced lab* $(n + 1)$ ‘ $f = \{0..n\}$ **unfolding** *as*(*2*)[*THEN sym*] **apply−**
apply(*rule set-ext*) **unfolding** *image-iff* **by** *auto*
moreover guess s **using** *as*(*1*)[*unfolded simplex-top-face*[*OF assms*(*1*) *allp*,*THEN*
sym]] **.. then guess** a **..**
ultimately show $\exists s\ a.\ ksimplex\ p\ (n + 1)\ s \wedge$
$a \in s \wedge f = s - \{a\} \wedge$ *reduced lab* $(n + 1)$ ‘ $f = \{0..n\} \wedge ((\exists j{\in}\{1..n + 1\}.\ \forall x{\in}f.\ x\ j = 0) \vee (\exists j{\in}\{1..n + 1\}.\ \forall x{\in}f.\ x\ j = p))$ (**is** *?ex*)
apply(*rule-tac* $x{=}s$ **in** *exI*,*rule-tac* $x{=}a$ **in** *exI*) **unfolding** *complete-face-top*[*OF*
$*$] **using** *allp as*(*1*) **by** *auto*

next fix *f* **assume** *as*:$\exists\, s\ a.$ *ksimplex p* $(n + 1)$ *s* $\wedge$
$a \in s \wedge f = s - \{a\} \wedge$ *reduced lab* $(n + 1)$ ‘ $f = \{0..n\} \wedge ((\exists j{\in}\{1..n + 1\}.\ \forall x{\in}f.\ x\ j = 0) \vee (\exists j{\in}\{1..n + 1\}.\ \forall x{\in}f.\ x\ j = p))$ (**is** *?ex*)
then guess *s* **.. then guess** *a* **apply**−**apply**(*erule exE*,(*erule conjE*)+) **. note** *sa*=*this*
{ fix *x* **assume** *x*∈*f* **hence** *reduced lab* $(n + 1)$ *x* $\in$ *reduced lab* $(n + 1)$ ‘ *f* **by** *auto*
hence *reduced lab* $(n + 1)$ *x* $<$ *n* + *1* **using** *sa*(*4*) **by** *auto*
hence *reduced lab* $(n + 1)$ *x* = *reduced lab n x* **apply**−**apply**(*rule reduced-labelling-Suc*) **using** *reduced-labelling*(*1*) **by** *auto* **}**
thus *part1*:*reduced lab n* ‘ *f* = {*0..n*} **unfolding** *sa*(*4*)[*THEN sym*] **apply**−**apply**(*rule set-ext*) **unfolding** *image-iff* **by** *auto*
have ∗:$\forall\, x{\in}f.\ x\ (n + 1) = p$ **proof**(*cases* $\exists j{\in}\{1..n + 1\}.\ \forall x{\in}f.\ x\ j = 0$)
case *True* **then guess** *j* **.. hence** $\bigwedge x.\ x{\in}f \Longrightarrow$ *reduced lab* $(n + 1)$ *x* $\neq j - 1$ **apply**−**apply**(*rule reduced-labelling-0*) **apply** *assumption*
apply(*rule assms*(*2*)[*rule-format*]) **using** *sa*(*1*)[*unfolded ksimplex-def*] **unfolding** *sa* **by** *auto* **moreover**
have $j - 1 \in \{0..n\}$ **using** ‹*j*∈{*1..n*+*1*}› **by** *auto*
ultimately have *False* **unfolding** *sa*(*4*)[*THEN sym*] **unfolding** *image-iff* **by** *fastsimp* **thus** *?thesis* **by** *auto* **next**
case *False* **hence** $\exists j{\in}\{1..n + 1\}.\ \forall x{\in}f.\ x\ j = p$ **using** *sa*(*5*) **by** *fastsimp* **then guess** *j* **.. note** *j*=*this*
thus *?thesis* **proof**(*cases j* = *n*+*1*)
case *False* **hence** ∗:*j*∈{*1..n*} **using** *j* **by** *auto*
hence $\bigwedge x.\ x{\in}f \Longrightarrow$ *reduced lab n x* $<$ *j* **apply**(*rule reduced-labelling-1*) **proof**− **fix** *x* **assume** *x*∈*f*
hence *lab x j* = *1* **apply**−**apply**(*rule assms*(*3*)[*rule-format*,*OF j*(*1*)])
using *sa*(*1*)[*unfolded ksimplex-def*] **using** *j* **unfolding** *sa* **by** *auto* **thus** *lab x j* $\neq$ *0* **by** *auto* **qed**
moreover have *j*∈{*0..n*} **using** ∗ **by** *auto*
ultimately have *False* **unfolding** *part1*[*THEN sym*] **using** ∗ **unfolding** *image-iff* **by** *auto* **thus** *?thesis* **by** *auto* **qed** *auto* **qed**
thus *ksimplex p n f* **using** *as* **unfolding** *simplex-top-face*[*OF assms*(*1*) ∗,*THEN sym*] **by** *auto* **qed qed**

lemma *kuhn-induction-Suc*:
assumes *0* $<$ *p* $\forall\, x.\ \forall j{\in}\{1..Suc\ n\}.\ (\forall j.\ x\ j \leq p) \wedge (x\ j = 0) \longrightarrow (lab\ x\ j = 0)$
$\forall\, x.\ \forall j{\in}\{1..Suc\ n\}.\ (\forall j.\ x\ j \leq p) \wedge (x\ j = p) \longrightarrow (lab\ x\ j = 1)$
odd (*card* {*f*. *ksimplex p n f* $\wedge$ ((*reduced lab n*) ‘ *f* = {*0..n*})})
shows *odd* (*card* {*s*. *ksimplex p* (*Suc n*) *s* $\wedge$((*reduced lab* (*Suc n*)) ‘ *s* = {*0..Suc n*})})
using *assms* **unfolding** *Suc-eq-plus1* **by**(*rule kuhn-induction*)

20.10 And so we get the final combinatorial result.

lemma *ksimplex-0*: *ksimplex p 0 s* $\longleftrightarrow$ *s* = {($\lambda x.$ *p*)} (**is** *?l* = *?r*) **proof**
assume *l*:*?l* **guess** *a* **using** *ksimplexD*(*3*)[*OF l*, *unfolded add-0*] **unfolding** *card-1-exists* **.. note** *a*=*this*

have $a = (\lambda x.\ p)$ **using** *ksimplexD*(*5*)[*OF l*, *rule-format*, *OF a*(*1*)] **by**(*rule*,*auto*)
thus *?r* **using** *a* **by** *auto* **next**
assume *r*:*?r* **show** *?l* **unfolding** *r ksimplex-eq* **by** *auto* **qed**

lemma *reduce-labelling-0*[*simp*]: *reduced lab 0 x = 0* **apply**(*rule reduced-labelling-unique*)
by *auto*

lemma *kuhn-combinatorial*:
assumes $0 < p\ \forall x\ j.\ (\forall j.\ x(j) \leq p) \wedge 1 \leq j \wedge j \leq n \wedge (x\ j = 0) \longrightarrow (lab\ x\ j = 0)$
$\forall x\ j.\ (\forall j.\ x(j) \leq p) \wedge 1 \leq j \wedge j \leq n\ \wedge (x\ j = p) \longrightarrow (lab\ x\ j = 1)$
shows *odd* (*card* {*s*. *ksimplex p n s* ∧ ((*reduced lab n*) ' *s* = {*0*..*n*})}) **using** *assms* **proof**(*induct n*)
let *?M* = λ*n*. {*s*. *ksimplex p n s* ∧ ((*reduced lab n*) ' *s* = {*0*..*n*})}
{ case *0* **have** ∗:*?M 0* = {{(λ*x*. *p*)}} **unfolding** *ksimplex-0* **by** *auto* **show** *?case* **unfolding** ∗ **by** *auto* **}**
case (*Suc n*) **have** *odd* (*card* (*?M n*)) **apply**(*rule Suc*(*1*)[*OF Suc*(*2*)]) **using** *Suc*(*3*−) **by** *auto*
thus *?case* **apply**−**apply**(*rule kuhn-induction-Suc*) **using** *Suc*(*2*−) **by** *auto* **qed**

lemma *kuhn-lemma*: **assumes** *0* < (*p*::*nat*) *0* < (*n*::*nat*)
∀*x*. (∀*i*∈{*1*..*n*}. *x i* ≤ *p*) ⟶ (∀*i*∈{*1*..*n*}. (*label x i* = (*0*::*nat*)) ∨ (*label x i* = *1*))
∀*x*. (∀*i*∈{*1*..*n*}. *x i* ≤ *p*) ⟶ (∀*i*∈{*1*..*n*}. (*x i* = *0*) ⟶ (*label x i* = *0*))
∀*x*. (∀*i*∈{*1*..*n*}. *x i* ≤ *p*) ⟶ (∀*i*∈{*1*..*n*}. (*x i* = *p*) ⟶ (*label x i* = *1*))
obtains *q* **where** ∀*i*∈{*1*..*n*}. *q i* < *p*
∀*i*∈{*1*..*n*}. ∃*r s*. (∀*j*∈{*1*..*n*}. *q*(*j*) ≤ *r*(*j*) ∧ *r*(*j*) ≤ *q*(*j*) + *1*) ∧
(∀*j*∈{*1*..*n*}. *q*(*j*) ≤ *s*(*j*) ∧ *s*(*j*) ≤ *q*(*j*) + *1*) ∧
~(*label r i* = *label s i*) **proof**−
let *?A* = {*s*. *ksimplex p n s* ∧ *reduced label n* ' *s* = {*0*..*n*}} **have** *n*≠*0* **using** *assms* **by** *auto*
have *conjD*:⋀*P Q*. *P* ∧ *Q* ⟹ *P* ⋀*P Q*. *P* ∧ *Q* ⟹ *Q* **by** *auto*
have *odd* (*card ?A*) **apply**(*rule kuhn-combinatorial*[*of p n label*]) **using** *assms* **by** *auto*
hence *card ?A* ≠ *0* **apply**−**apply**(*rule ccontr*) **by** *auto* **hence** *?A* ≠ {} **unfolding** *card-eq-0-iff* **by** *auto*
then obtain *s* **where** *s*∈*?A* **by** *auto* **note** *s*=*conjD*[*OF this*[*unfolded mem-Collect-eq*]]
guess *a b* **apply**(*rule ksimplex-extrema-strong*[*OF s*(*1*) ‹*n*≠*0*›]) **. note** *ab*=*this*
show *?thesis* **apply**(*rule that*[*of a*]) **proof**(*rule-tac*[!] *ballI*) **fix** *i* **assume** *i*∈{*1*..*n*}
hence *a i* + *1* ≤ *p* **apply**−**apply**(*rule order-trans*[*of* - *b i*]) **apply**(*subst ab*(*5*)[*THEN spec*[**where** *x*=*i*]])
using *s*(*1*)[*unfolded ksimplex-def*] **defer apply**− **apply**(*erule conjE*)+ **apply**(*drule-tac bspec*[*OF* - *ab*(*2*)])+ **by** *auto*
thus *a i* < *p* **by** *auto*
case *goal2* **hence** *i* ∈ *reduced label n* ' *s* **using** *s* **by** *auto* **then guess** *u* **unfolding** *image-iff* **.. note** *u*=*this*
from *goal2* **have** *i* − *1* ∈ *reduced label n* ' *s* **using** *s* **by** *auto* **then guess** *v* **unfolding** *image-iff* **.. note** *v*=*this*
show *?case* **apply**(*rule-tac x*=*u* **in** *exI*, *rule-tac x*=*v* **in** *exI*) **apply**(*rule conjI*)

defer apply(*rule conjI*) **defer** *2* **proof**(*rule-tac*[*1−2*] *ballI*)
show *label u i* ≠ *label v i* **using** *reduced-labelling*[*of label n u*] *reduced-labelling*[*of label n v*]
unfolding *u*(*2*)[*THEN sym*] *v*(*2*)[*THEN sym*] **using** *goal2* **by** *auto*
fix *j* **assume** *j*:*j*∈{*1..n*} **show** *a j* ≤ *u j* ∧ *u j* ≤ *a j* + *1 a j* ≤ *v j* ∧ *v j* ≤ *a j* + *1*
using *conjD*[*OF ab*(*4*)[*rule-format*, *OF u*(*1*)]] **and** *conjD*[*OF ab*(*4*)[*rule-format*, *OF v*(*1*)]] **apply**−
apply(*drule-tac*[!] *kle-imp-pointwise*)+ **apply**(*erule-tac*[!] *x*=*j* **in** *allE*)+ **unfolding** *ab*(*5*)[*rule-format*] **using** *j*
by *auto* **qed qed qed**

20.11 The main result for the unit cube.

lemma *kuhn-labelling-lemma′*:
assumes (∀ *x*::*nat*⇒*real*. *P x* ⟶ *P* (*f x*)) ∀ *x*. *P x* ⟶ (∀ *i*::*nat*. *Q i* ⟶ *0* ≤ *x i* ∧ *x i* ≤ *1*)
shows ∃ *l*. (∀ *x i*. *l x i* ≤ (*1*::*nat*)) ∧
(∀ *x i*. *P x* ∧ *Q i* ∧ (*x i* = *0*) ⟶ (*l x i* = *0*)) ∧
(∀ *x i*. *P x* ∧ *Q i* ∧ (*x i* = *1*) ⟶ (*l x i* = *1*)) ∧
(∀ *x i*. *P x* ∧ *Q i* ∧ (*l x i* = *0*) ⟶ *x i* ≤ *f*(*x*) *i*) ∧
(∀ *x i*. *P x* ∧ *Q i* ∧ (*l x i* = *1*) ⟶ *f*(*x*) *i* ≤ *x i*) **proof**−
have *and-forall-thm*:⋀*P Q*. (∀ *x*. *P x*) ∧ (∀ *x*. *Q x*) ⟷ (∀ *x*. *P x* ∧ *Q x*) **by** *auto*
have ∗:∀ *x y*::*real*. *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1* ⟶ (*x* ≠ *1* ∧ *x* ≤ *y* ∨ *x* ≠ *0* ∧ *y* ≤ *x*) **by** *auto*
show *?thesis* **unfolding** *and-forall-thm* **apply**(*subst choice-iff*[*THEN sym*])+ **proof**(*rule*,*rule*) **case** *goal1*
let *?R* = λ*y*. (*P x* ∧ *Q xa* ∧ *x xa* = *0* ⟶ *y* = (*0*::*nat*)) ∧
(*P x* ∧ *Q xa* ∧ *x xa* = *1* ⟶ *y* = *1*) ∧ (*P x* ∧ *Q xa* ∧ *y* = *0* ⟶ *x xa* ≤ (*f x*) *xa*) ∧ (*P x* ∧ *Q xa* ∧ *y* = *1* ⟶ (*f x*) *xa* ≤ *x xa*)
{ **assume** *P x Q xa* **hence** *0* ≤ (*f x*) *xa* ∧ (*f x*) *xa* ≤ *1* **using** *assms*(*2*)[*rule-format*,*of f x xa*]
apply(*drule-tac assms*(*1*)[*rule-format*]) **by** *auto* }
hence *?R 0* ∨ *?R 1* **by** *auto* **thus** *?case* **by** *auto* **qed qed**

lemma *brouwer-cube*: **fixes** *f*::*real*^′*n* ⇒ *real*^′*n*
assumes *continuous-on* {*0..1*} *f f* ‘ {*0..1*} ⊆ {*0..1*}
shows ∃ *x*∈{*0..1*}. *f x* = *x* **apply**(*rule ccontr*) **proof**−
def *n* ≡ *CARD*(′*n*) **have** *n*:*1* ≤ *n 0* < *n n* ≠ *0* **unfolding** *n-def* **by** *auto*
assume ¬ (∃ *x*∈{*0..1*}. *f x* = *x*) **hence** ∗:¬ (∃ *x*∈{*0..1*}. *f x* − *x* = *0*) **by** *auto*
guess *d* **apply**(*rule brouwer-compactness-lemma*[*OF compact-interval* - ∗])
apply(*rule continuous-on-intros assms*)+ **. note** *d*=*this*[*rule-format*]
have ∗:∀ *x*. *x* ∈ {*0..1*} ⟶ *f x* ∈ {*0..1*} ∀ *x*. *x* ∈ {*0..1*::*real*^′*n*} ⟶ (∀ *i*. *True* ⟶ *0* ≤ *x* $ *i* ∧ *x* $ *i* ≤ *1*)
using *assms*(*2*)[*unfolded image-subset-iff Ball-def*] **unfolding** *mem-interval* **by** *auto*
guess *label* **using** *kuhn-labelling-lemma*[*OF* ∗] **apply**−**apply**(*erule exE*,(*erule conjE*)+) **. note** *label* = *this*[*rule-format*]

have *lem1*:$\forall x{\in}\{0..1\}.\forall y{\in}\{0..1\}.\forall i.\ label\ x\ i \neq label\ y\ i$
$\longrightarrow abs(f\ x\ \$\ i - x\ \$\ i) \leq norm(f\ y - f\ x) + norm(y - x)$
proof(*rule*,*rule*,*rule*,*rule*)
fix *x y* **assume** *xy*:$x{\in}\{0..1::real\hat{}'n\}\ y{\in}\{0..1::real\hat{}'n\}$ **fix** *i*::'*n* **assume** *i*:*label x i* $\neq$ *label y i*
have $*$:$\bigwedge x\ y\ fx\ fy{::}real.\ (x \leq fx \wedge fy \leq y \vee fx \leq x \wedge y \leq fy)$
$\Longrightarrow abs(fx - x) \leq abs(fy - fx) + abs(y - x)$ **by** *auto*
have $|(f\ x - x)\ \$\ i| \leq abs((f\ y - f\ x)\$i) + abs((y - x)\$i)$ **unfolding** *vector-minus-component*
apply(*rule* $*$) **apply**(*cases label x i = 0*) **apply**(*rule disjI1*,*rule*) **prefer** *3*
proof(*rule disjI2*,*rule*)
assume *lx*:*label x i = 0* **hence** *ly*:*label y i = 1* **using** *i label*(*1*)[*of y i*] **by** *auto*
show *x* $\$$ *i* $\leq$ *f x* $\$$ *i* **apply**(*rule label*(*4*)[*rule-format*]) **using** *xy lx* **by** *auto*
show *f y* $\$$ *i* $\leq$ *y* $\$$ *i* **apply**(*rule label*(*5*)[*rule-format*]) **using** *xy ly* **by** *auto*
next
assume *label x i* $\neq$ *0* **hence** *l*:*label x i = 1 label y i = 0*
using *i label*(*1*)[*of x i*] *label*(*1*)[*of y i*] **by** *auto*
show *f x* $\$$ *i* $\leq$ *x* $\$$ *i* **apply**(*rule label*(*5*)[*rule-format*]) **using** *xy l* **by** *auto*
show *y* $\$$ *i* $\leq$ *f y* $\$$ *i* **apply**(*rule label*(*4*)[*rule-format*]) **using** *xy l* **by** *auto*
qed
also have $\ldots \leq norm\ (f\ y - f\ x) + norm\ (y - x)$ **apply**(*rule add-mono*)
by(*rule component-le-norm*)+
finally show $|f\ x\ \$\ i - x\ \$\ i| \leq norm\ (f\ y - f\ x) + norm\ (y - x)$ **unfolding** *vector-minus-component* **. qed**
have $\exists e{>}0.\ \forall x{\in}\{0..1\}.\ \forall y{\in}\{0..1\}.\ \forall z{\in}\{0..1\}.\ \forall i.$
$norm(x - z) < e \wedge norm(y - z) < e \wedge label\ x\ i \neq label\ y\ i \longrightarrow abs((f(z) - z)\$i) < d\ /\ (real\ n)$ **proof**−
have *d'*:*d / real n / 8 > 0* **apply**(*rule divide-pos-pos*)+ **using** *d*(*1*) **unfolding** *n-def* **by** *auto*
have $*$:*uniformly-continuous-on* $\{0..1\}$ *f* **by**(*rule compact-uniformly-continuous*[*OF assms*(*1*) *compact-interval*])
guess *e* **using** $*$[*unfolded uniformly-continuous-on-def*,*rule-format*,*OF d'*] **apply**−**apply**(*erule exE*,(*erule conjE*)+) **.**
note *e*=*this*[*rule-format*,*unfolded dist-norm*]
show *?thesis* **apply**(*rule-tac x*=*min* (*e*/*2*) (*d*/*real n*/*8*) **in** *exI*) **apply**(*rule*)
defer
apply(*rule*,*rule*,*rule*,*rule*,*rule*) **apply**(*erule conjE*)+ **proof**−
show *0 < min* (*e / 2*) (*d / real n / 8*) **using** *d' e* **by** *auto*
fix *x y z i* **assume** *as*:$x \in \{0..1\}\ y \in \{0..1\}\ z \in \{0..1\}$ *norm* $(x - z) <$ *min* (*e / 2*) (*d / real n / 8*)
norm $(y - z) <$ *min* (*e / 2*) (*d / real n / 8*) *label x i* $\neq$ *label y i*
have $*$:$\bigwedge z\ fz\ x\ fx\ n1\ n2\ n3\ n4\ d4\ d{::}real.\ abs(fx - x) \leq n1 + n2 \Longrightarrow abs(fx - fz) \leq n3 \Longrightarrow abs(x - z) \leq n4 \Longrightarrow$
$n1 < d4 \Longrightarrow n2 < 2 * d4 \Longrightarrow n3 < d4 \Longrightarrow n4 < d4 \Longrightarrow (8 * d4 = d) \Longrightarrow abs(fz - z) < d$ **by** *auto*
show $|(f\ z - z)\ \$\ i| < d\ /\ real\ n$ **unfolding** *vector-minus-component*
proof(*rule* $*$)
show $|f\ x\ \$\ i - x\ \$\ i| \leq norm\ (f\ y - f\ x) + norm\ (y - x)$ **apply**(*rule*

lem1[*rule-format*]) **using** *as* **by** *auto*
show $|f\,x\,\$\,i - f\,z\,\$\,i| \leq norm\ (f\,x - f\,z)\ |x\,\$\,i - z\,\$\,i| \leq norm\ (x - z)$
unfolding *vector-minus-component*[*THEN sym*] **by**(*rule component-le-norm*)+
have *tria*:$norm\ (y - x) \leq norm\ (y - z) + norm\ (x - z)$ **using** *dist-triangle*[*of y x z*,*unfolded dist-norm*]
unfolding *norm-minus-commute* **by** *auto*
also have $\ldots < e\ /\ 2 + e\ /\ 2$ **apply**(*rule add-strict-mono*) **using** *as*(*4*,*5*) **by** *auto*
finally show $norm\ (f\,y - f\,x) < d\ /\ real\ n\ /\ 8$ **apply**− **apply**(*rule e*(*2*)) **using** *as* **by** *auto*
have $norm\ (y - z) + norm\ (x - z) < d\ /\ real\ n\ /\ 8 + d\ /\ real\ n\ /\ 8$ **apply**(*rule add-strict-mono*) **using** *as* **by** *auto*
thus $norm\ (y - x) < 2 * (d\ /\ real\ n\ /\ 8)$ **using** *tria* **by** *auto*
show $norm\ (f\,x - f\,z) < d\ /\ real\ n\ /\ 8$ **apply**(*rule e*(*2*)) **using** *as e*(*1*) **by** *auto* **qed**(*insert as*, *auto*) **qed qed**
then guess *e* **apply**−**apply**(*erule exE*,(*erule conjE*)+) **. note** *e*=*this*[*rule-format*]

guess *p* **using** *real-arch-simple*[*of 1 + real n / e*] **.. note** *p*=*this*
have $1 + real\,n\ /\ e > 0$ **apply**(*rule add-pos-pos*) **defer apply**(*rule divide-pos-pos*) **using** *e*(*1*) *n* **by** *auto*
hence $p > 0$ **using** *p* **by** *auto*
guess *b* **using** *ex-bij-betw-nat-finite-1*[*OF finite-UNIV*[**where** *'a*=*'n*]] **.. note** *b*=*this*
def $b' \equiv inv\text{-}into\ \{1..n\}\ b$
have *b'*:*bij-betw b' UNIV* $\{1..n\}$ **using** *bij-betw-inv-into*[*OF b*] **unfolding** *b'-def n-def* **by** *auto*
have *bb'*[*simp*]:$\bigwedge i.\ b\ (b'\ i) = i$ **unfolding** *b'-def* **apply**(*rule f-inv-into-f*) **unfolding** *n-def* **using** *b*
unfolding *bij-betw-def* **by** *auto*
have *b'b*[*simp*]:$\bigwedge i.\ i \in \{1..n\} \Longrightarrow b'\ (b\ i) = i$ **unfolding** *b'-def* **apply**(*rule inv-into-f-eq*)
using *b* **unfolding** *n-def bij-betw-def* **by** *auto*
have $*$:$\bigwedge x$::*nat*. $x=0 \vee x=1 \longleftrightarrow x \leq 1$ **by** *auto*
have *q1*:$0 < p\ 0 < n\ \ \forall x.\ (\forall i \in \{1..n\}.\ x\ i \leq p) \longrightarrow$
$(\forall i \in \{1..n\}.\ (label\ (\chi\ i.\ real\ (x\ (b'\ i))\ /\ real\ p) \circ b)\ i = 0 \vee (label\ (\chi\ i.\ real\ (x\ (b'\ i))\ /\ real\ p) \circ b)\ i = 1)$
unfolding $*$ **using** ‹*p*>*0*› ‹*n*>*0*› **using** *label*(*1*) **by** *auto*
have *q2*:$\forall x.\ (\forall i \in \{1..n\}.\ x\ i \leq p) \longrightarrow (\forall i \in \{1..n\}.\ x\ i = 0 \longrightarrow (label\ (\chi\ i.\ real\ (x\ (b'\ i))\ /\ real\ p) \circ b)\ i = 0)$
$\forall x.\ (\forall i \in \{1..n\}.\ x\ i \leq p) \longrightarrow (\forall i \in \{1..n\}.\ x\ i = p \longrightarrow (label\ (\chi\ i.\ real\ (x\ (b'\ i))\ /\ real\ p) \circ b)\ i = 1)$
apply(*rule*,*rule*,*rule*,*rule*) **defer proof**(*rule*,*rule*,*rule*,*rule*) **fix** *x i*
assume *as*:$\forall i \in \{1..n\}.\ x\ i \leq p\ i \in \{1..n\}$
{ assume $x\ i = p \vee x\ i = 0$
have $(\chi\ i.\ real\ (x\ (b'\ i))\ /\ real\ p) \in \{0..1\}$ **unfolding** *mem-interval Cart-lambda-beta* **proof**(*rule*,*rule*)
fix *j*::*'n* **have** *j*:$b'\ j \in \{1..n\}$ **using** *b'* **unfolding** *n-def bij-betw-def* **by** *auto*
show $0\ \$\ j \leq real\ (x\ (b'\ j))\ /\ real\ p$ **unfolding** *zero-index*

apply(*rule divide-nonneg-pos*) **using** ‹*p*>*0*› **using** *as*(*1*)[*rule-format*,*OF j*] **by** *auto*
show *real* (*x* (*b' j*)) / *real p* ≤ *1* $ *j* **unfolding** *one-index divide-le-eq-1*
using *as*(*1*)[*rule-format*,*OF j*] **by** *auto* **qed** } **note** *cube*=*this*
{ **assume** *x i* = *p* **thus** (*label* (χ *i*. *real* (*x* (*b' i*)) / *real p*) ∘ *b*) *i* = *1* **unfolding** *o-def*
apply−**apply**(*rule label*(*3*)) **using** *cube* **using** *as* ‹*p*>*0*› **by** *auto* }
{ **assume** *x i* = *0* **thus** (*label* (χ *i*. *real* (*x* (*b' i*)) / *real p*) ∘ *b*) *i* = *0* **unfolding** *o-def*
apply−**apply**(*rule label*(*2*)) **using** *cube* **using** *as* ‹*p*>*0*› **by** *auto* } **qed**
guess *q* **apply**(*rule kuhn-lemma*[*OF q1 q2*]) **. note** *q*=*this*
def *z* ≡ χ *i*. *real* (*q* (*b' i*)) / *real p*
have ∃ *i*. *d* / *real n* ≤ *abs*((*f z* − *z*)$*i*) **proof**(*rule ccontr*)
have ∀ *i*. *q* (*b' i*) ∈ {*0*..<*p*} **using** *q*(*1*) *b'*[*unfolded bij-betw-def*] **by** *auto*
hence ∀ *i*. *q* (*b' i*) ∈ {*0*..*p*} **apply**−**apply**(*rule*,*erule-tac x*=*i* **in** *allE*) **by** *auto*
hence *z*∈{*0*..*1*} **unfolding** *z-def mem-interval* **unfolding** *one-index zero-index Cart-lambda-beta*
apply−**apply**(*rule*,*rule*) **apply**(*rule divide-nonneg-pos*) **using** ‹*p*>*0*› **unfolding** *divide-le-eq-1* **by** *auto*
hence *d-fz-z*:*d* ≤ *norm* (*f z* − *z*) **apply**(*drule-tac d*) **.**
case *goal1* **hence** *as*:∀ *i*. |*f z* $ *i* − *z* $ *i*| < *d* / *real n* **using** ‹*n*>*0*› **by**(*auto simp add*:*not-le*)
have *norm* (*f z* − *z*) ≤ (∑ *i*∈*UNIV*. |*f z* $ *i* − *z* $ *i*|) **unfolding** *vector-minus-component*[*THEN sym*] **by**(*rule norm-le-l1*)
also have ... < (∑ (*i*::'*n*)∈*UNIV*. *d* / *real n*) **apply**(*rule setsum-strict-mono*) **using** *as* **by** *auto*
also have ... = *d* **unfolding** *real-eq-of-nat n-def* **using** *n* **by** *auto*
finally show *False* **using** *d-fz-z* **by** *auto* **qed then guess** *i* **.. note** *i*=*this*
have ∗:*b' i* ∈ {*1*..*n*} **using** *b'*[*unfolded bij-betw-def*] **by** *auto*
guess *r* **using** *q*(*2*)[*rule-format*,*OF* ∗] **.. then guess** *s* **apply**−**apply**(*erule exE*,(*erule conjE*)+) **. note** *rs*=*this*[*rule-format*]
have *b'-im*:⋀*i*. *b' i* ∈ {*1*..*n*} **using** *b'* **unfolding** *bij-betw-def* **by** *auto*
def *r'* ≡ χ *i*. *real* (*r* (*b' i*)) / *real p*
have ⋀*i*. *r* (*b' i*) ≤ *p* **apply**(*rule order-trans*) **apply**(*rule rs*(*1*)[*OF b'-im*,*THEN conjunct2*])
using *q*(*1*)[*rule-format*,*OF b'-im*] **by**(*auto simp add*: *Suc-le-eq*)
hence *r'* ∈ {*0*..*1*::*real*^'*n*} **unfolding** *r'-def mem-interval Cart-lambda-beta one-index zero-index*
apply−**apply**(*rule*,*rule*,*rule divide-nonneg-pos*)
using *rs*(*1*)[*OF b'-im*] *q*(*1*)[*rule-format*,*OF b'-im*] ‹*p*>*0*› **by** *auto*
def *s'* ≡ χ *i*. *real* (*s* (*b' i*)) / *real p*
have ⋀*i*. *s* (*b' i*) ≤ *p* **apply**(*rule order-trans*) **apply**(*rule rs*(*2*)[*OF b'-im*,*THEN conjunct2*])
using *q*(*1*)[*rule-format*,*OF b'-im*] **by**(*auto simp add*: *Suc-le-eq*)
hence *s'* ∈ {*0*..*1*::*real*^'*n*} **unfolding** *s'-def mem-interval Cart-lambda-beta one-index zero-index*
apply−**apply**(*rule*,*rule*,*rule divide-nonneg-pos*)
using *rs*(*1*)[*OF b'-im*] *q*(*1*)[*rule-format*,*OF b'-im*] ‹*p*>*0*› **by** *auto*
have *z*∈{*0*..*1*} **unfolding** *z-def mem-interval Cart-lambda-beta one-index zero-index*

apply(*rule*,*rule*,*rule divide-nonneg-pos*) **using** *q*(*1*)[*rule-format*,*OF b'-im*] ‹*p*>*0*›
by(*auto intro*:*less-imp-le*)
have ∗:⋀*x*. *1* + *real x* = *real* (*Suc x*) **by** *auto*
{ have (∑ *i*∈*UNIV*. |*real* (*r* (*b'* *i*)) − *real* (*q* (*b'* *i*))|) ≤ (∑ (*i*::'*n*)∈*UNIV*. *1*)
apply(*rule setsum-mono*) **using** *rs*(*1*)[*OF b'-im*] **by**(*auto simp add*:∗ *field-simps*)
also have ... < *e* ∗ *real p* **using** *p* ‹*e*>*0*› ‹*p*>*0*› **unfolding** *n-def real-of-nat-def*
by(*auto simp add*:*field-simps*)
finally have (∑ *i*∈*UNIV*. |*real* (*r* (*b'* *i*)) − *real* (*q* (*b'* *i*))|) < *e* ∗ *real p* **. }**
moreover
{ have (∑ *i*∈*UNIV*. |*real* (*s* (*b'* *i*)) − *real* (*q* (*b'* *i*))|) ≤ (∑ (*i*::'*n*)∈*UNIV*. *1*)
apply(*rule setsum-mono*) **using** *rs*(*2*)[*OF b'-im*] **by**(*auto simp add*:∗ *field-simps*)
also have ... < *e* ∗ *real p* **using** *p* ‹*e*>*0*› ‹*p*>*0*› **unfolding** *n-def real-of-nat-def*
by(*auto simp add*:*field-simps*)
finally have (∑ *i*∈*UNIV*. |*real* (*s* (*b'* *i*)) − *real* (*q* (*b'* *i*))|) < *e* ∗ *real p* **. }**
ultimately
have *norm* (*r'* − *z*) < *e norm* (*s'* − *z*) < *e* **unfolding** *r'-def s'-def z-def* **apply**−
apply(*rule-tac*[!] *le-less-trans*[*OF norm-le-l1*]) **using** ‹*p*>*0*›
by(*auto simp add*:*field-simps setsum-divide-distrib*[*THEN sym*])
hence |(*f z* − *z*) $ *i*| < *d* / *real n* **apply**−**apply**(*rule e*(*2*)[*OF* ‹*r'*∈{*0*..*1*}›
‹*s'*∈{*0*..*1*}› ‹*z*∈{*0*..*1*}›])
using *rs*(*3*) **unfolding** *r'-def*[*symmetric*] *s'-def*[*symmetric*] *o-def bb'* **by** *auto*
thus *False* **using** *i* **by** *auto* **qed**

20.12 Retractions.

definition *retraction s t* (*r*::*real*^'*n*⇒*real*^'*n*) ⟷
t ⊆ *s* ∧ *continuous-on s r* ∧ (*r* ‘ *s* ⊆ *t*) ∧ (∀ *x*∈*t*. *r x* = *x*)

definition *retract-of* (**infixl** *retract'-of 12*) **where**
(*t retract-of s*) ⟷ (∃ *r*. *retraction s t r*)

lemma *retraction-idempotent*: *retraction s t r* ⟹ *x* ∈ *s* ⟹ *r*(*r x*) = *r x*
unfolding *retraction-def* **by** *auto*

20.13 preservation of fixpoints under (more general notion of) retraction.

lemma *invertible-fixpoint-property*: **fixes** *s*::(*real*^'*n*) *set* **and** *t*::(*real*^'*m*) *set*
assumes *continuous-on t i i* ‘ *t* ⊆ *s continuous-on s r r* ‘ *s* ⊆ *t* ∀ *y*∈*t*. *r* (*i y*) = *y*
∀ *f*. *continuous-on s f* ∧ *f* ‘ *s* ⊆ *s* ⟶ (∃ *x*∈*s*. *f x* = *x*) *continuous-on t g g* ‘ *t* ⊆ *t*
obtains *y* **where** *y*∈*t g y* = *y* **proof**−
have ∃ *x*∈*s*. (*i* ∘ *g* ∘ *r*) *x* = *x* **apply**(*rule assms*(*6*)[*rule-format*],*rule*)
apply(*rule continuous-on-compose assms*)+ **apply**((*rule continuous-on-subset*)*?*,*rule assms*)+
using *assms*(*2*,*4*,*8*) **unfolding** *image-compose* **by**(*auto*,*blast*)
then guess *x* **.. note** *x* = *this* **hence** ∗:*g* (*r x*) ∈ *t* **using** *assms*(*4*,*8*) **by** *auto*

have r $((i \circ g \circ r)\ x) = r\ x$ **using** x **by** *auto*
thus *?thesis* **apply**(*rule-tac that*[*of r x*]) **using** x **unfolding** *o-def*
unfolding *assms*(*5*)[*rule-format*,*OF* $*$] **using** *assms*(*4*) **by** *auto* **qed**

lemma *homeomorphic-fixpoint-property*:
fixes s::(*real^'n*) *set* **and** t::(*real^'m*) *set* **assumes** s *homeomorphic* t
shows $(\forall f.$ *continuous-on* $s\ f \wedge f\ `\ s \subseteq s \longrightarrow (\exists x{\in}s.\ f\ x = x)) \longleftrightarrow$
$(\forall g.$ *continuous-on* $t\ g \wedge g\ `\ t \subseteq t \longrightarrow (\exists y{\in}t.\ g\ y = y))$ **proof**−
guess r **using** *assms*[*unfolded homeomorphic-def homeomorphism-def*] **.. then guess** i **..**
thus *?thesis* **apply**− **apply** *rule* **apply**(*rule-tac*[!] *allI impI*)+
apply(*rule-tac g*=g **in** *invertible-fixpoint-property*[*of t i s r*]) **prefer** *10*
apply(*rule-tac g*=f **in** *invertible-fixpoint-property*[*of s r t i*]) **by** *auto* **qed**

lemma *retract-fixpoint-property*:
assumes t *retract-of* s $\forall f.$ *continuous-on* $s\ f \wedge f\ `\ s \subseteq s \longrightarrow (\exists x{\in}s.\ f\ x = x)$ *continuous-on* $t\ g\ g\ `\ t \subseteq t$
obtains y **where** $y \in t\ g\ y = y$ **proof**− **guess** h **using** *assms*(*1*) **unfolding** *retract-of-def* **..**
thus *?thesis* **unfolding** *retraction-def* **apply**−
apply(*rule invertible-fixpoint-property*[*OF continuous-on-id - - - - assms*(*2*), *of t h g*]) **prefer** *7*
apply(*rule-tac y*=y **in** *that*) **using** *assms* **by** *auto* **qed**

20.14 So the Brouwer theorem for any set with nonempty interior.

lemma *brouwer-weak*: **fixes** f::*real^'n* $\Rightarrow$ *real^'n*
assumes *compact s convex s interior s* $\neq \{\}$ *continuous-on s f f ` s* $\subseteq s$
obtains x **where** $x \in s\ f\ x = x$ **proof**−
have $*$:*interior* $\{0..1$::*real^'n*$\} \neq \{\}$ **unfolding** *interior-closed-interval interval-eq-empty* **by** *auto*
have $*$:$\{0..1$::*real^'n*$\}$ *homeomorphic s* **using** *homeomorphic-convex-compact*[*OF convex-interval*(*1*) *compact-interval* $*$ *assms*(*2*,*1*,*3*)] **.**
have $\forall f.$ *continuous-on* $\{0..1\}\ f \wedge f\ `\ \{0..1\} \subseteq \{0..1\} \longrightarrow (\exists x{\in}\{0..1$::*real^'n*$\}.$ $f\ x = x)$ **using** *brouwer-cube* **by** *auto*
thus *?thesis* **unfolding** *homeomorphic-fixpoint-property*[*OF* $*$] **apply**(*erule-tac x*=f **in** *allE*)
apply(*erule impE*) **defer apply**(*erule bexE*) **apply**(*rule-tac x*=y **in** *that*) **using** *assms* **by** *auto* **qed**

20.15 And in particular for a closed ball.

lemma *brouwer-ball*: **fixes** f::*real^'n* $\Rightarrow$ *real^'n*
assumes $0 < e$ *continuous-on* $(cball\ a\ e)\ f\ f\ `\ (cball\ a\ e) \subseteq (cball\ a\ e)$
obtains x **where** $x \in cball\ a\ e\ f\ x = x$
using *brouwer-weak*[*OF compact-cball convex-cball*,*of a e f*] **unfolding** *interior-cball ball-eq-empty*
using *assms* **by** *auto*

Still more general form; could derive this directly without using the rather involved *HOMEOMORPHIC-CONVEX-COMPACT* theorem, just using a scaling and translation to put the set inside the unit cube.

lemma *brouwer*: **fixes** *f*::*real^'n* ⇒ *real^'n*
assumes *compact s convex s s* ≠ {} *continuous-on s f f ' s* ⊆ *s*
obtains *x* **where** *x* ∈ *s f x* = *x* **proof**−
have ∃ *e*>*0*. *s* ⊆ *cball 0 e* **using** *compact-imp-bounded*[*OF assms*(*1*)] **unfolding** *bounded-pos*
apply(*erule-tac exE*,*rule-tac x*=*b* **in** *exI*) **by**(*auto simp add*: *dist-norm*)
then guess *e* **apply**−**apply**(*erule exE*,(*erule conjE*)+) **. note** *e*=*this*
have ∃ *x*∈ *cball 0 e*. (*f* ∘ *closest-point s*) *x* = *x*
apply(*rule-tac brouwer-ball*[*OF e*(*1*), *of 0 f* ∘ *closest-point s*]) **apply**(*rule continuous-on-compose*)
apply(*rule continuous-on-closest-point*[*OF assms*(*2*) *compact-imp-closed*[*OF assms*(*1*)] *assms*(*3*)])
apply(*rule continuous-on-subset*[*OF assms*(*4*)])
using *closest-point-in-set*[*OF compact-imp-closed*[*OF assms*(*1*)] *assms*(*3*)] **apply** − **defer**
using *assms*(*5*)[*unfolded subset-eq*] **using** *e*(*2*)[*unfolded subset-eq mem-cball*] **by**(*auto simp add*: *dist-norm*)
then guess *x* **..** **note** *x*=*this*
have ∗:*closest-point s x* = *x* **apply**(*rule closest-point-self*)
apply(*rule assms*(*5*)[*unfolded subset-eq*,*THEN bspec*[**where** *x*=*x*],*unfolded image-iff*])
apply(*rule-tac x*=*closest-point s x* **in** *bexI*) **using** *x* **unfolding** *o-def*
using *closest-point-in-set*[*OF compact-imp-closed*[*OF assms*(*1*)] *assms*(*3*), *of x*] **by** *auto*
show *thesis* **apply**(*rule-tac x*=*closest-point s x* **in** *that*) **unfolding** *x*(*2*)[*unfolded o-def*]
apply(*rule closest-point-in-set*[*OF compact-imp-closed*[*OF assms*(*1*)] *assms*(*3*)]) **using** ∗ **by** *auto* **qed**

So we get the no-retraction theorem.

lemma *no-retraction-cball*: **assumes** *0* < *e*
shows ¬ (*frontier*(*cball a e*) *retract-of* (*cball a e*)) **proof case** *goal1*
have ∗:⋀*xa*. *a* − (*2* ∗*R a* − *xa*) = −(*a* − *xa*) **using** *scaleR-left-distrib*[*of 1 1 a*] **by** *auto*
guess *x* **apply**(*rule retract-fixpoint-property*[*OF goal1*, *of* λ*x*. *scaleR 2 a* − *x*])
apply(*rule*,*rule*,*erule conjE*) **apply**(*rule brouwer-ball*[*OF assms*]) **apply** *assumption*+
apply(*rule-tac x*=*x* **in** *bexI*) **apply** *assumption*+ **apply**(*rule continuous-on-intros*)+
unfolding *frontier-cball subset-eq Ball-def image-iff* **apply**(*rule*,*rule*,*erule bexE*)
unfolding *dist-norm* **apply**(*simp add*: ∗ *norm-minus-commute*) **. note** *x* = *this*
hence *scaleR 2 a* = *scaleR 1 x* + *scaleR 1 x* **by**(*auto simp add*:*algebra-simps*)
hence *a* = *x* **unfolding** *scaleR-left-distrib*[*THEN sym*] **by** *auto*
thus *False* **using** *x* **using** *assms* **by** *auto* **qed**

20.16 Bijections between intervals.

definition *interval-bij* = (λ (*a*,*b*) (*u*,*v*) (*x*::*real^'n*).
(χ *i*. *u*$*i* + (*x*$*i* − *a*$*i*) / (*b*$*i* − *a*$*i*) ∗ (*v*$*i* − *u*$*i*))::*real^'n*)

lemma *interval-bij-affine*:
interval-bij (*a*,*b*) (*u*,*v*) = (λ*x*. (χ *i*. (*v*$*i* − *u*$*i*) / (*b*$*i* − *a*$*i*) ∗ *x*$*i*) +
(χ *i*. *u*$*i* − (*v*$*i* − *u*$*i*) / (*b*$*i* − *a*$*i*) ∗ *a*$*i*))
apply *rule* **unfolding** *Cart-eq interval-bij-def vector-component-simps*
by(*auto simp add*: *field-simps add-divide-distrib*[*THEN sym*])

lemma *continuous-interval-bij*:
continuous (*at x*) (*interval-bij* (*a*,*b*::*real^'n*) (*u*,*v*))
unfolding *interval-bij-affine* **apply**(*rule continuous-intros*)
apply(*rule linear-continuous-at*) **unfolding** *linear-conv-bounded-linear*[*THEN sym*]
unfolding *linear-def* **unfolding** *Cart-eq* **unfolding** *Cart-lambda-beta* **defer**
apply(*rule continuous-intros*) **by**(*auto simp add*:*field-simps add-divide-distrib*[*THEN sym*])

lemma *continuous-on-interval-bij*: *continuous-on s* (*interval-bij* (*a*,*b*) (*u*,*v*))
apply(*rule continuous-at-imp-continuous-on*) **by**(*rule*, *rule continuous-interval-bij*)

lemma *divide-nonneg-nonneg*:**assumes** *a* ≥ *0 b* ≥ *0* **shows** *0* ≤ *a* / (*b*::*real*)
apply(*cases b=0*) **defer apply**(*rule divide-nonneg-pos*) **using** *assms* **by** *auto*

lemma *in-interval-interval-bij*: **assumes** *x* ∈ {*a*..*b*} {*u*..*v*} ≠ {}
shows *interval-bij* (*a*,*b*) (*u*,*v*) *x* ∈ {*u*..*v*::*real^'n*}
unfolding *interval-bij-def split-conv mem-interval Cart-lambda-beta* **proof**(*rule*,*rule*)

fix *i*::*'n* **have** {*a*..*b*} ≠ {} **using** *assms* **by** *auto*
hence ∗:*a*$*i* ≤ *b*$*i u*$*i* ≤ *v*$*i* **using** *assms*(*2*) **unfolding** *interval-eq-empty not-ex*
apply−
apply(*erule-tac*[!] *x=i* **in** *allE*)+ **by** *auto*
have *x*:*a*$*i*≤*x*$*i x*$*i*≤*b*$*i* **using** *assms*(*1*)[*unfolded mem-interval*] **by** *auto*
have *0* ≤ (*x* $ *i* − *a* $ *i*) / (*b* $ *i* − *a* $ *i*) ∗ (*v* $ *i* − *u* $ *i*)
apply(*rule mult-nonneg-nonneg*) **apply**(*rule divide-nonneg-nonneg*)
using ∗ *x* **by**(*auto simp add*:*field-simps*)
thus *u* $ *i* ≤ *u* $ *i* + (*x* $ *i* − *a* $ *i*) / (*b* $ *i* − *a* $ *i*) ∗ (*v* $ *i* − *u* $ *i*) **using**
∗ **by** *auto*
have ((*x* $ *i* − *a* $ *i*) / (*b* $ *i* − *a* $ *i*)) ∗ (*v* $ *i* − *u* $ *i*) ≤ *1* ∗ (*v* $ *i* − *u* $ *i*)
apply(*rule mult-right-mono*) **unfolding** *divide-le-eq-1* **using** ∗ *x* **by** *auto*
thus *u* $ *i* + (*x* $ *i* − *a* $ *i*) / (*b* $ *i* − *a* $ *i*) ∗ (*v* $ *i* − *u* $ *i*) ≤ *v* $ *i* **using**
∗ **by** *auto* **qed**

lemma *interval-bij-bij*: **assumes** ∀ *i*. *a*$*i* < *b*$*i* ∧ *u*$*i* < *v*$*i*
shows *interval-bij* (*a*,*b*) (*u*,*v*) (*interval-bij* (*u*,*v*) (*a*,*b*) *x*) = *x*
unfolding *interval-bij-def split-conv Cart-eq Cart-lambda-beta*
apply(*rule*,*insert assms*,*erule-tac x=i* **in** *allE*) **by** *auto*

end

21 Operator-Norm: Operator Norm

theory *Operator-Norm*
imports *Euclidean-Space*
begin

definition *onorm f* = *Sup* {*norm* (*f x*)| *x*. *norm x* = *1*}

lemma *norm-bound-generalize*:
fixes *f*:: *real* ^*'n* ⇒ *real*^*'m*
assumes *lf*: *linear f*
shows (∀ *x*. *norm x* = *1* ⟶ *norm* (*f x*) ≤ *b*) ⟷ (∀ *x*. *norm* (*f x*) ≤ *b* ∗ *norm x*) (**is** *?lhs* ⟷ *?rhs*)
proof−
{assume *H*: *?rhs*
{fix *x* :: *real*^*'n* **assume** *x*: *norm x* = *1*
from *H*[*rule-format*, *of x*] *x* **have** *norm* (*f x*) ≤ *b* **by** *simp***}**
then have *?lhs* **by** *blast* **}**

moreover
{assume *H*: *?lhs*
from *H*[*rule-format*, *of basis arbitrary*]
have *bp*: *b* ≥ *0* **using** *norm-ge-zero*[*of f* (*basis arbitrary*)]
by (*auto simp add*: *norm-basis elim*: *order-trans* [*OF norm-ge-zero*])
{fix *x* :: *real* ^*'n*
{assume *x* = *0*
then have *norm* (*f x*) ≤ *b* ∗ *norm x* **by** (*simp add*: *linear-0*[*OF lf*] *bp*)**}**
moreover
{assume *x0*: *x* ≠ *0*
hence *n0*: *norm x* ≠ *0* **by** (*metis norm-eq-zero*)
let *?c* = *1*/ *norm x*
have *norm* (*?c* ∗*R x*) = *1* **using** *x0* **by** (*simp add*: *n0*)
with *H* **have** *norm* (*f* (*?c* ∗*R x*)) ≤ *b* **by** *blast*
hence *?c* ∗ *norm* (*f x*) ≤ *b*
by (*simp add*: *linear-cmul*[*OF lf*])
hence *norm* (*f x*) ≤ *b* ∗ *norm x*
using *n0 norm-ge-zero*[*of x*] **by** (*auto simp add*: *field-simps*)**}**
ultimately have *norm* (*f x*) ≤ *b* ∗ *norm x* **by** *blast***}**
then have *?rhs* **by** *blast***}**
ultimately show *?thesis* **by** *blast*
qed

lemma *onorm*:
fixes *f*:: *real* ^*'n* ⇒ *real* ^*'m*
assumes *lf*: *linear f*

```
  shows norm (f x) <= onorm f * norm x
  and ∀x. norm (f x) <= b * norm x ⟹ onorm f <= b
proof−
  {
    let ?S = {norm (f x) |x. norm x = 1}
    have Se: ?S ≠ {} using  norm-basis by auto
    from linear-bounded[OF lf] have b: ∃ b. ?S *<= b
        unfolding norm-bound-generalize[OF lf, symmetric] by (auto simp add:
setle-def)
    {from Sup[OF Se b, unfolded onorm-def[symmetric]]
      show norm (f x) <= onorm f * norm x
        apply −
        apply (rule spec[where x = x])
        unfolding norm-bound-generalize[OF lf, symmetric]
        by (auto simp add: isLub-def isUb-def leastP-def setge-def setle-def)}
    {
      show ∀x. norm (f x) <= b * norm x ⟹ onorm f <= b
        using Sup[OF Se b, unfolded onorm-def[symmetric]]
        unfolding norm-bound-generalize[OF lf, symmetric]
        by (auto simp add: isLub-def isUb-def leastP-def setge-def setle-def)}
  }
qed

lemma onorm-pos-le: assumes lf: linear (f::real ^'n ⇒ real ^'m) shows 0 <=
onorm f
  using order-trans[OF norm-ge-zero onorm(1)[OF lf, of basis arbitrary], unfolded
norm-basis] by simp

lemma onorm-eq-0: assumes lf: linear (f::real ^'n ⇒ real ^'m)
  shows onorm f = 0 ⟷ (∀x. f x = 0)
  using onorm[OF lf]
  apply (auto simp add: onorm-pos-le)
  apply atomize
  apply (erule allE[where x=0::real])
  using onorm-pos-le[OF lf]
  apply arith
  done

lemma onorm-const: onorm(λx::real^'n. (y::real ^'m)) = norm y
proof−
  let ?f = λx::real^'n. (y::real ^ 'm)
  have th: {norm (?f x)| x. norm x = 1} = {norm y}
    by(auto intro: vector-choose-size set-ext)
  show ?thesis
    unfolding onorm-def th
    apply (rule Sup-unique) by (simp-all  add: setle-def)
qed

lemma onorm-pos-lt: assumes lf: linear (f::real ^ 'n ⇒ real ^'m)
```

shows *0 < onorm f ⟷ ~(∀ x. f x = 0)*
unfolding *onorm-eq-0[OF lf, symmetric]*
using *onorm-pos-le[OF lf]* **by** *arith*

lemma *onorm-compose*:
assumes *lf*: *linear (f::real ^'n ⇒ real ^'m)*
and *lg*: *linear (g::real^'k ⇒ real^'n)*
shows *onorm (f o g) <= onorm f * onorm g*
apply (*rule onorm(2)[OF linear-compose[OF lg lf], rule-format]*)
unfolding *o-def*
apply (*subst mult-assoc*)
apply (*rule order-trans*)
apply (*rule onorm(1)[OF lf]*)
apply (*rule mult-mono1*)
apply (*rule onorm(1)[OF lg]*)
apply (*rule onorm-pos-le[OF lf]*)
done

lemma *onorm-neg-lemma*: **assumes** *lf*: *linear (f::real ^'n ⇒ real^'m)*
shows *onorm (λx. − f x) ≤ onorm f*
using *onorm[OF linear-compose-neg[OF lf]] onorm[OF lf]*
unfolding *norm-minus-cancel* **by** *metis*

lemma *onorm-neg*: **assumes** *lf*: *linear (f::real ^'n ⇒ real^'m)*
shows *onorm (λx. − f x) = onorm f*
using *onorm-neg-lemma[OF lf] onorm-neg-lemma[OF linear-compose-neg[OF lf]]*
by *simp*

lemma *onorm-triangle*:
assumes *lf*: *linear (f::real ^'n ⇒ real ^'m)* **and** *lg*: *linear g*
shows *onorm (λx. f x + g x) <= onorm f + onorm g*
apply(*rule onorm(2)[OF linear-compose-add[OF lf lg], rule-format]*)
apply (*rule order-trans*)
apply (*rule norm-triangle-ineq*)
apply (*simp add*: *distrib*)
apply (*rule add-mono*)
apply (*rule onorm(1)[OF lf]*)
apply (*rule onorm(1)[OF lg]*)
done

lemma *onorm-triangle-le*: *linear (f::real ^'n ⇒ real ^'m) ⟹ linear g ⟹ onorm(f) + onorm(g) <= e*
⟹ onorm(λx. f x + g x) <= e
apply (*rule order-trans*)
apply (*rule onorm-triangle*)
apply *assumption+*
done

lemma *onorm-triangle-lt*: *linear (f::real ^'n ⇒ real ^'m) ⟹ linear g ⟹ onorm(f)*

+ *onorm*(*g*) < *e*
==> *onorm*(λ*x*. *f x* + *g x*) < *e*
apply (*rule order-le-less-trans*)
apply (*rule onorm-triangle*)
by *assumption*+

end

22 Derivative: Multivariate calculus in Euclidean space.

theory *Derivative*
imports *Brouwer-Fixpoint Vec1 RealVector Operator-Norm*
begin

lemmas *linear-linear* = *linear-conv-bounded-linear*[*THEN sym*]

22.1 Derivatives

The definition is slightly tricky since we make it work over nets of a particular form. This lets us prove theorems generally and use "at a" or "at a within s" for restriction to a set (1-sided on R etc.)

definition *has-derivative* :: ('*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*) ⇒ ('*a* ⇒ '*b*) ⇒ ('*a net* ⇒ *bool*)
(**infixl** (*has'-derivative*) *12*) **where**
(*f has-derivative f'*) *net* ≡ *bounded-linear f'* ∧ ((λ*y*. (*1* / (*norm* (*y* − *netlimit net*))) *$_R$*
(*f y* − (*f* (*netlimit net*) + *f'*(*y* − *netlimit net*)))) −−−> *0*) *net*

lemma *derivative-linear*[*dest*]:(*f has-derivative f'*) *net* ⟹ *bounded-linear f'*
unfolding *has-derivative-def* **by** *auto*

lemma *DERIV-conv-has-derivative*:(*DERIV f x* :> *f'*) = (*f has-derivative op* ∗ *f'*) (*at* (*x*::*real*)) (**is** *?l* = *?r*) **proof**
assume *?l* **note** *as* = *this*[*unfolded deriv-def LIM-def*,*rule-format*]
show *?r* **unfolding** *has-derivative-def Lim-at* **apply**− **apply**(*rule*,*rule mult.bounded-linear-right*)
apply *safe* **apply**(*drule as*,*safe*) **apply**(*rule-tac x*=*s* **in** *exI*) **apply** *safe*
apply(*erule-tac x*=*xa* − *x* **in** *allE*) **unfolding** *dist-norm netlimit-at*[*of x*]
unfolding *diff-0-right norm-scaleR*
by(*auto simp add*:*field-simps*)
next assume *?r* **note** *this*[*unfolded has-derivative-def Lim-at*] **note** *as*=*conjunct2*[*OF this*,*rule-format*]
have ∗:⋀*x xa f'*. *xa* ≠ *0* ⟹ |(*f* (*xa* + *x*) − *f x*) / *xa* − *f'*| = |(*f* (*xa* + *x*) − *f x*) − *xa* ∗ *f'*| / |*xa*| **by**(*auto simp add*:*field-simps*)

show *?l* **unfolding** *deriv-def LIM-def* **apply** *safe* **apply**(*drule as*,*safe*)
apply(*rule-tac x=d* **in** *exI*,*safe*) **apply**(*erule-tac x=xa + x* **in** *allE*)
unfolding *dist-norm diff-0-right norm-scaleR*
unfolding *dist-norm netlimit-at*[*of x*] **by**(*auto simp add*:*algebra-simps* ∗) **qed**

lemma *FDERIV-conv-has-derivative*:*FDERIV f* (*x*::*'a*::{*real-normed-vector*,*perfect-space*})
:> *f' = (f has-derivative f') (at x)* (**is** *?l = ?r*) **proof**
assume *?l* **note** *as = this*[*unfolded fderiv-def*]
show *?r* **unfolding** *has-derivative-def Lim-at* **apply**−**apply**(*rule*,*rule as*[*THEN conjunct1*]) **proof**(*rule*,*rule*)
fix *e*::*real* **assume** *e>0*
guess *d* **using** *as*[*THEN conjunct2*,*unfolded LIM-def*,*rule-format*,*OF*‹*e>0*›] **..**
thus ∃ *d>0*. ∀ *xa*. *0 < dist xa x* ∧ *dist xa x < d* ⟶
dist ((*1 / norm* (*xa − netlimit* (*at x*))) ∗*R* (*f xa* − (*f* (*netlimit* (*at x*)) + *f'* (*xa − netlimit* (*at x*))))) (*0*) < *e*
apply(*rule-tac x=d* **in** *exI*) **apply**(*erule conjE*,*rule*,*assumption*) **apply** *rule*
apply(*erule-tac x=xa − x* **in** *allE*)
unfolding *dist-norm netlimit-at*[*of x*] **by** (*auto simp add*: *diff-diff-eq*) **qed**
next
assume *?r* **note** *as = this*[*unfolded has-derivative-def*]
show *?l* **unfolding** *fderiv-def LIM-def* **apply**−**apply**(*rule*,*rule as*[*THEN conjunct1*]) **proof**(*rule*,*rule*)
fix *e*::*real* **assume** *e>0*
guess *d* **using** *as*[*THEN conjunct2*,*unfolded Lim-at*,*rule-format*,*OF*‹*e>0*›] **..**
thus ∃ *s>0*. ∀ *xa*. *xa* ≠ *0* ∧ *dist xa 0 < s* ⟶ *dist* (*norm* (*f* (*x + xa*) − *f x* − *f' xa*) / *norm xa*) *0 < e* **apply**−
apply(*rule-tac x=d* **in** *exI*) **apply**(*erule conjE*,*rule*,*assumption*) **apply** *rule*
apply(*erule-tac x=xa + x* **in** *allE*)
unfolding *dist-norm netlimit-at*[*of x*] **by** (*auto simp add*: *diff-diff-eq add.commute*)
qed qed

22.2 These are the only cases we'll care about, probably.

lemma *has-derivative-within*: (*f has-derivative f'*) (*at x within s*) ⟷
bounded-linear f' ∧ ((λ*y*. (*1 / norm*(*y − x*)) ∗*R* (*f y* − (*f x* + *f'* (*y − x*)))) −−−> *0*) (*at x within s*)
unfolding *has-derivative-def* **and** *Lim* **by**(*auto simp add*:*netlimit-within*)

lemma *has-derivative-at*: (*f has-derivative f'*) (*at x*) ⟷
bounded-linear f' ∧ ((λ*y*. (*1 / (norm*(*y − x*))) ∗*R* (*f y* − (*f x* + *f'* (*y − x*)))) −−−> *0*) (*at x*)
apply(*subst within-UNIV*[*THEN sym*]) **unfolding** *has-derivative-within* **unfolding** *within-UNIV* **by** *auto*

22.3 More explicit epsilon-delta forms.

lemma *has-derivative-within'*:
(*f has-derivative f'*)(*at x within s*) ⟷ *bounded-linear f'* ∧
(∀ *e>0*. ∃ *d>0*. ∀ *x'*∈*s*. *0 < norm*(*x' − x*) ∧ *norm*(*x' − x*) < *d*
⟶ *norm*(*f x' − f x − f'*(*x' − x*)) / *norm*(*x' − x*) < *e*)

unfolding *has-derivative-within Lim-within dist-norm*
unfolding *diff-0-right norm-mul* **by** (*simp add*: *diff-diff-eq*)

lemma *has-derivative-at'*:
(*f has-derivative f'*) (*at x*) ⟷ *bounded-linear f'* ∧
(∀ *e*>*0*. ∃ *d*>*0*. ∀ *x'*. *0* < *norm*(*x'* − *x*) ∧ *norm*(*x'* − *x*) < *d*
⟶ *norm*(*f x'* − *f x* − *f'*(*x'* − *x*)) / *norm*(*x'* − *x*) < *e*)
apply(*subst within-UNIV*[*THEN sym*]) **unfolding** *has-derivative-within'* **by** *auto*

lemma *has-derivative-at-within*: (*f has-derivative f'*) (*at x*) ⟹ (*f has-derivative f'*) (*at x within s*)
unfolding *has-derivative-within' has-derivative-at'* **by** *meson*

lemma *has-derivative-within-open*:
a ∈ *s* ⟹ *open s* ⟹ ((*f has-derivative f'*) (*at a within s*) ⟷ (*f has-derivative f'*) (*at a*))
unfolding *has-derivative-within has-derivative-at* **using** *Lim-within-open* **by** *auto*

22.4 Derivatives on real = Derivatives on (*real*, *1*) *cart*

lemma *has-derivative-within-vec1-dest-vec1*: **fixes** *f*::*real*⇒*real* **shows**
((*vec1* ∘ *f* ∘ *dest-vec1*) *has-derivative* (*vec1* ∘ *f'* ∘ *dest-vec1*)) (*at* (*vec1 x*) *within vec1* ` *s*)
= (*f has-derivative f'*) (*at x within s*)
unfolding *has-derivative-within* **unfolding** *bounded-linear-vec1-dest-vec1*[*unfolded linear-conv-bounded-linear*]
unfolding *o-def Lim-within Ball-def* **unfolding** *forall-vec1*
unfolding *vec1-dest-vec1-simps dist-vec1-0 image-iff* **by** *auto*

lemma *has-derivative-at-vec1-dest-vec1*: **fixes** *f*::*real*⇒*real* **shows**
((*vec1* ∘ *f* ∘ *dest-vec1*) *has-derivative* (*vec1* ∘ *f'* ∘ *dest-vec1*)) (*at* (*vec1 x*)) = (*f has-derivative f'*) (*at x*)
using *has-derivative-within-vec1-dest-vec1*[**where** *s*=*UNIV*, *unfolded range-vec1 within-UNIV*] **by** *auto*

lemma *bounded-linear-vec1*: **fixes** *f*::'*a*::*real-normed-vector*⇒*real*
shows *bounded-linear f* = *bounded-linear* (*vec1* ∘ *f*)
unfolding *bounded-linear-def additive-def bounded-linear-axioms-def o-def*
unfolding *vec1-dest-vec1-simps* **by** *auto*

lemma *bounded-linear-dest-vec1*: **fixes** *f*::*real*⇒'*a*::*real-normed-vector*
shows *bounded-linear f* = *bounded-linear* (*f* ∘ *dest-vec1*)
unfolding *bounded-linear-def additive-def bounded-linear-axioms-def o-def*
unfolding *vec1-dest-vec1-simps* **by** *auto*

lemma *has-derivative-at-vec1*: **fixes** *f*::'*a*::*real-normed-vector*⇒*real* **shows**
(*f has-derivative f'*) (*at x*) = ((*vec1* ∘ *f*) *has-derivative* (*vec1* ∘ *f'*)) (*at x*)
unfolding *has-derivative-at* **unfolding** *bounded-linear-vec1*[*unfolded linear-conv-bounded-linear*]
unfolding *o-def Lim-at* **unfolding** *vec1-dest-vec1-simps dist-vec1-0* **by** *auto*

lemma *has-derivative-within-dest-vec1*:**fixes** *f*::*real*⇒'*a*::*real-normed-vector* **shows**
((*f* ∘ *dest-vec1*) *has-derivative* (*f'* ∘ *dest-vec1*)) (*at* (*vec1 x*) *within vec1* ' *s*) =
(*f has-derivative f'*) (*at x within s*)
unfolding *has-derivative-within bounded-linear-dest-vec1* **unfolding** *o-def Lim-within Ball-def*
unfolding *vec1-dest-vec1-simps dist-vec1-0 image-iff* **by** *auto*

lemma *has-derivative-at-dest-vec1*:**fixes** *f*::*real*⇒'*a*::*real-normed-vector* **shows**
((*f* ∘ *dest-vec1*) *has-derivative* (*f'* ∘ *dest-vec1*)) (*at* (*vec1 x*)) = (*f has-derivative*
f') (*at x*)
using *has-derivative-within-dest-vec1*[**where** *s*=*UNIV*] **by**(*auto simp add*:*within-UNIV*)

lemma *derivative-is-linear*: **fixes** *f*::*real*^'*a* ⇒ *real*^'*b* **shows**
(*f has-derivative f'*) *net* ⟹ *linear f'*
unfolding *has-derivative-def* **and** *linear-conv-bounded-linear* **by** *auto*

22.5 Combining theorems.

lemma (**in** *bounded-linear*) *has-derivative*: (*f has-derivative f*) *net*
unfolding *has-derivative-def* **apply**(*rule*,*rule bounded-linear-axioms*)
unfolding *diff* **by**(*simp add*: *Lim-const*)

lemma *has-derivative-id*: ((λ*x*. *x*) *has-derivative* (λ*h*. *h*)) *net*
apply(*rule bounded-linear.has-derivative*) **using** *bounded-linear-ident*[*unfolded id-def*] **by** *simp*

lemma *has-derivative-const*: ((λ*x*. *c*) *has-derivative* (λ*h*. *0*)) *net*
unfolding *has-derivative-def* **apply**(*rule*,*rule bounded-linear-zero*) **by**(*simp add*: *Lim-const*)

lemma (**in** *bounded-linear*) *cmul*: **shows** *bounded-linear* (λ*x*. (*c*::*real*) ∗$_R$ *f x*)
proof −
have *bounded-linear* (λ*x*. *c* ∗$_R$ *x*)
by (*rule scaleR.bounded-linear-right*)
moreover have *bounded-linear f* **..**
ultimately show *?thesis*
by (*rule bounded-linear-compose*)
qed

lemma *has-derivative-cmul*: **assumes** (*f has-derivative f'*) *net* **shows** ((λ*x*. *c* ∗$_R$ *f*(*x*)) *has-derivative* (λ*h*. *c* ∗$_R$ *f'*(*h*))) *net*
unfolding *has-derivative-def* **apply**(*rule*,*rule bounded-linear.cmul*)
using *assms*[*unfolded has-derivative-def*] **using** *Lim-cmul*[*OF assms*[*unfolded has-derivative-def*,*THEN conjunct2*]]
unfolding *scaleR-right-diff-distrib scaleR-right-distrib* **by** *auto*

lemma *has-derivative-cmul-eq*: **assumes** *c* ≠ *0*
shows (((λ*x*. *c* ∗$_R$ *f*(*x*)) *has-derivative* (λ*h*. *c* ∗$_R$ *f'*(*h*))) *net* ⟷ (*f has-derivative*

f ′) *net*)
apply(*rule*) **defer apply**(*rule has-derivative-cmul*,*assumption*)
apply(*drule has-derivative-cmul*[**where** *c*=*1* / *c*]) **using** *assms* **by** *auto*

lemma *has-derivative-neg*:
(*f has-derivative f ′*) *net* ⟹ ((λ*x*. −(*f x*)) *has-derivative* (λ*h*. −(*f ′ h*))) *net*
apply(*drule has-derivative-cmul*[**where** *c*=−*1*]) **by** *auto*

lemma *has-derivative-neg-eq*: ((λ*x*. −(*f x*)) *has-derivative* (λ*h*. −(*f ′ h*))) *net* ⟷
(*f has-derivative f ′*) *net*
apply(*rule*, *drule-tac*[!] *has-derivative-neg*) **by** *auto*

lemma *has-derivative-add*: **assumes** (*f has-derivative f ′*) *net* (*g has-derivative g ′*)
net
shows ((λ*x*. *f*(*x*) + *g*(*x*)) *has-derivative* (λ*h*. *f ′*(*h*) + *g ′*(*h*))) *net* **proof**−
note *as* = *assms*[*unfolded has-derivative-def*]
show *?thesis* **unfolding** *has-derivative-def* **apply**(*rule*,*rule bounded-linear-add*)
using *Lim-add*[*OF as*(*1*)[*THEN conjunct2*] *as*(*2*)[*THEN conjunct2*]] **and** *as*
by (*auto simp add*:*algebra-simps scaleR-right-diff-distrib scaleR-right-distrib*)
qed

lemma *has-derivative-add-const*:(*f has-derivative f ′*) *net* ⟹ ((λ*x*. *f x* + *c*) *has-derivative*
f ′) *net*
apply(*drule has-derivative-add*) **apply**(*rule has-derivative-const*) **by** *auto*

lemma *has-derivative-sub*:
(*f has-derivative f ′*) *net* ⟹ (*g has-derivative g ′*) *net* ⟹ ((λ*x*. *f*(*x*) − *g*(*x*))
has-derivative (λ*h*. *f ′*(*h*) − *g ′*(*h*))) *net*
apply(*drule has-derivative-add*) **apply**(*drule has-derivative-neg*,*assumption*) **by**(*simp*
add:*algebra-simps*)

lemma *has-derivative-setsum*: **assumes** *finite s* ∀ *a*∈*s*. ((*f a*) *has-derivative* (*f ′*
a)) *net*
shows ((λ*x*. *setsum* (λ*a*. *f a x*) *s*) *has-derivative* (λ*h*. *setsum* (λ*a*. *f ′ a h*) *s*))
net
apply(*induct-tac s rule*:*finite-subset-induct*[**where** *A*=*s*]) **apply**(*rule assms*(*1*))

proof− **fix** *x F* **assume** *as*:*finite F x* ∉ *F x*∈*s* ((λ*x*. ∑ *a*∈*F*. *f a x*) *has-derivative*
(λ*h*. ∑ *a*∈*F*. *f ′ a h*)) *net*
thus ((λ*xa*. ∑ *a*∈*insert x F*. *f a xa*) *has-derivative* (λ*h*. ∑ *a*∈*insert x F*. *f ′ a*
h)) *net*
unfolding *setsum-insert*[*OF as*(*1*,*2*)] **apply**−**apply**(*rule has-derivative-add*)
apply(*rule assms*(*2*)[*rule-format*]) **by** *auto*
qed(*auto intro*!: *has-derivative-const*)

lemma *has-derivative-setsum-numseg*:
∀ *i*. *m* ≤ *i* ∧ *i* ≤ *n* ⟶ ((*f i*) *has-derivative* (*f ′ i*)) *net* ⟹
((λ*x*. *setsum* (λ*i*. *f i x*) {*m*..*n*::*nat*}) *has-derivative* (λ*h*. *setsum* (λ*i*. *f ′ i h*)
{*m*..*n*})) *net*

apply(*rule has-derivative-setsum*) **by** *auto*

22.6 somewhat different results for derivative of scalar multiplier.

lemma *has-derivative-vmul-component*: **fixes** *c*::*real^'a* ⇒ *real^'b* **and** *v*::*real^'c*
assumes (*c has-derivative c'*) *net*
shows ((λ*x*. *c*(*x*)$*k* *$_R$ *v*) *has-derivative* (λ*x*. (*c' x*)$*k* *$_R$ *v*)) *net* **proof**−
have *:⋀*y*. (*c y* $ *k* *$_R$ *v* − (*c* (*netlimit net*) $ *k* *$_R$ *v* + *c'* (*y* − *netlimit net*) $ *k* *$_R$ *v*)) =
(*c y* $ *k* − (*c* (*netlimit net*) $ *k* + *c'* (*y* − *netlimit net*) $ *k*)) *$_R$ *v*
unfolding *scaleR-left-diff-distrib scaleR-left-distrib* **by** *auto*
show *?thesis* **unfolding** *has-derivative-def* **and** * **and** *linear-conv-bounded-linear*[*symmetric*]
apply(*rule*,*rule linear-vmul-component*[*of c' k v*, *unfolded smult-conv-scaleR*])
defer
apply(*subst vector-smult-lzero*[*THEN sym*, *of v*]) **unfolding** *scaleR-scaleR smult-conv-scaleR* **apply**(*rule Lim-vmul*)
using *assms*[*unfolded has-derivative-def*] **unfolding** *Lim o-def* **apply**− **apply**(*cases trivial-limit net*)
apply(*rule*,*assumption*,*rule disjI2*,*rule*,*rule*) **proof**−
have *:⋀*x*. *x* − *vec 0* = (*x*::*real^'n*) **by** *auto*
have **:⋀*d x*. *d* * (*c x* $ *k* − (*c* (*netlimit net*) $ *k* + *c'* (*x* − *netlimit net*) $ *k*)) = (*d* *$_R$ (*c x* − (*c* (*netlimit net*) + *c'* (*x* − *netlimit net*)))) $*k* **by**(*auto simp add*:*field-simps*)
fix *e* **assume** ¬ *trivial-limit net* 0 < (*e*::*real*)
then have *eventually* (λ*x*. *dist* ((*1* / *norm* (*x* − *netlimit net*)) *$_R$ (*c x* − (*c* (*netlimit net*) + *c'* (*x* − *netlimit net*)))) *0* < *e*) *net*
using *assms*[*unfolded has-derivative-def Lim*] **by** *auto*
thus *eventually* (λ*x*. *dist* (*1* / *norm* (*x* − *netlimit net*) * (*c x* $ *k* − (*c* (*netlimit net*) $ *k* + *c'* (*x* − *netlimit net*) $ *k*))) *0* < *e*) *net*
proof (*rule eventually-elim1*)
case *goal1* **thus** *?case* **apply** − **unfolding** *dist-norm* **apply**(*rule le-less-trans*)
prefer *2* **apply** *assumption* **unfolding** * ** **and** *norm-vec1*
using *component-le-norm*[*of* (*1* / *norm* (*x* − *netlimit net*)) *$_R$ (*c x* − (*c* (*netlimit net*) + *c'* (*x* − *netlimit net*))) − *0 k*] **by** *auto*
qed
qed(*insert assms*[*unfolded has-derivative-def*], *auto simp add*:*linear-conv-bounded-linear*)
qed

lemma *has-derivative-vmul-within*: **fixes** *c*::*real* ⇒ *real* **and** *v*::*real^'a*
assumes (*c has-derivative c'*) (*at x within s*)
shows ((λ*x*. (*c x*) *$_R$ *v*) *has-derivative* (λ*x*. (*c' x*) *$_R$ *v*)) (*at x within s*) **proof**−
have *:⋀*c*. (λ*x*. (*vec1* ∘ *c* ∘ *dest-vec1*) *x* $ *1* *$_R$ *v*) = (λ*x*. (*c x*) *$_R$ *v*) ∘ *dest-vec1*
unfolding *o-def* **by** *auto*
show *?thesis* **using** *has-derivative-vmul-component*[*of vec1* ∘ *c* ∘ *dest-vec1 vec1* ∘ *c'* ∘ *dest-vec1 at* (*vec1 x*) *within vec1* ‘ *s 1 v*]
unfolding * **and** *has-derivative-within-vec1-dest-vec1* **unfolding** *has-derivative-within-dest-vec1*
using *assms* **by** *auto* **qed**

lemma *has-derivative-vmul-at*: **fixes** *c*::*real* ⇒ *real* **and** *v*::*real*^'*a*
assumes (*c has-derivative c'*) (*at x*)
shows ((λ*x*. (*c x*) *$*_R$ *v*) *has-derivative* (λ*x*. (*c' x*) *$*_R$ *v*)) (*at x*)
using *has-derivative-vmul-within*[**where** *s*=*UNIV*] **and** *assms* **by**(*auto simp add*: *within-UNIV*)

lemma *has-derivative-lift-dot*:
assumes (*f has-derivative f'*) *net*
shows ((λ*x*. *inner v* (*f x*)) *has-derivative* (λ*t*. *inner v* (*f' t*))) *net* **proof**−
show *?thesis* **using** *assms* **unfolding** *has-derivative-def* **apply**− **apply**(*erule conjE*,*rule*)
apply(*rule bounded-linear-compose*[*of - f'*]) **apply**(*rule inner.bounded-linear-right*,*assumption*)
apply(*drule Lim-inner*[**where** *a*=*v*]) **unfolding** *o-def*
by(*auto simp add*:*inner.scaleR-right inner.add-right inner.diff-right*) **qed**

lemmas *has-derivative-intros* = *has-derivative-sub has-derivative-add has-derivative-cmul has-derivative-id has-derivative-const*
has-derivative-neg has-derivative-vmul-component has-derivative-vmul-at has-derivative-vmul-within has-derivative-cmul
bounded-linear.has-derivative has-derivative-lift-dot

22.7 limit transformation for derivatives.

lemma *has-derivative-transform-within*:
assumes *0* < *d x* ∈ *s* ∀ *x'*∈*s*. *dist x' x* < *d* ⟶ *f x'* = *g x'* (*f has-derivative f'*) (*at x within s*)
shows (*g has-derivative f'*) (*at x within s*)
using *assms*(*4*) **unfolding** *has-derivative-within* **apply**− **apply**(*erule conjE*,*rule*,*assumption*)
apply(*rule Lim-transform-within*[*OF assms*(*1*)]) **defer apply** *assumption*
apply(*rule*,*rule*) **apply**(*drule assms*(*3*)[*rule-format*]) **using** *assms*(*3*)[*rule-format*, *OF assms*(*2*)] **by** *auto*

lemma *has-derivative-transform-at*:
assumes *0* < *d* ∀ *x'*. *dist x' x* < *d* ⟶ *f x'* = *g x'* (*f has-derivative f'*) (*at x*)
shows (*g has-derivative f'*) (*at x*)
apply(*subst within-UNIV*[*THEN sym*]) **apply**(*rule has-derivative-transform-within*[*OF assms*(*1*)])
using *assms*(*2*−*3*) **unfolding** *within-UNIV* **by** *auto*

lemma *has-derivative-transform-within-open*:
assumes *open s x* ∈ *s* ∀ *y*∈*s*. *f y* = *g y* (*f has-derivative f'*) (*at x*)
shows (*g has-derivative f'*) (*at x*)
using *assms*(*4*) **unfolding** *has-derivative-at* **apply**− **apply**(*erule conjE*,*rule*,*assumption*)
apply(*rule Lim-transform-within-open*[*OF assms*(*1*,*2*)]) **defer apply** *assumption*
apply(*rule*,*rule*) **apply**(*drule assms*(*3*)[*rule-format*]) **using** *assms*(*3*)[*rule-format*, *OF assms*(*2*)] **by** *auto*

22.8 differentiability.

no-notation *Deriv.differentiable* (**infixl** *differentiable 60*)

definition *differentiable* :: (*'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector*) ⇒ *'a net* ⇒ *bool* (**infixr** *differentiable 30*) **where**
f differentiable net ≡ (∃*f'*. (*f has-derivative f'*) *net*)

definition *differentiable-on* :: (*'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector*) ⇒ *'a set* ⇒ *bool* (**infixr** *differentiable'-on 30*) **where**
f differentiable-on s ≡ (∀*x*∈*s*. *f differentiable* (*at x within s*))

lemma *differentiableI*: (*f has-derivative f'*) *net* ⟹ *f differentiable net*
unfolding *differentiable-def* **by** *auto*

lemma *differentiable-at-withinI*: *f differentiable* (*at x*) ⟹ *f differentiable* (*at x within s*)
unfolding *differentiable-def* **using** *has-derivative-at-within* **by** *blast*

lemma *differentiable-within-open*: **assumes** *a* ∈ *s open s* **shows**
f differentiable (*at a within s*) ⟷ (*f differentiable* (*at a*))
unfolding *differentiable-def has-derivative-within-open*[*OF assms*] **by** *auto*

lemma *differentiable-at-imp-differentiable-on*: (∀*x*∈(*s*::(*real^'n*) *set*). *f differentiable at x*) ⟹ *f differentiable-on s*
unfolding *differentiable-on-def* **by**(*auto intro*!: *differentiable-at-withinI*)

lemma *differentiable-on-eq-differentiable-at*: *open s* ⟹ (*f differentiable-on s* ⟷ (∀*x*∈*s*. *f differentiable at x*))
unfolding *differentiable-on-def* **by**(*auto simp add*: *differentiable-within-open*)

lemma *differentiable-transform-within*:
assumes *0* < *d x* ∈ *s* ∀*x'*∈*s*. *dist x' x* < *d* ⟶ *f x'* = *g x' f differentiable* (*at x within s*)
shows *g differentiable* (*at x within s*)
using *assms*(*4*) **unfolding** *differentiable-def* **by**(*auto intro*!: *has-derivative-transform-within*[*OF assms*(*1*−*3*)])

lemma *differentiable-transform-at*:
assumes *0* < *d* ∀*x'*. *dist x' x* < *d* ⟶ *f x'* = *g x' f differentiable at x*
shows *g differentiable at x*
using *assms*(*3*) **unfolding** *differentiable-def* **using** *has-derivative-transform-at*[*OF assms*(*1*−*2*)] **by** *auto*

22.9 Frechet derivative and Jacobian matrix.

definition *frechet-derivative f net* = (*SOME f'*. (*f has-derivative f'*) *net*)

lemma *frechet-derivative-works*:
f differentiable net ⟷ (*f has-derivative* (*frechet-derivative f net*)) *net*
unfolding *frechet-derivative-def differentiable-def* **and** *some-eq-ex*[*of* λ *f'* . (*f has-derivative f'*) *net*] **..**

lemma *linear-frechet-derivative*: **fixes** *f*::*real^'a* ⇒ *real^'b*
shows *f differentiable net* ⟹ *linear*(*frechet-derivative f net*)
unfolding *frechet-derivative-works has-derivative-def* **unfolding** *linear-conv-bounded-linear*
by *auto*

definition *jacobian f net* = *matrix*(*frechet-derivative f net*)

lemma *jacobian-works*: (*f*::(*real^'a*) ⇒ (*real^'b*)) *differentiable net* ⟷ (*f has-derivative* (λ*h*. (*jacobian f net*) ∗*v h*)) *net*
apply *rule* **unfolding** *jacobian-def* **apply**(*simp only*: *matrix-works*[*OF linear-frechet-derivative*])
defer
apply(*rule differentiableI*) **apply** *assumption* **unfolding** *frechet-derivative-works*
by *assumption*

22.10 Differentiability implies continuity.

lemma *Lim-mul-norm-within*: **fixes** *f*::*'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector*
shows (*f* −−−> *0*) (*at a within s*) ⟹ ((λ*x*. *norm*(*x* − *a*) ∗*R f*(*x*)) −−−> *0*) (*at a within s*)
unfolding *Lim-within* **apply**(*rule*,*rule*) **apply**(*erule-tac x*=*e* **in** *allE*,*erule impE*,*assumption*,*erule exE*,*erule conjE*)
apply(*rule-tac x*=*min d 1* **in** *exI*) **apply** *rule* **defer apply**(*rule*,*erule-tac x*=*x* **in** *ballE*) **unfolding** *dist-norm diff-0-right norm-mul*
by(*auto intro*!: *mult-strict-mono*[*of - 1*::*real*, *unfolded mult-1-left*])

lemma *differentiable-imp-continuous-within*: **assumes** *f differentiable* (*at x within s*)
shows *continuous* (*at x within s*) *f* **proof**−
from *assms* **guess** *f'* **unfolding** *differentiable-def has-derivative-within* **..** **note** *f'*=*this*
then interpret *bounded-linear f'* **by** *auto*
have ∗:⋀*xa*. *x*≠*xa* ⟹ (*f'* ∘ (λ*y*. *y* − *x*)) *xa* + *norm* (*xa* − *x*) ∗*R* ((*1* / *norm* (*xa* − *x*)) ∗*R* (*f xa* − (*f x* + *f'* (*xa* − *x*)))) − ((*f'* ∘ (λ*y*. *y* − *x*)) *x* + *0*) = *f xa* − *f x*
using *zero* **by** *auto*
have ∗∗:*continuous* (*at x within s*) (*f'* ∘ (λ*y*. *y* − *x*))
apply(*rule continuous-within-compose*) **apply**(*rule continuous-intros*)+
by(*rule linear-continuous-within*[*OF f'*[*THEN conjunct1*]])
show *?thesis* **unfolding** *continuous-within* **using** *f'*[*THEN conjunct2*, *THEN Lim-mul-norm-within*]
apply−**apply**(*drule Lim-add*) **apply**(*rule* ∗∗[*unfolded continuous-within*]) **unfolding** *Lim-within* **and** *dist-norm*
apply(*rule*,*rule*) **apply**(*erule-tac x*=*e* **in** *allE*) **apply**(*erule impE*|*assumption*)+ **apply**(*erule exE*,*rule-tac x*=*d* **in** *exI*)
by(*auto simp add*:*zero* ∗ *elim*!:*allE*) **qed**

lemma *differentiable-imp-continuous-at*: *f differentiable at x* ⟹ *continuous* (*at x*) *f*

by(*rule differentiable-imp-continuous-within*[*of - x UNIV*, *unfolded within-UNIV*])

lemma *differentiable-imp-continuous-on*: *f differentiable-on s* $\Longrightarrow$ *continuous-on s f*
unfolding *differentiable-on-def continuous-on-eq-continuous-within*
using *differentiable-imp-continuous-within* **by** *blast*

lemma *has-derivative-within-subset*:
(*f has-derivative f'*) (*at x within s*) $\Longrightarrow$ $t \subseteq s$ $\Longrightarrow$ (*f has-derivative f'*) (*at x within t*)
unfolding *has-derivative-within* **using** *Lim-within-subset* **by** *blast*

lemma *differentiable-within-subset*:
f differentiable (*at x within t*) $\Longrightarrow$ $s \subseteq t$ $\Longrightarrow$ *f differentiable* (*at x within s*)
unfolding *differentiable-def* **using** *has-derivative-within-subset* **by** *blast*

lemma *differentiable-on-subset*: *f differentiable-on t* $\Longrightarrow$ $s \subseteq t$ $\Longrightarrow$ *f differentiable-on s*
unfolding *differentiable-on-def* **using** *differentiable-within-subset* **by** *blast*

lemma *differentiable-on-empty*: *f differentiable-on* {}
unfolding *differentiable-on-def* **by** *auto*

22.11 Several results are easier using a "multiplied-out" variant. *) (* (I got this idea from Dieudonne's proof of the chain rule).

lemma *has-derivative-within-alt*:
(*f has-derivative f'*) (*at x within s*) $\longleftrightarrow$ *bounded-linear f'* $\wedge$
($\forall e{>}0.\ \exists d{>}0.\ \forall y{\in}s.\ norm(y - x) < d \longrightarrow norm(f(y) - f(x) - f'(y - x)) \leq e * norm(y - x)$) (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof assume *?lhs* **thus** *?rhs* **unfolding** *has-derivative-within* **apply**$-$**apply**(*erule conjE*,*rule*,*assumption*)
unfolding *Lim-within* **apply**(*rule*,*erule-tac x*=*e* **in** *allE*,*rule*,*erule impE*,*assumption*)
apply(*erule exE*,*rule-tac x*=*d* **in** *exI*) **apply**(*erule conjE*,*rule*,*assumption*,*rule*,*rule*)
proof$-$
fix *x y e d* **assume** *as*:$0 < e$ $0 < d$ *norm* $(y - x) < d$ $\forall xa{\in}s.\ 0 < dist\ xa\ x \wedge dist\ xa\ x < d \longrightarrow$
dist $((1\ /\ norm\ (xa - x)) *_R (f\ xa - (f\ x + f'\ (xa - x))))\ 0 < e$ $y \in s$ *bounded-linear f'*
then interpret *bounded-linear f'* **by** *auto*
show *norm* $(f\ y - f\ x - f'\ (y - x)) \leq e * norm\ (y - x)$ **proof**(*cases y*=*x*)
case *True* **thus** *?thesis* **using** ‹*bounded-linear f'*› **by**(*auto simp add*: *zero*)
next
case *False* **hence** *norm* $(f\ y - (f\ x + f'\ (y - x))) < e * norm\ (y - x)$ **using** *as*(*4*)[*rule-format*, *OF* ‹*y*∈*s*›]
unfolding *dist-norm diff-0-right norm-mul* **using** *as*(*3*)
using *pos-divide-less-eq*[*OF False*[*unfolded dist-nz*], *unfolded dist-norm*]
by (*auto simp add*: *linear-0 linear-sub*)

thus *?thesis* **by**(*auto simp add*:*algebra-simps*) **qed qed next**
assume *?rhs* **thus** *?lhs* **unfolding** *has-derivative-within Lim-within* **apply**−**apply**(*erule conjE*,*rule*,*assumption*)
apply(*rule*,*erule-tac x*=*e*/*2* **in** *allE*,*rule*,*erule impE*) **defer apply**(*erule exE*,*rule-tac x*=*d* **in** *exI*)
apply(*erule conjE*,*rule*,*assumption*,*rule*,*rule*) **unfolding** *dist-norm diff-0-right norm-scaleR*
apply(*erule-tac x*=*xa* **in** *ballE*,*erule impE*) **proof**−
fix *e d y* **assume** *bounded-linear f' 0 < e 0 < d y ∈ s 0 < norm (y − x) ∧ norm (y − x) < d*
*norm (f y − f x − f' (y − x)) ≤ e / 2 * norm (y − x)*
thus *|1 / norm (y − x)| * norm (f y − (f x + f' (y − x))) < e*
apply(*rule-tac le-less-trans*[*of - e*/*2*]) **by**(*auto intro*!:*mult-imp-div-pos-le simp add*:*algebra-simps*) **qed** *auto* **qed**

lemma *has-derivative-at-alt*:
(f has-derivative f') (at x) ⟷ bounded-linear f' ∧
*(∀ e>0. ∃ d>0. ∀ y. norm(y − x) < d ⟶ norm(f y − f x − f'(y − x)) ≤ e * norm(y − x))*
using *has-derivative-within-alt*[**where** *s*=*UNIV*] **unfolding** *within-UNIV* **by** *auto*

22.12 The chain rule.

lemma *diff-chain-within*:
assumes *(f has-derivative f') (at x within s) (g has-derivative g') (at (f x) within (f ' s))*
shows *((g o f) has-derivative (g' o f'))(at x within s)*
unfolding *has-derivative-within-alt* **apply**(*rule*,*rule bounded-linear-compose*[*unfolded o-def*[*THEN sym*]])
apply(*rule assms*(*2*)[*unfolded has-derivative-def*,*THEN conjE*],*assumption*)
apply(*rule assms*(*1*)[*unfolded has-derivative-def*,*THEN conjE*],*assumption*) **proof**(*rule*,*rule*)
note *assms* = *assms*[*unfolded has-derivative-within-alt*]
fix *e*::*real* **assume** *0*<*e*
guess *B1* **using** *bounded-linear.pos-bounded*[*OF assms*(*1*)[*THEN conjunct1*, *rule-format*]] **.. note** *B1* = *this*
guess *B2* **using** *bounded-linear.pos-bounded*[*OF assms*(*2*)[*THEN conjunct1*, *rule-format*]] **.. note** *B2* = *this*
have *∗*:*e / 2 / B2 > 0* **using** ‹*e*>*0*› *B2* **apply**−**apply**(*rule divide-pos-pos*) **by** *auto*
guess *d1* **using** *assms*(*1*)[*THEN conjunct2*, *rule-format*, *OF ∗*] **.. note** *d1* = *this*
have *∗*:*e / 2 / (1 + B1) > 0* **using** ‹*e*>*0*› *B1* **apply**−**apply**(*rule divide-pos-pos*) **by** *auto*
guess *de* **using** *assms*(*2*)[*THEN conjunct2*, *rule-format*, *OF ∗*] **.. note** *de* = *this*
guess *d2* **using** *assms*(*1*)[*THEN conjunct2*, *rule-format*, *OF zero-less-one*] **.. note** *d2* = *this*

def *d0* ≡ (*min d1 d2*)/*2* **have** *d0*:*d0*>*0 d0* < *d1 d0* < *d2* **unfolding** *d0-def* **using** *d1 d2* **by** *auto*
def *d* ≡ (*min d0* (*de* / (*B1* + *1*))) / *2* **have** *de* ∗ *2* / (*B1* + *1*) > *de* / (*B1* + *1*) **apply**(*rule divide-strict-right-mono*) **using** *B1 de* **by** *auto*
hence *d*:*d*>*0 d* < *d1 d* < *d2 d* < (*de* / (*B1* + *1*)) **unfolding** *d-def* **using** *d0 d1 d2 de B1* **by**(*auto intro*!: *divide-pos-pos simp add*:*min-less-iff-disj not-less*)

show ∃ *d*>*0*. ∀ *y*∈*s*. *norm* (*y* − *x*) < *d* ⟶ *norm* ((*g* ∘ *f*) *y* − (*g* ∘ *f*) *x* − (*g*′ ∘ *f*′) (*y* − *x*)) ≤ *e* ∗ *norm* (*y* − *x*) **apply**(*rule-tac x*=*d* **in** *exI*)
proof(*rule*,*rule* ‹*d*>*0*›,*rule*,*rule*)
fix *y* **assume** *as*:*y* ∈ *s norm* (*y* − *x*) < *d*
hence *1*:*norm* (*f y* − *f x* − *f*′ (*y* − *x*)) ≤ *min* (*norm* (*y* − *x*)) (*e* / *2* / *B2* ∗ *norm* (*y* − *x*)) **using** *d1 d2 d* **by** *auto*

have *norm* (*f y* − *f x*) ≤ *norm* (*f y* − *f x* − *f*′ (*y* − *x*)) + *norm* (*f*′ (*y* − *x*))
using *norm-triangle-sub*[*of f y* − *f x f*′ (*y* − *x*)] **by**(*auto simp add*:*algebra-simps*)
also have ... ≤ *norm* (*f y* − *f x* − *f*′ (*y* − *x*)) + *B1* ∗ *norm* (*y* − *x*) **apply**(*rule add-left-mono*) **using** *B1* **by**(*auto simp add*:*algebra-simps*)
also have ... ≤ *min* (*norm* (*y* − *x*)) (*e* / *2* / *B2* ∗ *norm* (*y* − *x*)) + *B1* ∗ *norm* (*y* − *x*) **apply**(*rule add-right-mono*) **using** *d1 d2 d as* **by** *auto*
also have ... ≤ *norm* (*y* − *x*) + *B1* ∗ *norm* (*y* − *x*) **by** *auto*
also have ... = *norm* (*y* − *x*) ∗ (*1* + *B1*) **by**(*auto simp add*:*field-simps*)
finally have *3*:*norm* (*f y* − *f x*) ≤ *norm* (*y* − *x*) ∗ (*1* + *B1*) **by** *auto*

hence *norm* (*f y* − *f x*) ≤ *d* ∗ (*1* + *B1*) **apply**− **apply**(*rule order-trans*,*assumption*,*rule mult-right-mono*) **using** *as B1* **by** *auto*
also have ... < *de* **using** *d B1* **by**(*auto simp add*:*field-simps*)
finally have *norm* (*g* (*f y*) − *g* (*f x*) − *g*′ (*f y* − *f x*)) ≤ *e* / *2* / (*1* + *B1*) ∗ *norm* (*f y* − *f x*)
apply−**apply**(*rule de*[*THEN conjunct2*,*rule-format*]) **using** ‹*y*∈*s*› **using** *d as* **by** *auto*
also have ... = (*e* / *2*) ∗ (*1* / (*1* + *B1*) ∗ *norm* (*f y* − *f x*)) **by** *auto*
also have ... ≤ *e* / *2* ∗ *norm* (*y* − *x*) **apply**(*rule mult-left-mono*) **using** ‹*e*>*0*› **and** *3* **using** *B1* **and** ‹*e*>*0*› **by**(*auto simp add*:*divide-le-eq*)
finally have *4*:*norm* (*g* (*f y*) − *g* (*f x*) − *g*′ (*f y* − *f x*)) ≤ *e* / *2* ∗ *norm* (*y* − *x*) **by** *auto*

interpret *g*′: *bounded-linear g*′ **using** *assms*(*2*) **by** *auto*
interpret *f*′: *bounded-linear f*′ **using** *assms*(*1*) **by** *auto*
have *norm* (− *g*′ (*f*′ (*y* − *x*)) + *g*′ (*f y* − *f x*)) = *norm* (*g*′ (*f y* − *f x* − *f*′ (*y* − *x*)))
by(*auto simp add*:*algebra-simps f*′.*diff g*′.*diff g*′.*add*)
also have ... ≤ *B2* ∗ *norm* (*f y* − *f x* − *f*′ (*y* − *x*)) **using** *B2* **by**(*auto simp add*:*algebra-simps*)
also have ... ≤ *B2* ∗ (*e* / *2* / *B2* ∗ *norm* (*y* − *x*)) **apply**(*rule mult-left-mono*) **using** *as d1 d2 d B2* **by** *auto*
also have ... ≤ *e* / *2* ∗ *norm* (*y* − *x*) **using** *B2* **by** *auto*
finally have *5*:*norm* (− *g*′ (*f*′ (*y* − *x*)) + *g*′ (*f y* − *f x*)) ≤ *e* / *2* ∗ *norm* (*y* − *x*) **by** *auto*

have *norm* $(g\ (f\ y) - g\ (f\ x) - g'\ (f\ y - f\ x)) + norm\ (g\ (f\ y) - g\ (f\ x) - g'\ (f'\ (y - x)) - (g\ (f\ y) - g\ (f\ x) - g'\ (f\ y - f\ x))) \leq e * norm\ (y - x)$ **using** *5 4* **by** *auto*
thus *norm* $((g \circ f)\ y - (g \circ f)\ x - (g' \circ f')\ (y - x)) \leq e * norm\ (y - x)$ **unfolding** *o-def* **apply**− **apply**(*rule order-trans*, *rule norm-triangle-sub*) **by** *assumption* **qed qed**

lemma *diff-chain-at*:
$(f$ *has-derivative* $f')\ (at\ x) \Longrightarrow (g$ *has-derivative* $g')\ (at\ (f\ x)) \Longrightarrow ((g\ o\ f)$ *has-derivative* $(g'\ o\ f'))\ (at\ x)$
using *diff-chain-within*[*of f f' x UNIV g g'*] **using** *has-derivative-within-subset*[*of g g' f x UNIV range f*] **unfolding** *within-UNIV* **by** *auto*

22.13 Composition rules stated just for differentiability.

lemma *differentiable-const*[*intro*]: $(\lambda z.\ c)$ *differentiable* (*net*::'*a*::*real-normed-vector net*)
unfolding *differentiable-def* **using** *has-derivative-const* **by** *auto*

lemma *differentiable-id*[*intro*]: $(\lambda z.\ z)$ *differentiable* (*net*::'*a*::*real-normed-vector net*)
unfolding *differentiable-def* **using** *has-derivative-id* **by** *auto*

lemma *differentiable-cmul*[*intro*]: *f differentiable net* $\Longrightarrow (\lambda x.\ c *_R f(x))$ *differentiable* (*net*::'*a*::*real-normed-vector net*)
unfolding *differentiable-def* **apply**(*erule exE*, *drule has-derivative-cmul*) **by** *auto*

lemma *differentiable-neg*[*intro*]: *f differentiable net* $\Longrightarrow (\lambda z.\ -(f\ z))$ *differentiable* (*net*::'*a*::*real-normed-vector net*)
unfolding *differentiable-def* **apply**(*erule exE*, *drule has-derivative-neg*) **by** *auto*

lemma *differentiable-add*: *f differentiable net* $\Longrightarrow$ *g differentiable net*
$\Longrightarrow (\lambda z.\ f\ z + g\ z)$ *differentiable* (*net*::'*a*::*real-normed-vector net*)
unfolding *differentiable-def* **apply**(*erule exE*)+ **apply**(*rule-tac x*=$\lambda z.\ f'\ z + f'a\ z$ **in** *exI*)
apply(*rule has-derivative-add*) **by** *auto*

lemma *differentiable-sub*: *f differentiable net* $\Longrightarrow$ *g differentiable net*
$\Longrightarrow (\lambda z.\ f\ z - g\ z)$ *differentiable* (*net*::'*a*::*real-normed-vector net*)
unfolding *differentiable-def* **apply**(*erule exE*)+ **apply**(*rule-tac x*=$\lambda z.\ f'\ z - f'a\ z$ **in** *exI*)
apply(*rule has-derivative-sub*) **by** *auto*

lemma *differentiable-setsum*: **fixes** *f*::'*a* $\Rightarrow$ (*real*^'*n* $\Rightarrow$*real*^'*n*)
assumes *finite s* $\forall\ a \in s.\ (f\ a)$ *differentiable net*
shows $(\lambda x.\ setsum\ (\lambda a.\ f\ a\ x)\ s)$ *differentiable net* **proof**−
guess *f'* **using** *bchoice*[*OF assms*(*2*)[*unfolded differentiable-def*]] **..**
thus *?thesis* **unfolding** *differentiable-def* **apply**− **apply**(*rule*,*rule has-derivative-setsum*[**where**

f'=f'],*rule assms*(*1*)) **by** *auto* **qed**

lemma *differentiable-setsum-numseg*: **fixes** *f*::- ⇒ (*real^'n* ⇒*real^'n*)
shows ∀ *i*. *m* ≤ *i* ∧ *i* ≤ *n* ⟶ (*f i*) *differentiable net* ⟹ (λ*x*. *setsum* (λ*a*. *f a x*) {*m*::*nat*..*n*}) *differentiable net*
apply(*rule differentiable-setsum*) **using** *finite-atLeastAtMost*[*of n m*] **by** *auto*

lemma *differentiable-chain-at*:
f differentiable (*at x*) ⟹ *g differentiable* (*at*(*f x*)) ⟹ (*g o f*) *differentiable* (*at x*)
unfolding *differentiable-def* **by**(*meson diff-chain-at*)

lemma *differentiable-chain-within*:
f differentiable (*at x within s*) ⟹ *g differentiable* (*at*(*f x*) *within* (*f ' s*))
⟹ (*g o f*) *differentiable* (*at x within s*)
unfolding *differentiable-def* **by**(*meson diff-chain-within*)

22.14 Uniqueness of derivative. *) (* *) (* The general result is a bit messy because we need approachability of the *) (* limit point from any direction. But OK for nontrivial intervals etc.

lemma *frechet-derivative-unique-within*: **fixes** *f*::*real^'a* ⇒ *real^'b*
assumes (*f has-derivative f'*) (*at x within s*) (*f has-derivative f''*) (*at x within s*)
(∀ *i*::*'a*::*finite*. ∀ *e*>*0*. ∃ *d*. *0* < *abs*(*d*) ∧ *abs*(*d*) < *e* ∧ (*x* + *d* ∗*R* *basis i*) ∈ *s*)
shows *f' = f''* **proof**−
note *as* = *assms*(*1*,*2*)[*unfolded has-derivative-def*]
then interpret *f'*: *bounded-linear f'* **by** *auto* **from** *as* **interpret** *f''*: *bounded-linear f''* **by** *auto*
have *x islimpt s* **unfolding** *islimpt-approachable* **proof**(*rule*,*rule*)
guess *a* **using** *UNIV-witness*[**where** *'a*=*'a*] **..**
fix *e*::*real* **assume** *0*<*e* **guess** *d* **using** *assms*(*3*)[*rule-format*,*OF*‹*e*>*0*›,*of a*] **..**
thus ∃ *x'*∈*s*. *x'* ≠ *x* ∧ *dist x' x* < *e* **apply**(*rule-tac x*=*x* + *d*∗*R* *basis a* **in** *bexI*)
using *basis-nonzero*[*of a*] *norm-basis*[*of a*] **unfolding** *dist-norm* **by** *auto* **qed**
hence ∗:*netlimit* (*at x within s*) = *x* **apply**−**apply**(*rule netlimit-within*) **unfolding** *trivial-limit-within* **by** *simp*
show *?thesis* **apply**(*rule linear-eq-stdbasis*) **unfolding** *linear-conv-bounded-linear*
apply(*rule as*(*1*,*2*)[*THEN conjunct1*])+ **proof**(*rule*,*rule ccontr*)
fix *i*::*'a* **def** *e* ≡ *norm* (*f'* (*basis i*) − *f''* (*basis i*))
assume *f'* (*basis i*) ≠ *f''* (*basis i*) **hence** *e*>*0* **unfolding** *e-def* **by** *auto*
guess *d* **using** *Lim-sub*[*OF as*(*1*,*2*)[*THEN conjunct2*], *unfolded* ∗ *Lim-within*,*rule-format*,*OF* ‹*e*>*0*›] **..** **note** *d*=*this*
guess *c* **using** *assms*(*3*)[*rule-format*,*OF d*[*THEN conjunct1*],*of i*] **..** **note** *c*=*this*
have ∗:*norm* (− ((*1* / |*c*|) ∗*R* *f'* (*c* ∗*R* *basis i*)) + (*1* / |*c*|) ∗*R* *f''* (*c* ∗*R* *basis i*)) = *norm* ((*1* / *abs c*) ∗*R* (− (*f'* (*c* ∗*R* *basis i*)) + *f''* (*c* ∗*R* *basis i*)))
unfolding *scaleR-right-distrib* **by** *auto*
also have ... = *norm* ((*1* / *abs c*) ∗*R* (*c* ∗*R* (− (*f'* (*basis i*)) + *f''* (*basis i*))))

unfolding *f'.scaleR f''.scaleR* **unfolding** *scaleR-right-distrib scaleR-minus-right*
by *auto*
also have ... = *e* **unfolding** *e-def norm-mul* **using** *c*[*THEN conjunct1*] **using**
norm-minus-cancel[*of f' (basis i) − f'' (basis i)*] **by** (*auto simp add*: *add.commute*
ab-diff-minus)
finally show *False* **using** *c* **using** *d*[*THEN conjunct2*,*rule-format*,*of x + c* $*_R$
basis i] **using** *norm-basis*[*of i*] **unfolding** *dist-norm*
unfolding *f'.scaleR f''.scaleR f'.add f''.add f'.diff f''.diff scaleR-scaleR*
scaleR-right-diff-distrib scaleR-right-distrib **by** *auto* **qed qed**

lemma *frechet-derivative-unique-at*: **fixes** *f*::*real^'a* ⇒ *real^'b*
shows (*f has-derivative f'*) (*at x*) ⟹ (*f has-derivative f''*) (*at x*) ⟹ *f' = f''*
apply(*rule frechet-derivative-unique-within*[*of f f' x UNIV f''*]) **unfolding** *within-UNIV*
apply(*assumption*)+
apply(*rule*,*rule*,*rule*) **apply**(*rule-tac x=e/2* **in** *exI*) **by** *auto*

lemma *isCont f x = continuous* (*at x*) *f* **unfolding** *isCont-def LIM-def*
unfolding *continuous-at Lim-at* **unfolding** *dist-nz* **by** *auto*

lemma *frechet-derivative-unique-within-closed-interval*: **fixes** *f*::*real^'a* ⇒ *real^'b*
assumes ∀ *i*. *a$i* < *b$i x* ∈ {*a..b*} (**is** *x*∈*?I*) **and**
(*f has-derivative f'*) (*at x within* {*a..b*}) **and**
(*f has-derivative f''*) (*at x within* {*a..b*})
shows *f' = f''* **apply**(*rule frechet-derivative-unique-within*) **apply**(*rule assms*(*3*,*4*))+
proof(*rule*,*rule*,*rule*)
fix *e*::*real* **and** *i*::*'a* **assume** *e>0*
thus ∃ *d*. *0* < |*d*| ∧ |*d*| < *e* ∧ *x* + *d* $*_R$ *basis i* ∈ {*a..b*} **proof**(*cases x$i=a$i*)
case *True* **thus** *?thesis* **apply**(*rule-tac x=*(*min* (*b$i − a$i*) *e*) */ 2* **in** *exI*)
using *assms*(*1*)[*THEN spec*[**where** *x=i*]] **and** ‹*e>0*› **and** *assms*(*2*)
unfolding *mem-interval* **by**(*auto simp add*:*field-simps*) **next**
note ∗ = *assms*(*2*)[*unfolded mem-interval*,*THEN spec*[**where** *x=i*]]
case *False* **moreover have** *a $ i < x $ i* **using** *False* ∗ **by** *auto*
moreover { **have** *a $ i ∗ 2 + min* (*x $ i − a $ i*) *e* ≤ *a$i ∗2 + x$i − a$i*
by *auto*
also have ... = *a$i + x$i* **by** *auto* **also have** ... ≤ *2 ∗ x$i* **using** ∗ **by** *auto*
finally have *a $ i ∗ 2 + min* (*x $ i − a $ i*) *e* ≤ *x $ i ∗ 2* **by** *auto* }
moreover have *min* (*x $ i − a $ i*) *e* ≥ *0* **using** ∗ **and** ‹*e>0*› **by** *auto*
hence *x $ i ∗ 2* ≤ *b $ i ∗ 2 + min* (*x $ i − a $ i*) *e* **using** ∗ **by** *auto*
ultimately show *?thesis* **apply**(*rule-tac x=− (min* (*x$i − a$i*) *e*) */ 2* **in**
exI)
using *assms*(*1*)[*THEN spec*[**where** *x=i*]] **and** ‹*e>0*› **and** *assms*(*2*)
unfolding *mem-interval* **by**(*auto simp add*:*field-simps*) **qed qed**

lemma *frechet-derivative-unique-within-open-interval*: **fixes** *f*::*real^'a* ⇒ *real^'b*
assumes *x* ∈ {*a<..<b*} (*f has-derivative f'*) (*at x within* {*a<..<b*})
(*f has-derivative f''*) (*at x within* {*a<..<b*})
shows *f' = f''* **apply**(*rule frechet-derivative-unique-within*) **apply**(*rule assms*(*2−3*))+
proof(*rule*,*rule*,*rule*)
fix *e*::*real* **and** *i*::*'a* **assume** *e>0*

note $* = assms(1)[unfolded\ mem\text{-}interval, THEN\ spec[\textbf{where}\ x{=}i]]$
have $a\ \$\ i < x\ \$\ i$ **using** $*$ **by** *auto*
moreover { have $a\ \$\ i * 2 + min\ (x\ \$\ i - a\ \$\ i)\ e \leq a\$i * 2 + x\$i - a\$i$ **by** *auto*
also have $\ldots = a\$i + x\$i$ **by** *auto* **also have** $\ldots < 2 * x\$i$ **using** $*$ **by** *auto*
finally have $a\ \$\ i * 2 + min\ (x\ \$\ i - a\ \$\ i)\ e < x\ \$\ i * 2$ **by** *auto* **}**
moreover have $min\ (x\ \$\ i - a\ \$\ i)\ e \geq 0$ **using** $*$ **and** ‹$e{>}0$› **by** *auto*
hence $x\ \$\ i * 2 < b\ \$\ i * 2 + min\ (x\ \$\ i - a\ \$\ i)\ e$ **using** $*$ **by** *auto*
ultimately show $\exists d.\ 0 < |d| \wedge |d| < e \wedge x + d *_R basis\ i \in \{a{<}..{<}b\}$
apply(*rule-tac x=*$-\ (min\ (x\$i - a\$i)\ e)\ /\ 2$ **in** *exI*)
using ‹$e{>}0$› **and** $assms(1)$ **unfolding** *mem-interval* **by**(*auto simp add:field-simps*)
qed

lemma *frechet-derivative-at*: **fixes** $f{::}real\hat{}'a \Rightarrow real\hat{}'b$
shows $(f\ has\text{-}derivative\ f')\ (at\ x) \Longrightarrow (f' = frechet\text{-}derivative\ f\ (at\ x))$
apply(*rule frechet-derivative-unique-at*[*of f*],*assumption*)
unfolding *frechet-derivative-works*[*THEN sym*] **using** *differentiable-def* **by** *auto*

lemma *frechet-derivative-within-closed-interval*: **fixes** $f{::}real\hat{}'a \Rightarrow real\hat{}'b$
assumes $\forall i.\ a\$i < b\$i\ x \in \{a..b\}\ (f\ has\text{-}derivative\ f')\ (at\ x\ within\ \{a..\ b\})$
shows $frechet\text{-}derivative\ f\ (at\ x\ within\ \{a..\ b\}) = f'$
apply(*rule frechet-derivative-unique-within-closed-interval*[**where** *f=f*])
apply(*rule assms*(*1*,*2*))+ **unfolding** *frechet-derivative-works*[*THEN sym*]
unfolding *differentiable-def* **using** $assms(3)$ **by** *auto*

22.15 Component of the differential must be zero if it exists at a local *) (* maximum or minimum for that corresponding component.

lemma *differential-zero-maxmin-component*: **fixes** $f{::}real\hat{}'a \Rightarrow real\hat{}'b$
assumes $0 < e\ ((\forall y \in ball\ x\ e.\ (f\ y)\$k \leq (f\ x)\$k) \vee (\forall y{\in}ball\ x\ e.\ (f\ x)\$k \leq (f\ y)\$k))$
$f\ differentiable\ (at\ x)$ **shows** $jacobian\ f\ (at\ x)\ \$\ k = 0$ **proof**(*rule ccontr*)
def $D \equiv jacobian\ f\ (at\ x)$ **assume** $jacobian\ f\ (at\ x)\ \$\ k \neq 0$
then obtain j **where** $j{:}D\$k\$j \neq 0$ **unfolding** *Cart-eq D-def* **by** *auto*
hence $*{:}abs\ (jacobian\ f\ (at\ x)\ \$\ k\ \$\ j)\ /\ 2 > 0$ **unfolding** *D-def* **by** *auto*
note $as = assms(3)[unfolded\ jacobian\text{-}works\ has\text{-}derivative\text{-}at\text{-}alt]$
guess e' **using** $as[THEN\ conjunct2, rule\text{-}format, OF\ *]$ **.. note** $e' = this$
guess d **using** $real\text{-}lbound\text{-}gt\text{-}zero[OF\ assms(1)\ e'[THEN\ conjunct1]]$ **.. note** $d = this$
{ fix c **assume** $abs\ c \leq d$
hence $*{:}norm\ (x + c *_R basis\ j - x) < e'$ **using** $norm\text{-}basis[of\ j]\ d$ **by** *auto*
have $|(f\ (x + c *_R basis\ j) - f\ x - D *v\ (c *_R basis\ j))\ \$\ k| \leq norm\ (f\ (x + c *_R basis\ j) - f\ x - D *v\ (c *_R basis\ j))$ **by**(*rule component-le-norm*)
also have $\ldots \leq |D\ \$\ k\ \$\ j|\ /\ 2 * |c|$ **using** $e'[THEN\ conjunct2, rule\text{-}format, OF\ *]$ **and** $norm\text{-}basis[of\ j]$ **unfolding** $D\text{-}def[symmetric]$ **by** *auto*
finally have $|(f\ (x + c *_R basis\ j) - f\ x - D *v\ (c *_R basis\ j))\ \$\ k| \leq |D\ \$\ k\ \$\ j|\ /\ 2 * |c|$ **by** *simp*
hence $|f\ (x + c *_R basis\ j)\ \$\ k - f\ x\ \$\ k - c * D\ \$\ k\ \$\ j| \leq |D\ \$\ k\ \$\ j|\ /\ 2$

$* |c|$
unfolding *vector-component-simps matrix-vector-mul-component* **unfolding** *smult-conv-scaleR*[*symmetric*]
unfolding *inner-simps dot-basis smult-conv-scaleR* **by** *simp* **}** **note** $* = this$
have $x + d *_R basis\ j \in ball\ x\ e\ x - d *_R basis\ j \in ball\ x\ e$
unfolding *mem-ball dist-norm* **using** *norm-basis*[*of j*] *d* **by** *auto*
hence $**$:$((f\ (x - d *_R basis\ j))\$k \leq (f\ x)\$k \wedge (f\ (x + d *_R basis\ j))\$k \leq (f\ x)\$k) \vee$
$((f\ (x - d *_R basis\ j))\$k \geq (f\ x)\$k \wedge (f\ (x + d *_R basis\ j))\$k \geq (f\ x)\$k)$
using *assms*(*2*) **by** *auto*
have $***$:$\bigwedge y\ y1\ y2\ d\ dx$::*real*. $(y1 \leq y \wedge y2 \leq y) \vee (y \leq y1 \wedge y \leq y2) \Longrightarrow d < abs\ dx \Longrightarrow abs(y1 - y - - dx) \leq d \Longrightarrow (abs\ (y2 - y - dx) \leq d) \Longrightarrow False$ **by** *arith*
show *False* **apply**(*rule* $***$[*OF* $**$, **where** dx=$d * D\ \$\ k\ \$\ j$ **and** d=$|D\ \$\ k\ \$\ j| / 2 * |d|$])
using $*$[*of* $-d$] **and** $*$[*of d*] **and** *d*[*THEN conjunct1*] **and** *j* **unfolding** *mult-minus-left*
unfolding *abs-mult diff-minus-eq-add scaleR.minus-left* **unfolding** *algebra-simps*
by (*auto intro*: *mult-pos-pos*)
qed

22.16 In particular if we have a mapping into (*real*, *1*) *cart*.

lemma *differential-zero-maxmin*: **fixes** *f*::*real^'a* $\Rightarrow$ *real*
assumes $x \in s$ *open s* (*f has-derivative f'*) (*at x*)
$(\forall y \in s.\ f\ y \leq f\ x) \vee (\forall y \in s.\ f\ x \leq f\ y)$
shows $f' = (\lambda v.\ 0)$ **proof**−
note *deriv* = *assms*(*3*)[*unfolded has-derivative-at-vec1*]
obtain *e* **where** *e*:$e>0$ *ball x e* $\subseteq$ *s* **using** *assms*(*2*)[*unfolded open-contains-ball*]
and *assms*(*1*) **by** *auto*
hence $**$:(*jacobian* (*vec1* $\circ$ *f*) (*at x*)) $ *1* = *0* **using** *differential-zero-maxmin-component*[*of e x* $\lambda x.$ *vec1* (*f x*) *1*]
using *assms*(*4*) **and** *assms*(*3*)[*unfolded has-derivative-at-vec1 o-def*]
unfolding *differentiable-def o-def* **by** *auto*
have $*$:*jacobian* (*vec1* $\circ$ *f*) (*at x*) = *matrix* (*vec1* $\circ$ *f'*) **unfolding** *jacobian-def*
and *frechet-derivative-at*[*OF deriv*] **..**
have *vec1* $\circ$ *f'* = $(\lambda x.\ 0)$ **apply**(*rule*) **unfolding** *matrix-works*[*OF derivative-is-linear*[*OF deriv*],*THEN sym*]
unfolding *Cart-eq matrix-vector-mul-component* **using** $**$[*unfolded* $*$] **by** *auto*
thus *?thesis* **apply**−**apply**(*rule*,*subst vec1-eq*[*THEN sym*]) **unfolding** *o-def* **apply**(*drule fun-cong*) **by** *auto* **qed**

22.17 The traditional Rolle theorem in one dimension.

lemma *rolle*: **fixes** *f*::*real*$\Rightarrow$*real*
assumes $a < b\ f\ a = f\ b$ *continuous-on* $\{a..b\}$ *f*
$\forall x \in \{a<..<b\}.$ (*f has-derivative f'*(*x*)) (*at x*)
shows $\exists x \in \{a<..<b\}.\ f'\ x = (\lambda v.\ 0)$ **proof**−
have $\exists x \in \{a<..<b\}.\ ((\forall y \in \{a<..<b\}.\ f\ x \leq f\ y) \vee (\forall y \in \{a<..<b\}.\ f\ y \leq f\ x))$
proof−
have $(a + b) / 2 \in \{a\ ..\ b\}$ **using** *assms*(*1*) **by** *auto* **hence** $*$:$\{a\ ..\ b\} \neq \{\}$ **by** *auto*

guess *d* **using** *continuous-attains-sup*[*OF compact-real-interval* ∗ *assms*(*3*)] **..**
note *d*=*this*
guess *c* **using** *continuous-attains-inf*[*OF compact-real-interval* ∗ *assms*(*3*)] **..**
note *c*=*this*
show *?thesis* **proof**(*cases d*∈{*a*<..<*b*} ∨ *c*∈{*a*<..<*b*})
case *True* **thus** *?thesis* **apply**(*erule-tac disjE*) **apply**(*rule-tac x*=*d* **in** *bexI*)
apply(*rule-tac*[*3*] *x*=*c* **in** *bexI*) **using** *d c* **by** *auto* **next def** *e* ≡ (*a* + *b*)
/*2*
case *False* **hence** *f d* = *f c* **using** *d c assms*(*2*) **by** *auto*
hence ⋀*x*. *x*∈{*a*..*b*} ⟹ *f x* = *f d* **using** *c d* **apply**− **apply**(*erule-tac x*=*x*
in *ballE*)+ **by** *auto*
thus *?thesis* **apply**(*rule-tac x*=*e* **in** *bexI*) **unfolding** *e-def* **using** *assms*(*1*)
by *auto* **qed qed**
then guess *x* **.. note** *x*=*this*
hence *f* ′ *x* ∘ *dest-vec1* = (λ*v*. *0*) **apply**(*rule-tac differential-zero-maxmin*[*of vec1*
x vec1 ‘ {*a*<..<*b*} *f* ∘ *dest-vec1* (*f* ′ *x*) ∘ *dest-vec1*])
unfolding *vec1-interval* **defer apply**(*rule open-interval*)
apply(*rule assms*(*4*)[*unfolded has-derivative-at-dest-vec1*[*THEN sym*],*THEN*
bspec[**where** *x*=*x*]],*assumption*)
unfolding *o-def* **apply**(*erule disjE*,*rule disjI2*) **by**(*auto simp add*: *vector-less-def*)

thus *?thesis* **apply**(*rule-tac x*=*x* **in** *bexI*) **unfolding** *o-def* **apply** *rule*
apply(*drule-tac x*=*vec1 v* **in** *fun-cong*) **unfolding** *vec1-dest-vec1* **using** *x*(*1*)
by *auto* **qed**

22.18 One-dimensional mean value theorem.

lemma *mvt*: **fixes** *f*::*real* ⇒ *real*
assumes *a* < *b continuous-on* {*a* .. *b*} *f* ∀ *x*∈{*a*<..<*b*}. (*f has-derivative* (*f* ′ *x*))
(*at x*)
shows ∃ *x*∈{*a*<..<*b*}. (*f b* − *f a* = (*f* ′ *x*) (*b* − *a*)) **proof**−
have ∃ *x*∈{*a*<..<*b*}. (λ*xa*. *f* ′ *x xa* − (*f b* − *f a*) / (*b* − *a*) ∗ *xa*) = (λ*v*. *0*)
apply(*rule rolle*[*OF assms*(*1*), *of* λ*x*. *f x* − (*f b* − *f a*) / (*b* − *a*) ∗ *x*]) **defer**
apply(*rule continuous-on-intros assms*(*2*) *continuous-on-cmul*[**where** ′*b*=*real*,
unfolded real-scaleR-def])+ **proof**
fix *x* **assume** *x*:*x* ∈ {*a*<..<*b*}
show ((λ*x*. *f x* − (*f b* − *f a*) / (*b* − *a*) ∗ *x*) *has-derivative* (λ*xa*. *f* ′ *x xa* − (*f*
b − *f a*) / (*b* − *a*) ∗ *xa*)) (*at x*)
by(*rule has-derivative-intros assms*(*3*)[*rule-format*,*OF x*]
has-derivative-cmul[**where** ′*b*=*real*, *unfolded real-scaleR-def*])+
qed(*insert assms*(*1*), *auto simp add*:*field-simps*)
then guess *x* **.. thus** *?thesis* **apply**(*rule-tac x*=*x* **in** *bexI*) **apply**(*drule fun-cong*[*of*
- - *b* − *a*]) **by** *auto* **qed**

lemma *mvt-simple*: **fixes** *f*::*real* ⇒ *real*
assumes *a*<*b* ∀ *x*∈{*a*..*b*}. (*f has-derivative f* ′ *x*) (*at x within* {*a*..*b*})
shows ∃ *x*∈{*a*<..<*b*}. *f b* − *f a* = *f* ′ *x* (*b* − *a*)
apply(*rule mvt*) **apply**(*rule assms*(*1*), *rule differentiable-imp-continuous-on*)
unfolding *differentiable-on-def differentiable-def* **defer proof**

fix x **assume** $x{:}x \in \{a<..<b\}$ **show** $(f$ *has-derivative* $f')\ x)\ (at\ x)$ **unfolding** *has-derivative-within-open*[*OF x open-interval-real*, *THEN sym*]
apply(*rule has-derivative-within-subset*) **apply**(*rule assms*(*2*)[*rule-format*]) **using** x **by** *auto* **qed**(*insert assms*(*2*), *auto*)

lemma *mvt-very-simple*: **fixes** $f{::}real \Rightarrow real$
assumes $a \leq b\ \forall x{\in}\{a..b\}.\ (f$ *has-derivative* $f'(x))\ (at\ x\ within\ \{a..b\})$
shows $\exists x{\in}\{a..b\}.\ f\ b - f\ a = f'\ x\ (b - a)$ **proof**(*cases* $a = b$)
interpret *bounded-linear* f' b **using** *assms*(*2*) *assms*(*1*) **by** *auto*
case *True* **thus** *?thesis* **apply**(*rule-tac* $x{=}a$ **in** *bexI*)
using *assms*(*2*)[*THEN bspec*[**where** $x{=}a$]] **unfolding** *has-derivative-def*
unfolding *True* **using** *zero* **by** *auto* **next**
case *False* **thus** *?thesis* **using** *mvt-simple*[*OF - assms*(*2*)] **using** *assms*(*1*) **by** *auto* **qed**

22.19 A nice generalization (see Havin's proof of 5.19 from Rudin's book).

lemma *mvt-general*: **fixes** $f{::}real{\Rightarrow}real\hat{}'n$
assumes $a{<}b$ *continuous-on* $\{a..b\}\ f\ \forall x{\in}\{a<..<b\}.\ (f$ *has-derivative* $f'(x))\ (at\ x)$
shows $\exists x{\in}\{a<..<b\}.\ norm(f\ b - f\ a) \leq norm(f'(x)\ (b - a))$ **proof**−
have $\exists x{\in}\{a<..<b\}.\ (op \cdot (f\ b - f\ a) \circ f)\ b - (op \cdot (f\ b - f\ a) \circ f)\ a = (f\ b - f\ a) \cdot f'\ x\ (b - a)$
apply(*rule mvt*) **apply**(*rule assms*(*1*)) **apply**(*rule continuous-on-inner continuous-on-intros assms*(*2*))+
unfolding *o-def* **apply**(*rule*,*rule has-derivative-lift-dot*) **using** *assms*(*3*) **by** *auto*
then guess x **.. note** $x{=}this$
show *?thesis* **proof**(*cases* $f\ a = f\ b$)
case *False*
have $norm\ (f\ b - f\ a) * norm\ (f\ b - f\ a) = norm\ (f\ b - f\ a)\hat{}2$ **by**(*simp add*: *power2-eq-square*)
also have $\ldots = (f\ b - f\ a) \cdot (f\ b - f\ a)$ **unfolding** *power2-norm-eq-inner* **..**
also have $\ldots = (f\ b - f\ a) \cdot f'\ x\ (b - a)$ **using** x **unfolding** *inner-simps* **by** *auto*
also have $\ldots \leq norm\ (f\ b - f\ a) * norm\ (f'\ x\ (b - a))$ **by**(*rule norm-cauchy-schwarz*)
finally show *?thesis* **using** *False* $x(1)$ **by**(*auto simp add*: *real-mult-left-cancel*) **next**
case *True* **thus** *?thesis* **using** *assms*(*1*) **apply**(*rule-tac* $x{=}(a + b)\ /2$ **in** *bexI*) **by** *auto* **qed qed**

22.20 Still more general bound theorem.

lemma *differentiable-bound*: **fixes** $f{::}real\hat{}'a \Rightarrow real\hat{}'b$
assumes *convex* $s\ \forall x{\in}s.\ (f$ *has-derivative* $f'(x))\ (at\ x\ within\ s)\ \forall x{\in}s.\ onorm(f'\ x) \leq B$ **and** $x{:}x{\in}s$ **and** $y{:}y{\in}s$
shows $norm(f\ x - f\ y) \leq B * norm(x - y)$ **proof**−
let *?p* $= \lambda u.\ x + u *_R (y - x)$

have $*$:$\bigwedge u.\ u \in \{0..1\} \Longrightarrow x + u *_R (y - x) \in s$
using *assms*(1)[*unfolded convex-alt*,*rule-format*,*OF x y*] **unfolding** *scaleR-left-diff-distrib scaleR-right-diff-distrib* **by**(*auto simp add*:*algebra-simps*)
hence *1*:*continuous-on* $\{0..1\}$ $(f \circ ?p)$ **apply**− **apply**(*rule continuous-on-intros continuous-on-vmul*)+
unfolding *continuous-on-eq-continuous-within* **apply**(*rule*,*rule differentiable-imp-continuous-within*)
unfolding *differentiable-def* **apply**(*rule-tac x*=*f' xa* **in** *exI*)
apply(*rule has-derivative-within-subset*) **apply**(*rule assms*(2)[*rule-format*]) **by** *auto*
have *2*:$\forall u \in \{0<..<1\}.$ $((f \circ ?p)$ *has-derivative* $f'\ (x + u *_R (y - x)) \circ (\lambda u.\ 0 + u *_R (y - x)))\ (at\ u)$ **proof** *rule* **case** *goal1*
let *?u* = $x + u *_R (y - x)$
have $(f \circ ?p$ *has-derivative* $(f'\ ?u) \circ (\lambda u.\ 0 + u *_R (y - x)))\ (at\ u\ within\ \{0<..<1\})$
apply(*rule diff-chain-within*) **apply**(*rule has-derivative-intros*)+
apply(*rule has-derivative-within-subset*) **apply**(*rule assms*(2)[*rule-format*]) **using** *goal1* $*$ **by** *auto*
thus *?case* **unfolding** *has-derivative-within-open*[*OF goal1 open-interval-real*] **by** *auto* **qed**
guess *u* **using** *mvt-general*[*OF zero-less-one 1 2*] **..** **note** $u = this$
have $**$:$\bigwedge x\ y.\ x \in s \Longrightarrow norm\ (f'\ x\ y) \leq B * norm\ y$ **proof**− **case** *goal1*
have $norm\ (f'\ x\ y) \leq onorm\ (f'\ x) * norm\ y$
using *onorm*(1)[*OF derivative-is-linear*[*OF assms*(2)[*rule-format*,*OF goal1*]]] **by** *assumption*
also have $\ldots \leq B * norm\ y$ **apply**(*rule mult-right-mono*)
using *assms*(3)[*rule-format*,*OF goal1*] **by**(*auto simp add*:*field-simps*)
finally show *?case* **by** *simp* **qed**
have $norm\ (f\ x - f\ y) = norm\ ((f \circ (\lambda u.\ x + u *_R (y - x)))\ 1 - (f \circ (\lambda u.\ x + u *_R (y - x)))\ 0)$
by(*auto simp add*:*norm-minus-commute*)
also have $\ldots \leq norm\ (f'\ (x + u *_R (y - x))\ (y - x))$ **using** *u* **by** *auto*
also have $\ldots \leq B * norm(y - x)$ **apply**(*rule* $**$) **using** $*$ **and** *u* **by** *auto*
finally show *?thesis* **by**(*auto simp add*:*norm-minus-commute*) **qed**

lemma *onorm-vec1*: **fixes** *f*::*real* $\Rightarrow$ *real*
shows $onorm\ (\lambda x.\ vec1\ (f\ (dest\text{-}vec1\ x))) = onorm\ f$ **proof**−
have $\forall x$::*real^1*.* $norm\ x = 1 \longleftrightarrow x \in \{vec1\ -1,\ vec1\ (1::real)\}$ **unfolding** *forall-vec1* **by**(*auto simp add*:*Cart-eq*)
hence *1*:$\{x.\ norm\ x = 1\} = \{vec1\ -1,\ vec1\ (1::real)\}$ **by** *auto*
have *2*:$\{norm\ (vec1\ (f\ (dest\text{-}vec1\ x)))\ |x.\ norm\ x = 1\} = (\lambda x.\ norm\ (vec1\ (f\ (dest\text{-}vec1\ x))))\ `\ \{x.\ norm\ x=1\}$ **by** *auto*
have $\forall x$::*real*.* $norm\ x = 1 \longleftrightarrow x \in \{-1,\ 1\}$ **by** *auto* **hence** *3*:$\{x.\ norm\ x = 1\} = \{-1,\ (1::real)\}$ **by** *auto*
have *4*:$\{norm\ (f\ x)\ |x.\ norm\ x = 1\} = (\lambda x.\ norm\ (f\ x))\ `\ \{x.\ norm\ x=1\}$ **by** *auto*
show *?thesis* **unfolding** *onorm-def 1 2 3 4* **by**(*simp add*:*Sup-finite-Max*) **qed**

lemma *convex-vec1*:$convex\ (vec1\ `\ s) = convex\ (s::real\ set)$
unfolding *convex-def Ball-def forall-vec1* **unfolding** *vec1-dest-vec1-simps image-iff*

by *auto*

lemma *differentiable-bound-real*: **fixes** f::*real* ⇒ *real*
assumes *convex s* ∀ *x*∈*s*. (*f has-derivative f' x*) (*at x within s*) ∀ *x*∈*s*. *onorm*(*f' x*) ≤ *B* **and** *x*:*x*∈*s* **and** *y*:*y*∈*s*
shows *norm*(*f x* − *f y*) ≤ *B* ∗ *norm*(*x* − *y*)
using *differentiable-bound*[*of vec1 ' s vec1* ∘ *f* ∘ *dest-vec1* λ*x*. *vec1* ∘ (*f'* (*dest-vec1 x*)) ∘ *dest-vec1 B vec1 x vec1 y*]
unfolding *Ball-def forall-vec1* **unfolding** *has-derivative-within-vec1-dest-vec1 image-iff*
unfolding *convex-vec1* **unfolding** *o-def vec1-dest-vec1-simps onorm-vec1* **using** *assms* **by** *auto*

22.21 In particular.

lemma *has-derivative-zero-constant*: **fixes** f::*real*⇒*real*
assumes *convex s* ∀ *x*∈*s*. (*f has-derivative* (λ*h*. *0*)) (*at x within s*)
shows ∃ *c*. ∀ *x*∈*s*. *f x* = *c* **proof**(*cases s*={})
case *False* **then obtain** *x* **where** *x*∈*s* **by** *auto*
have ⋀*y*. *y*∈*s* ⟹ *f x* = *f y* **proof**− **case** *goal1*
thus *?case* **using** *differentiable-bound-real*[*OF assms*(*1*−*2*), *of 0 x y*] **and** ‹*x*∈*s*›
unfolding *onorm-vec1*[*of* λ*x*. *0*, *THEN sym*] *onorm-const norm-vec1* **by** *auto*
qed
thus *?thesis* **apply**(*rule-tac x*=*f x* **in** *exI*) **by** *auto* **qed** *auto*

lemma *has-derivative-zero-unique*: **fixes** f::*real*⇒*real*
assumes *convex s a* ∈ *s f a* = *c* ∀ *x*∈*s*. (*f has-derivative* (λ*h*. *0*)) (*at x within s*) *x*∈*s*
shows *f x* = *c* **using** *has-derivative-zero-constant*[*OF assms*(*1*,*4*)] **using** *assms*(*2*−*3*,*5*) **by** *auto*

22.22 Differentiability of inverse function (most basic form).

lemma *has-derivative-inverse-basic*: **fixes** f::*real^'b* ⇒ *real^'c*
assumes (*f has-derivative f'*) (*at* (*g y*)) *bounded-linear g' g'* ∘ *f'* = *id continuous* (*at y*) *g*
open t y ∈ *t* ∀ *z*∈*t*. *f*(*g z*) = *z*
shows (*g has-derivative g'*) (*at y*) **proof**−
interpret *f'*: *bounded-linear f'* **using** *assms* **unfolding** *has-derivative-def* **by** *auto*
interpret *g'*: *bounded-linear g'* **using** *assms* **by** *auto*
guess *C* **using** *bounded-linear.pos-bounded*[*OF assms*(*2*)] **.. note** *C* = *this*

have *lem1*:∀ *e*>*0*. ∃ *d*>*0*. ∀ *z*. *norm*(*z* − *y*) < *d* ⟶ *norm*(*g z* − *g y* − *g'*(*z* − *y*)) ≤ *e* ∗ *norm*(*g z* − *g y*) **proof**(*rule*,*rule*) **case** *goal1*
have ∗:*e* / *C* > *0* **apply**(*rule divide-pos-pos*) **using** ‹*e*>*0*› *C* **by** *auto*
guess *d0* **using** *assms*(*1*)[*unfolded has-derivative-at-alt*,*THEN conjunct2*,*rule-format*,*OF* ∗] **.. note** *d0*=*this*
guess *d1* **using** *assms*(*4*)[*unfolded continuous-at Lim-at*,*rule-format*,*OF d0*[*THEN conjunct1*]] **.. note** *d1*=*this*

guess *d2* **using** *assms*(*5*)[*unfolded open-dist*,*rule-format*,*OF assms*(*6*)] **.. note** *d2*=*this*
guess *d* **using** *real-lbound-gt-zero*[*OF d1*[*THEN conjunct1*] *d2*[*THEN conjunct1*]] **.. note** *d*=*this*
thus *?case* **apply**(*rule-tac x*=*d* **in** *exI*) **apply** *rule* **defer proof**(*rule*,*rule*)
fix *z* **assume** *as*:*norm* (*z* − *y*) < *d* **hence** *z*∈*t* **using** *d2 d* **unfolding** *dist-norm* **by** *auto*
have *norm* (*g z* − *g y* − *g*′ (*z* − *y*)) ≤ *norm* (*g*′ (*f* (*g z*) − *y* − *f*′ (*g z* − *g y*)))
unfolding *g*′.*diff f*′.*diff* **unfolding** *assms*(*3*)[*unfolded o-def id-def*, *THEN fun-cong*]
unfolding *assms*(*7*)[*rule-format*,*OF* ‹*z*∈*t*›] **apply**(*subst norm-minus-cancel*[*THEN sym*]) **by** *auto*
also have ... ≤ *norm*(*f* (*g z*) − *y* − *f*′ (*g z* − *g y*)) ∗ *C* **by**(*rule C*[*THEN conjunct2*,*rule-format*])
also have ... ≤ (*e* / *C*) ∗ *norm* (*g z* − *g y*) ∗ *C* **apply**(*rule mult-right-mono*)
apply(*rule d0*[*THEN conjunct2*,*rule-format*,*unfolded assms*(*7*)[*rule-format*,*OF* ‹*y*∈*t*›]]) **apply**(*cases z*=*y*) **defer**
apply(*rule d1*[*THEN conjunct2*, *unfolded dist-norm*,*rule-format*]) **using** *as d C d0* **by** *auto*
also have ... ≤ *e* ∗ *norm* (*g z* − *g y*) **using** *C* **by**(*auto simp add*:*field-simps*)
finally show *norm* (*g z* − *g y* − *g*′ (*z* − *y*)) ≤ *e* ∗ *norm* (*g z* − *g y*) **by** *simp* **qed** *auto* **qed**
have ∗:(*0*::*real*) < *1* / *2* **by** *auto* **guess** *d* **using** *lem1*[*rule-format*,*OF* ∗] **.. note** *d*=*this* **def** *B*≡*C*∗*2*
have *B*>*0* **unfolding** *B-def* **using** *C* **by** *auto*
have *lem2*:∀ *z*. *norm*(*z* − *y*) < *d* ⟶ *norm*(*g z* − *g y*) ≤ *B* ∗ *norm*(*z* − *y*) **proof**(*rule*,*rule*) **case** *goal1*
have *norm* (*g z* − *g y*) ≤ *norm*(*g*′ (*z* − *y*)) + *norm* ((*g z* − *g y*) − *g*′(*z* − *y*)) **by**(*rule norm-triangle-sub*)
also have ... ≤ *norm*(*g*′ (*z* − *y*)) + *1* / *2* ∗ *norm* (*g z* − *g y*) **apply**(*rule add-left-mono*) **using** *d* **and** *goal1* **by** *auto*
also have ... ≤ *norm* (*z* − *y*) ∗ *C* + *1* / *2* ∗ *norm* (*g z* − *g y*) **apply**(*rule add-right-mono*) **using** *C* **by** *auto*
finally show *?case* **unfolding** *B-def* **by**(*auto simp add*:*field-simps*) **qed**
show *?thesis* **unfolding** *has-derivative-at-alt* **proof**(*rule*,*rule assms*,*rule*,*rule*) **case** *goal1*
hence ∗:*e*/*B* >*0* **apply**−**apply**(*rule divide-pos-pos*) **using** ‹*B*>*0*› **by** *auto*
guess *d*′ **using** *lem1*[*rule-format*,*OF* ∗] **.. note** *d*′=*this*
guess *k* **using** *real-lbound-gt-zero*[*OF d*[*THEN conjunct1*] *d*′[*THEN conjunct1*]] **.. note** *k*=*this*
show *?case* **apply**(*rule-tac x*=*k* **in** *exI*,*rule*) **defer proof**(*rule*,*rule*) **fix** *z* **assume** *as*:*norm*(*z* − *y*) < *k*
hence *norm* (*g z* − *g y* − *g*′ (*z* − *y*)) ≤ *e* / *B* ∗ *norm*(*g z* − *g y*) **using** *d*′ *k* **by** *auto*
also have ... ≤ *e* ∗ *norm*(*z* − *y*) **unfolding** *times-divide-eq-left pos-divide-le-eq*[*OF* ‹*B*>*0*›]
using *lem2*[*THEN spec*[**where** *x*=*z*]] **using** *k as* **using** ‹*e*>*0*› **by**(*auto simp add*:*field-simps*)

finally show *norm* $(g\ z - g\ y - g'\ (z - y)) \leq e * norm\ (z - y)$ **by** *simp*
qed(*insert k*, *auto*) **qed qed**

22.23 Simply rewrite that based on the domain point x.

lemma *has-derivative-inverse-basic-x*: **fixes** f::*real^'b* $\Rightarrow$ *real^'c*
assumes (*f has-derivative f'*) (*at x*) *bounded-linear g' g' o f' = id*
continuous (*at* (*f x*)) *g g*(*f x*) = *x open t f x* $\in$ *t* $\forall$ *y*$\in$*t*. *f*(*g y*) = *y*
shows (*g has-derivative g'*) (*at* (*f*(*x*)))
apply(*rule has-derivative-inverse-basic*) **using** *assms* **by** *auto*

22.24 This is the version in Dieudonne', assuming continuity of f and g.

lemma *has-derivative-inverse-dieudonne*: **fixes** f::*real^'a* $\Rightarrow$ *real^'b*
assumes *open s open* (*f ' s*) *continuous-on s f continuous-on* (*f ' s*) *g* $\forall$ *x*$\in$*s*. *g*(*f x*) = *x*
x$\in$*s* (*f has-derivative f'*) (*at x*) *bounded-linear g' g' o f' = id*
shows (*g has-derivative g'*) (*at* (*f x*))
apply(*rule has-derivative-inverse-basic-x*[*OF assms*(*7*−*9*) - - *assms*(*2*)])
using *assms*(*3*−*6*) **unfolding** *continuous-on-eq-continuous-at*[*OF assms*(*1*)] *continuous-on-eq-continuous-a*
assms(*2*)] **by** *auto*

22.25 Here's the simplest way of not assuming much about g.

lemma *has-derivative-inverse*: **fixes** f::*real^'a* $\Rightarrow$ *real^'b*
assumes *compact s x* $\in$ *s f x* $\in$ *interior*(*f ' s*) *continuous-on s f*
$\forall$ *y*$\in$*s*. *g*(*f y*) = *y* (*f has-derivative f'*) (*at x*) *bounded-linear g' g'* $\circ$ *f' = id*
shows (*g has-derivative g'*) (*at* (*f x*)) **proof**−
{ fix *y* **assume** *y*$\in$*interior* (*f ' s*)
then obtain *x* **where** *x*$\in$*s* **and** $*$:*y* = *f x* **unfolding** *image-iff* **using** *interior-subset*
by *auto*
have *f* (*g y*) = *y* **unfolding** $*$ **and** *assms*(*5*)[*rule-format*,*OF* ‹*x*$\in$*s*›] **.. } note**
$* = this$
show *?thesis* **apply**(*rule has-derivative-inverse-basic-x*[*OF assms*(*6*−*8*)])
apply(*rule continuous-on-interior*[*OF* - *assms*(*3*)]) **apply**(*rule continuous-on-inverse*[*OF*
assms(*4*,*1*)])
apply(*rule assms*(*2*,*5*) *assms*(*5*)[*rule-format*] *open-interior assms*(*3*))+ **by**(*rule*,
rule $*$, *assumption*) **qed**

22.26 Proving surjectivity via Brouwer fixpoint theorem.

lemma *brouwer-surjective*: **fixes** f::*real^'n* $\Rightarrow$ *real^'n*
assumes *compact t convex t t* $\neq$ {} *continuous-on t f*
$\forall$ *x*$\in$*s*. $\forall$ *y*$\in$*t*. *x* + (*y* − *f y*) $\in$ *t x*$\in$*s*
shows $\exists$ *y*$\in$*t*. *f y* = *x* **proof**−
have $*$:$\bigwedge$*x y*. *f y* = *x* $\longleftrightarrow$ *x* + (*y* − *f y*) = *y* **by**(*auto simp add*:*algebra-simps*)
show *?thesis* **unfolding** $*$ **apply**(*rule brouwer*[*OF assms*(*1*−*3*), *of* $\lambda y.$ *x* + (*y*
− *f y*)])

apply(*rule continuous-on-intros assms*)+ **using** *assms*(*4*−*6*) **by** *auto* **qed**

lemma *brouwer-surjective-cball*: **fixes** *f*::*real^'n* ⇒ *real^'n*
assumes *0* < *e continuous-on* (*cball a e*) *f*
∀ *x*∈*s*. ∀ *y*∈*cball a e*. *x* + (*y* − *f y*) ∈ *cball a e x*∈*s*
shows ∃ *y*∈*cball a e*. *f y* = *x* **apply**(*rule brouwer-surjective*) **apply**(*rule compact-cball convex-cball*)+
unfolding *cball-eq-empty* **using** *assms* **by** *auto*

See Sussmann: "Multidifferential calculus", Theorem 2.1.1

lemma *sussmann-open-mapping*: **fixes** *f*::*real^'a* ⇒ *real^'b*
assumes *open s continuous-on s f x* ∈ *s*
(*f has-derivative f'*) (*at x*) *bounded-linear g' f'* ∘ *g'* = *id*
t ⊆ *s x* ∈ *interior t*
shows *f x* ∈ *interior* (*f* ‘ *t*) **proof**−
interpret *f'*:*bounded-linear f'* **using** *assms* **unfolding** *has-derivative-def* **by** *auto*
interpret *g'*:*bounded-linear g'* **using** *assms* **by** *auto*
guess *B* **using** *bounded-linear.pos-bounded*[*OF assms*(*5*)] **.. note** *B*=*this* **hence** ∗:*1*/(*2*∗*B*)>*0* **by**(*auto intro*!: *divide-pos-pos*)
guess *e0* **using** *assms*(*4*)[*unfolded has-derivative-at-alt*,*THEN conjunct2*,*rule-format*,*OF* ∗] **.. note** *e0*=*this*
guess *e1* **using** *assms*(*8*)[*unfolded mem-interior-cball*] **.. note** *e1*=*this*
have ∗:*0*<*e0*/*B* *0*<*e1*/*B* **apply**(*rule-tac*[!] *divide-pos-pos*) **using** *e0 e1 B* **by** *auto*
guess *e* **using** *real-lbound-gt-zero*[*OF* ∗] **.. note** *e*=*this*
have ∀ *z*∈*cball* (*f x*) (*e*/*2*). ∃ *y*∈*cball* (*f x*) *e*. *f* (*x* + *g'* (*y* − *f x*)) = *z*
apply(*rule*,*rule brouwer-surjective-cball*[**where** *s*=*cball* (*f x*) (*e*/*2*)])
prefer *3* **apply**(*rule*,*rule*) **proof**−
show *continuous-on* (*cball* (*f x*) *e*) (λ*y*. *f* (*x* + *g'* (*y* − *f x*))) **unfolding** *g'.diff*
apply(*rule continuous-on-compose*[*of - - f*, *unfolded o-def*])
apply(*rule continuous-on-intros linear-continuous-on*[*OF assms*(*5*)])+
apply(*rule continuous-on-subset*[*OF assms*(*2*)]) **apply**(*rule*,*unfold image-iff*,*erule bexE*) **proof**−
fix *y z* **assume** *as*:*y* ∈*cball* (*f x*) *e* *z* = *x* + (*g' y* − *g'* (*f x*))
have *dist x z* = *norm* (*g'* (*f x*) − *g' y*) **unfolding** *as*(*2*) **and** *dist-norm* **by** *auto*
also have ... ≤ *norm* (*f x* − *y*) ∗ *B* **unfolding** *g'.diff*[*THEN sym*] **using** *B* **by** *auto*
also have ... ≤ *e* ∗ *B* **using** *as*(*1*)[*unfolded mem-cball dist-norm*] **using** *B* **by** *auto*
also have ... ≤ *e1* **using** *e* **unfolding** *less-divide-eq* **using** *B* **by** *auto*
finally have *z*∈*cball x e1* **unfolding** *mem-cball* **by** *force*
thus *z* ∈ *s* **using** *e1 assms*(*7*) **by** *auto* **qed next**
fix *y z* **assume** *as*:*y* ∈ *cball* (*f x*) (*e* / *2*) *z* ∈ *cball* (*f x*) *e*
have *norm* (*g'* (*z* − *f x*)) ≤ *norm* (*z* − *f x*) ∗ *B* **using** *B* **by** *auto*
also have ... ≤ *e* ∗ *B* **apply**(*rule mult-right-mono*) **using** *as*(*2*)[*unfolded mem-cball dist-norm*] **and** *B* **unfolding** *norm-minus-commute* **by** *auto*
also have ... < *e0* **using** *e* **and** *B* **unfolding** *less-divide-eq* **by** *auto*

finally have ∗:*norm* (*x* + *g*′ (*z* − *f x*) − *x*) < *e0* **by** *auto*
have ∗∗:*f x* + *f*′ (*x* + *g*′ (*z* − *f x*) − *x*) = *z* **using** *assms*(*6*)[*unfolded o-def id-def*,*THEN cong*] **by** *auto*
have *norm* (*f x* − (*y* + (*z* − *f* (*x* + *g*′ (*z* − *f x*))))) ≤ *norm* (*f* (*x* + *g*′ (*z* − *f x*)) − *z*) + *norm* (*f x* − *y*)
using *norm-triangle-ineq*[*of f* (*x* + *g*′(*z* − *f x*)) − *z f x* − *y*] **by**(*auto simp add*:*algebra-simps*)
also have ... ≤ *1* / (*B* ∗ *2*) ∗ *norm* (*g*′ (*z* − *f x*)) + *norm* (*f x* − *y*) **using** *e0*[*THEN conjunct2*,*rule-format*,*OF* ∗] **unfolding** *algebra-simps* ∗∗ **by** *auto*
also have ... ≤ *1* / (*B* ∗ *2*) ∗ *norm* (*g*′ (*z* − *f x*)) + *e/2* **using** *as*(*1*)[*unfolded mem-cball dist-norm*] **by** *auto*
also have ... ≤ *1* / (*B* ∗ *2*) ∗ *B* ∗ *norm* (*z* − *f x*) + *e/2* **using** ∗ **and** *B* **by**(*auto simp add*:*field-simps*)
also have ... ≤ *1* / *2* ∗ *norm* (*z* − *f x*) + *e/2* **by** *auto*
also have ... ≤ *e/2* + *e/2* **apply**(*rule add-right-mono*) **using** *as*(*2*)[*unfolded mem-cball dist-norm*] **unfolding** *norm-minus-commute* **by** *auto*
finally show *y* + (*z* − *f* (*x* + *g*′ (*z* − *f x*))) ∈ *cball* (*f x*) *e* **unfolding** *mem-cball dist-norm* **by** *auto*
qed(*insert e*, *auto*) **note** *lem* = *this*
show *?thesis* **unfolding** *mem-interior* **apply**(*rule-tac x*=*e/2* **in** *exI*)
apply(*rule*,*rule divide-pos-pos*) **prefer** *3* **proof**
fix *y* **assume** *y* ∈ *ball* (*f x*) (*e/2*) **hence** ∗:*y*∈*cball* (*f x*) (*e/2*) **by** *auto*
guess *z* **using** *lem*[*rule-format*,*OF* ∗] **.. note** *z*=*this*
hence *norm* (*g*′ (*z* − *f x*)) ≤ *norm* (*z* − *f x*) ∗ *B* **using** *B* **by**(*auto simp add*:*field-simps*)
also have ... ≤ *e* ∗ *B* **apply**(*rule mult-right-mono*) **using** *z*(*1*) **unfolding** *mem-cball dist-norm norm-minus-commute* **using** *B* **by** *auto*
also have ... ≤ *e1* **using** *e B* **unfolding** *less-divide-eq* **by** *auto*
finally have *x* + *g*′(*z* − *f x*) ∈ *t* **apply**− **apply**(*rule e1*[*THEN conjunct2*,*unfolded subset-eq*,*rule-format*])
unfolding *mem-cball dist-norm* **by** *auto*
thus *y* ∈ *f* ‘ *t* **using** *z* **by** *auto* **qed**(*insert e*, *auto*) **qed**

Hence the following eccentric variant of the inverse function theorem. *) (* This has no continuity assumptions, but we do need the inverse function. *) (* We could put f' o g = I but this happens to fit with the minimal linear *) (* algebra theory I've set up so far.

lemma *has-derivative-inverse-strong*: **fixes** *f*::*real*^*'n* ⇒ *real*^*'n*
assumes *open s x* ∈ *s continuous-on s f*
∀ *x*∈*s*. *g*(*f x*) = *x* (*f has-derivative f'*) (*at x*) *f*′ *o g*′ = *id*
shows (*g has-derivative g'*) (*at* (*f x*)) **proof**−
have *linf*:*bounded-linear f*′ **using** *assms*(*5*) **unfolding** *has-derivative-def* **by** *auto*
hence *ling*:*bounded-linear g*′ **unfolding** *linear-conv-bounded-linear*[*THEN sym*]
apply− **apply**(*rule right-inverse-linear*) **using** *assms*(*6*) **by** *auto*
moreover have *g*′ ∘ *f*′ = *id* **using** *assms*(*6*) *linf ling* **unfolding** *linear-conv-bounded-linear*[*THEN sym*]
using *linear-inverse-left* **by** *auto*
moreover have ∗:∀ *t*⊆*s*. *x*∈*interior t* ⟶ *f x* ∈ *interior* (*f* ‘ *t*) **apply**(*rule*,*rule*,*rule*,*rule*

sussmann-open-mapping)
apply(*rule assms ling*)+ **by** *auto*
have *continuous* (*at* (*f x*)) *g* **unfolding** *continuous-at Lim-at* **proof**(*rule*,*rule*)
fix *e*::*real* **assume** *e*>*0*
hence *f x* ∈ *interior* (*f* ‘ (*ball x e* ∩ *s*)) **using** ∗[*rule-format*,*of ball x e* ∩ *s*] ‹*x*∈*s*›
by(*auto simp add*: *interior-open*[*OF open-ball*] *interior-open*[*OF assms*(*1*)])
then guess *d* **unfolding** *mem-interior* **..** **note** *d*=*this*
show ∃ *d*>*0*. ∀ *y*. *0* < *dist y* (*f x*) ∧ *dist y* (*f x*) < *d* ⟶ *dist* (*g y*) (*g* (*f x*)) < *e*
apply(*rule-tac x*=*d* **in** *exI*) **apply**(*rule*,*rule d*[*THEN conjunct1*]) **proof**(*rule*,*rule*)
case *goal1*
hence *g y* ∈ *g* ‘ *f* ‘ (*ball x e* ∩ *s*) **using** *d*[*THEN conjunct2*,*unfolded subset-eq*,*THEN bspec*[**where** *x*=*y*]]
by(*auto simp add*:*dist-commute*)
hence *g y* ∈ *ball x e* ∩ *s* **using** *assms*(*4*) **by** *auto*
thus *dist* (*g y*) (*g* (*f x*)) < *e* **using** *assms*(*4*)[*rule-format*,*OF* ‹*x*∈*s*›] **by**(*auto simp add*:*dist-commute*) **qed** **qed**
moreover have *f x* ∈ *interior* (*f* ‘ *s*) **apply**(*rule sussmann-open-mapping*)
apply(*rule assms ling*)+ **using** *interior-open*[*OF assms*(*1*)] **and** ‹*x*∈*s*› **by** *auto*
moreover have ⋀*y*. *y* ∈ *interior* (*f* ‘ *s*) ⟹ *f* (*g y*) = *y* **proof**− **case** *goal1*
hence *y*∈*f* ‘ *s* **using** *interior-subset* **by** *auto* **then guess** *z* **unfolding** *image-iff* **..**
thus *?case* **using** *assms*(*4*) **by** *auto* **qed**
ultimately show *?thesis* **apply**− **apply**(*rule has-derivative-inverse-basic-x*[*OF assms*(*5*)]) **using** *assms* **by** *auto* **qed**

22.27 A rewrite based on the other domain.

lemma *has-derivative-inverse-strong-x*: **fixes** *f*::*real*^*'n* ⇒ *real*^*'n*
assumes *open s g y* ∈ *s continuous-on s f*
∀ *x*∈*s*. *g*(*f x*) = *x* (*f has-derivative f'*) (*at* (*g y*)) *f' o g'* = *id f*(*g y*) = *y*
shows (*g has-derivative g'*) (*at y*)
using *has-derivative-inverse-strong*[*OF assms*(*1*−*6*)] **unfolding** *assms*(*7*) **by** *simp*

22.28 On a region.

lemma *has-derivative-inverse-on*: **fixes** *f*::*real*^*'n* ⇒ *real*^*'n*
assumes *open s* ∀ *x*∈*s*. (*f has-derivative f'*(*x*)) (*at x*) ∀ *x*∈*s*. *g*(*f x*) = *x f'*(*x*) *o* *g'*(*x*) = *id x*∈*s*
shows (*g has-derivative g'*(*x*)) (*at* (*f x*))
apply(*rule has-derivative-inverse-strong*[**where** *g'*=*g' x* **and** *f*=*f*]) **apply**(*rule assms*)+
unfolding *continuous-on-eq-continuous-at*[*OF assms*(*1*)]
apply(*rule*,*rule differentiable-imp-continuous-at*) **unfolding** *differentiable-def* **using** *assms* **by** *auto*

22.29 Invertible derivative continous at a point implies local injectivity. *) (* It's only for this we need continuity of the derivative, except of course *) (* if we want the fact that the inverse derivative is also continuous. So if *) (* we know for some other reason that the inverse function exists, it's OK.

lemma *bounded-linear-sub*: *bounded-linear f* $\Longrightarrow$ *bounded-linear g* ==> *bounded-linear* $(\lambda x.\ f\ x - g\ x)$
using *bounded-linear-add*[*of f* $\lambda x.\ -\ g\ x$] *bounded-linear-minus*[*of g*] **by**(*auto simp add*:*algebra-simps*)

lemma *has-derivative-locally-injective*: **fixes** *f*::*real^'n* $\Rightarrow$ *real^'m*
assumes $a \in s$ *open s bounded-linear g' g' o f'*$(a) = id$
$\forall x{\in}s.$ (*f has-derivative f'*(x)) (*at x*)
$\forall e{>}0.\ \exists d{>}0.\ \forall x.$ *dist a x* $<$ *d* $\longrightarrow$ *onorm*$(\lambda v.\ f'\ x\ v - f'\ a\ v) < e$
obtains *t* **where** $a \in t$ *open t* $\forall x{\in}t.\ \forall x'{\in}t.\ (f\ x' = f\ x) \longrightarrow (x' = x)$ **proof**−
interpret *bounded-linear g'* **using** *assms* **by** *auto*
note *f'g'* = *assms*(*4*)[*unfolded id-def o-def*,*THEN cong*]
have *g'* (*f' a 1*) = *1* **using** *f'g'* **by** *auto*
hence ∗:*0* < *onorm g'* **unfolding** *onorm-pos-lt*[*OF assms*(*3*)[*unfolded linear-linear*]] **by** *fastsimp*
def *k* ≡ *1* / *onorm g'* / *2* **have** ∗:*k*>*0* **unfolding** *k-def* **using** ∗ **by** *auto*
guess *d1* **using** *assms*(*6*)[*rule-format*,*OF* ∗] **..** **note** *d1*=*this*
from ‹*open s*› **obtain** *d2* **where** *d2*>*0 ball a d2* ⊆ *s* **using** ‹*a*∈*s*› **..**
obtain *d2* **where** *d2*>*0 ball a d2* ⊆ *s* **using** *assms*(*2*,*1*) **..**
guess *d2* **using** *assms*(*2*)[*unfolded open-contains-ball*,*rule-format*,*OF* ‹*a*∈*s*›] **..** **note** *d2*=*this*
guess *d* **using** *real-lbound-gt-zero*[*OF d1*[*THEN conjunct1*] *d2*[*THEN conjunct1*]] **..** **note** *d* = *this*
show *?thesis* **proof** **show** *a*∈*ball a d* **using** *d* **by** *auto*
show $\forall x{\in}$*ball a d*. $\forall x'{\in}$*ball a d*. *f x'* = *f x* $\longrightarrow$ *x'* = *x* **proof**(*intro strip*)
fix *x y* **assume** *as*:*x*∈*ball a d y*∈*ball a d f x* = *f y*
def *ph* ≡ $\lambda w.\ w - g'(f\ w - f\ x)$ **have** *ph'*:*ph* = *g'* ∘ $(\lambda w.\ f'\ a\ w - (f\ w - f\ x))$
unfolding *ph-def o-def* **unfolding** *diff* **using** *f'g'* **by**(*auto simp add*:*algebra-simps*)
have *norm* (*ph x* − *ph y*) ≤ (*1*/*2*) ∗ *norm* (*x* − *y*)
apply(*rule differentiable-bound*[*OF convex-ball - - as*(*1*−*2*), **where** *f'*=$\lambda x\ v.\ v - g'(f'\ x\ v)$])
apply(*rule-tac*[!] *ballI*) **proof**− **fix** *u* **assume** *u*:*u* ∈ *ball a d* **hence** *u*∈*s* **using** *d d2* **by** *auto*
have ∗:$(\lambda v.\ v - g'\ (f'\ u\ v)) = g' \circ (\lambda w.\ f'\ a\ w - f'\ u\ w)$ **unfolding** *o-def* **and** *diff* **using** *f'g'* **by** *auto*
show (*ph has-derivative* $(\lambda v.\ v - g'\ (f'\ u\ v)))$ (*at u within ball a d*)
unfolding *ph'* ∗ **apply**(*rule diff-chain-within*) **defer** **apply**(*rule bounded-linear.has-derivative*[*OF assms*(*3*)])
apply(*rule has-derivative-intros*) **defer** **apply**(*rule has-derivative-sub*[**where** *g'*=$\lambda x.0$,*unfolded diff-0-right*])
apply(*rule has-derivative-at-within*) **using** *assms*(*5*) **and** ‹*u*∈*s*› ‹*a*∈*s*›

by(*auto intro*!: *has-derivative-intros derivative-linear*)
have ∗∗:*bounded-linear* (λ*x*. *f* ′ *u x* − *f* ′ *a x*) *bounded-linear* (λ*x*. *f* ′ *a x* − *f* ′ *u x*) **apply**(*rule-tac*[!] *bounded-linear-sub*)
apply(*rule-tac*[!] *derivative-linear*) **using** *assms*(*5*) ‹*u*∈*s*› ‹*a*∈*s*› **by** *auto*
have *onorm* (λ*v*. *v* − *g* ′ (*f* ′ *u v*)) ≤ *onorm g* ′ ∗ *onorm* (λ*w*. *f* ′ *a w* − *f* ′ *u w*) **unfolding** ∗ **apply**(*rule onorm-compose*)
unfolding *linear-conv-bounded-linear* **by**(*rule assms*(*3*) ∗∗)+
also have ... ≤ *onorm g* ′ ∗ *k* **apply**(*rule mult-left-mono*)
using *d1*[*THEN conjunct2*,*rule-format*,*of u*] **using** *onorm-neg*[*OF* ∗∗(*1*)[*unfolded linear-linear*]]
using *d* **and** *u* **and** *onorm-pos-le*[*OF assms*(*3*)[*unfolded linear-linear*]] **by**(*auto simp add*:*algebra-simps*)
also have ... ≤ *1/2* **unfolding** *k-def* **by** *auto*
finally show *onorm* (λ*v*. *v* − *g* ′ (*f* ′ *u v*)) ≤ *1 / 2* **by** *assumption* **qed**
moreover have *norm* (*ph y* − *ph x*) = *norm* (*y* − *x*) **apply**(*rule arg-cong*[**where** *f*=*norm*])
unfolding *ph-def* **using** *diff* **unfolding** *as* **by** *auto*
ultimately show *x* = *y* **unfolding** *norm-minus-commute* **by** *auto* **qed qed** *auto* **qed**

22.30 Uniformly convergent sequence of derivatives.

lemma *has-derivative-sequence-lipschitz-lemma*: **fixes** *f*::*nat* ⇒ *real*^′*m* ⇒ *real*^′*n*
assumes *convex s* ∀ *n*. ∀ *x*∈*s*. ((*f n*) *has-derivative* (*f* ′ *n x*)) (*at x within s*)
∀ *n*≥*N*. ∀ *x*∈*s*. ∀ *h*. *norm*(*f* ′ *n x h* − *g* ′ *x h*) ≤ *e* ∗ *norm*(*h*)
shows ∀ *m*≥*N*. ∀ *n*≥*N*. ∀ *x*∈*s*. ∀ *y*∈*s*. *norm*((*f m x* − *f n x*) − (*f m y* − *f n y*)) ≤ *2* ∗ *e* ∗ *norm*(*x* − *y*) **proof**(*default*)+
fix *m n x y* **assume** *as*:*N*≤*m N*≤*n x*∈*s y*∈*s*
show *norm*((*f m x* − *f n x*) − (*f m y* − *f n y*)) ≤ *2* ∗ *e* ∗ *norm*(*x* − *y*)
apply(*rule differentiable-bound*[**where** *f*′=λ*x h*. *f* ′ *m x h* − *f* ′ *n x h*, *OF assms*(*1*) - - *as*(*3*−*4*)]) **apply**(*rule-tac*[!] *ballI*) **proof**−
fix *x* **assume** *x*∈*s* **show** ((λ*a*. *f m a* − *f n a*) *has-derivative* (λ*h*. *f* ′ *m x h* − *f* ′ *n x h*)) (*at x within s*)
by(*rule has-derivative-intros assms*(*2*)[*rule-format*] ‹*x*∈*s*›)+
{ **fix** *h* **have** *norm* (*f* ′ *m x h* − *f* ′ *n x h*) ≤ *norm* (*f* ′ *m x h* − *g* ′ *x h*) + *norm* (*f* ′ *n x h* − *g* ′ *x h*)
using *norm-triangle-ineq*[*of f* ′ *m x h* − *g* ′ *x h* − *f* ′ *n x h* + *g* ′ *x h*] **unfolding** *norm-minus-commute* **by**(*auto simp add*:*algebra-simps*)
also have ... ≤ *e* ∗ *norm h*+ *e* ∗ *norm h* **using** *assms*(*3*)[*rule-format*,*OF* ‹*N*≤*m*› ‹*x*∈*s*›, *of h*] *assms*(*3*)[*rule-format*,*OF* ‹*N*≤*n*› ‹*x*∈*s*›, *of h*]
by(*auto simp add*:*field-simps*)
finally have *norm* (*f* ′ *m x h* − *f* ′ *n x h*) ≤ *2* ∗ *e* ∗ *norm h* **by** *auto* }
thus *onorm* (λ*h*. *f* ′ *m x h* − *f* ′ *n x h*) ≤ *2* ∗ *e* **apply**−**apply**(*rule onorm*(*2*)) **apply**(*rule linear-compose-sub*)
unfolding *linear-conv-bounded-linear* **using** *assms*(*2*)[*rule-format*,*OF* ‹*x*∈*s*›, *THEN derivative-linear*] **by** *auto* **qed qed**

lemma *has-derivative-sequence-lipschitz*: **fixes** *f*::*nat* ⇒ *real*^′*m* ⇒ *real*^′*n*
assumes *convex s* ∀ *n*. ∀ *x*∈*s*. ((*f n*) *has-derivative* (*f* ′ *n x*)) (*at x within s*)

$\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall h.\ norm(f'\ n\ x\ h - g'\ x\ h) \leq e * norm(h)\ 0 < e$
shows $\forall e{>}0.\ \exists N.\ \forall m{\geq}N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall y{\in}s.\ norm((f\ m\ x - f\ n\ x) - (f\ m\ y - f\ n\ y)) \leq e * norm(x - y)$ **proof**(*rule*,*rule*)
case *goal1* **have** $*$:$2 * (1/2 * e) = e\ 1/2 * e > 0$ **using** ‹$e{>}0$› **by** *auto*
guess N **using** *assms*(3)[*rule-format*,*OF* $*$(2)] **..**
thus *?case* **apply**(*rule-tac* $x{=}N$ **in** *exI*) **apply**(*rule has-derivative-sequence-lipschitz-lemma*[**where** $e{=}1/2 * e$, *unfolded* $*$]) **using** *assms* **by** *auto* **qed**

lemma *has-derivative-sequence*: **fixes** f::*nat*⇒*real*^'*m*⇒*real*^'*n*
assumes *convex* s $\forall n.\ \forall x{\in}s.\ ((f\ n)\ has\text{-}derivative\ (f'\ n\ x))\ (at\ x\ within\ s)$
$\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall h.\ norm(f'\ n\ x\ h - g'\ x\ h) \leq e * norm(h)$
$x0 \in s\ \ ((\lambda n.\ f\ n\ x0) \longrightarrow l)\ sequentially$
shows $\exists g.\ \forall x{\in}s.\ ((\lambda n.\ f\ n\ x) \longrightarrow g\ x)\ sequentially \wedge (g\ has\text{-}derivative\ g'(x))\ (at\ x\ within\ s)$ **proof**−
have *lem1*:$\forall e{>}0.\ \exists N.\ \forall m{\geq}N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall y{\in}s.\ norm((f\ m\ x - f\ n\ x) - (f\ m\ y - f\ n\ y)) \leq e * norm(x - y)$
apply(*rule has-derivative-sequence-lipschitz*[**where** $e{=}42$::*nat*]) **apply**(*rule assms*)+ **by** *auto*
have $\exists g.\ \forall x{\in}s.\ ((\lambda n.\ f\ n\ x) \longrightarrow g\ x)\ sequentially$ **apply**(*rule bchoice*) **unfolding** *convergent-eq-cauchy* **proof**
fix x **assume** $x{\in}s$ **show** *Cauchy* $(\lambda n.\ f\ n\ x)$ **proof**(*cases* $x{=}x0$)
case *True* **thus** *?thesis* **using** *convergent-imp-cauchy*[*OF assms*(5)] **by** *auto*
next
case *False* **show** *?thesis* **unfolding** *Cauchy-def* **proof**(*rule*,*rule*)
fix e::*real* **assume** $e{>}0$ **hence** $*$:$e/2{>}0\ e/2/norm(x{-}x0){>}0$ **using** *False* **by**(*auto intro*!:*divide-pos-pos*)
guess M **using** *convergent-imp-cauchy*[*OF assms*(5), *unfolded Cauchy-def*, *rule-format*,*OF* $*$(1)] **..** **note** $M{=}this$
guess N **using** *lem1*[*rule-format*,*OF* $*$(2)] **..** **note** $N = this$
show $\exists M.\ \forall m{\geq}M.\ \forall n{\geq}M.\ dist\ (f\ m\ x)\ (f\ n\ x) < e$ **apply**(*rule-tac* $x{=}max\ M\ N$ **in** *exI*) **proof**(*default*+)
fix $m\ n$ **assume** *as*:$max\ M\ N \leq m\ max\ M\ N \leq n$
have $dist\ (f\ m\ x)\ (f\ n\ x) \leq norm\ (f\ m\ x0 - f\ n\ x0) + norm\ (f\ m\ x - f\ n\ x - (f\ m\ x0 - f\ n\ x0))$
unfolding *dist-norm* **by**(*rule norm-triangle-sub*)
also have $\ldots \leq norm\ (f\ m\ x0 - f\ n\ x0) + e\ /\ 2$ **using** N[*rule-format*,*OF* - - ‹$x{\in}s$› ‹$x0{\in}s$›, *of* $m\ n$] **and** *as* **and** *False* **by** *auto*
also have $\ldots < e\ /\ 2 + e\ /\ 2$ **apply**(*rule add-strict-right-mono*) **using** *as* **and** M[*rule-format*] **unfolding** *dist-norm* **by** *auto*
finally show $dist\ (f\ m\ x)\ (f\ n\ x) < e$ **by** *auto* **qed qed qed qed**
then guess g **..** **note** $g = this$
have *lem2*:$\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall y{\in}s.\ norm((f\ n\ x - f\ n\ y) - (g\ x - g\ y)) \leq e * norm(x - y)$ **proof**(*rule*,*rule*)
fix e::*real* **assume** $*$:$e{>}0$ **guess** N **using** *lem1*[*rule-format*,*OF* $*$] **..** **note** $N{=}this$
show $\exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall y{\in}s.\ norm\ (f\ n\ x - f\ n\ y - (g\ x - g\ y)) \leq e * norm\ (x - y)$ **apply**(*rule-tac* $x{=}N$ **in** *exI*) **proof**(*default*+)
fix $n\ x\ y$ **assume** *as*:$N \leq n\ x \in s\ y \in s$
have *eventually* $(\lambda xa.\ norm\ (f\ n\ x - f\ n\ y - (f\ xa\ x - f\ xa\ y)) \leq e * norm$

```
(x − y)) sequentially
    unfolding eventually-sequentially apply(rule-tac x=N in exI) proof(rule,rule)
      fix m assume N≤m thus norm (f n x − f n y − (f m x − f m y)) ≤ e ∗
norm (x − y)
       using N[rule-format, of n m x y] and as by(auto simp add:algebra-simps)
qed
    thus norm (f n x − f n y − (g x − g y)) ≤ e ∗ norm (x − y) apply−
      apply(rule Lim-norm-ubound[OF trivial-limit-sequentially, where f=λm.
(f n x − f n y) − (f m x − f m y)])
     apply(rule Lim-sub Lim-const g[rule-format] as)+ by assumption qed qed
  show ?thesis unfolding has-derivative-within-alt apply(rule-tac x=g in exI)
   apply(rule,rule,rule g[rule-format],assumption) proof fix x assume x∈s
  have lem3:∀ u. ((λn. f' n x u) −−−> g' x u) sequentially unfolding Lim-sequentially
proof(rule,rule,rule)
    fix u and e::real assume e>0 show ∃ N. ∀ n≥N. dist (f' n x u) (g' x u) <
e proof(cases u=0)
      case True guess N using assms(3)[rule-format,OF ‹e>0›] .. note N=this
      show ?thesis apply(rule-tac x=N in exI) unfolding True
        using N[rule-format,OF - ‹x∈s›,of - 0] and ‹e>0› by auto next
        case False hence ∗:e / 2 / norm u > 0 using ‹e>0› by(auto intro!:
divide-pos-pos)
      guess N using assms(3)[rule-format,OF ∗] .. note N=this
      show ?thesis apply(rule-tac x=N in exI) proof(rule,rule) case goal1
        show ?case unfolding dist-norm using N[rule-format,OF goal1 ‹x∈s›,
of u] False ‹e>0›
          by (auto simp add:field-simps) qed qed qed
  show bounded-linear (g' x) unfolding linear-linear linear-def apply(rule,rule,rule)
defer proof(rule,rule)
    fix x' y z::real^'m and c::real
    note lin = assms(2)[rule-format,OF ‹x∈s›,THEN derivative-linear]
   show g' x (c ∗R x') = c ∗R g' x x' apply(rule Lim-unique[OF trivial-limit-sequentially])
      apply(rule lem3[rule-format]) unfolding smult-conv-scaleR
      unfolding lin[unfolded bounded-linear-def bounded-linear-axioms-def,THEN
conjunct2,THEN conjunct1,rule-format]
      apply(rule Lim-cmul) by(rule lem3[rule-format])
   show g' x (y + z) = g' x y + g' x z apply(rule Lim-unique[OF trivial-limit-sequentially])
       apply(rule lem3[rule-format]) unfolding lin[unfolded bounded-linear-def
additive-def,THEN conjunct1,rule-format]
      apply(rule Lim-add) by(rule lem3[rule-format])+ qed
  show ∀ e>0. ∃ d>0. ∀ y∈s. norm (y − x) < d ⟶ norm (g y − g x − g' x (y
− x)) ≤ e ∗ norm (y − x) proof(rule,rule) case goal1
    have ∗:e/3>0 using goal1 by auto guess N1 using assms(3)[rule-format,OF
∗] .. note N1=this
    guess N2 using lem2[rule-format,OF ∗] .. note N2=this
    guess d1 using assms(2)[unfolded has-derivative-within-alt, rule-format,OF
‹x∈s›, of max N1 N2,THEN conjunct2,rule-format,OF ∗] .. note d1=this
      show ?case apply(rule-tac x=d1 in exI) apply(rule,rule d1[THEN con-
junct1]) proof(rule,rule)
      fix y assume as:y ∈ s norm (y − x) < d1 let ?N =max N1 N2
```

have *norm* $(g\ y - g\ x - (f\ ?N\ y - f\ ?N\ x)) \leq e\ /3 * norm\ (y - x)$ **apply**(*subst norm-minus-cancel*[*THEN sym*])
using *N2*[*rule-format*, *OF* - ‹$y \in s$› ‹$x \in s$›, *of ?N*] **by** *auto* **moreover**
have *norm* $(f\ ?N\ y - f\ ?N\ x - f'\ ?N\ x\ (y - x)) \leq e\ /\ 3 * norm\ (y - x)$ **using** *d1* **and** *as* **by** *auto* **ultimately**
have *norm* $(g\ y - g\ x - f'\ ?N\ x\ (y - x)) \leq 2 * e\ /\ 3 * norm\ (y - x)$
using *norm-triangle-le*[*of* $g\ y - g\ x - (f\ ?N\ y - f\ ?N\ x)$ $f\ ?N\ y - f\ ?N\ x - f'\ ?N\ x\ (y - x)$ $2 * e\ /\ 3 * norm\ (y - x)$]
by (*auto simp add*:*algebra-simps*) **moreover**
have *norm* $(f'\ ?N\ x\ (y - x) - g'\ x\ (y - x)) \leq e\ /\ 3 * norm\ (y - x)$ **using** *N1* ‹$x \in s$› **by** *auto*
ultimately show *norm* $(g\ y - g\ x - g'\ x\ (y - x)) \leq e * norm\ (y - x)$
using *norm-triangle-le*[*of* $g\ y - g\ x - f'\ (max\ N1\ N2)\ x\ (y - x)$ $f'\ (max\ N1\ N2)\ x\ (y - x) - g'\ x\ (y - x)$] **by**(*auto simp add*:*algebra-simps*)
qed qed qed qed

22.31 Can choose to line up antiderivatives if we want.

lemma *has-antiderivative-sequence*: **fixes** f::$nat \Rightarrow real\hat{}'m \Rightarrow real\hat{}'n$
assumes *convex s* $\forall n.\ \forall x \in s.\ ((f\ n)\ has\text{-}derivative\ (f'\ n\ x))\ (at\ x\ within\ s)$
$\forall e > 0.\ \exists N.\ \forall n \geq N.\ \forall x \in s.\ \forall h.\ norm(f'\ n\ x\ h - g'\ x\ h) \leq e * norm\ h$
shows $\exists g.\ \forall x \in s.\ (g\ has\text{-}derivative\ g'(x))\ (at\ x\ within\ s)$ **proof**(*cases* $s = \{\}$)
case *False* **then obtain** *a* **where** $a \in s$ **by** *auto* **have** $*$:$\bigwedge P\ Q.\ \exists g.\ \forall x \in s.\ P\ g\ x \wedge Q\ g\ x \Longrightarrow \exists g.\ \forall x \in s.\ Q\ g\ x$ **by** *auto*
show *?thesis* **apply**(*rule* $*$) **apply**(*rule has-derivative-sequence*[*OF assms*(*1*) - *assms*(*3*), *of* $\lambda n\ x.\ f\ n\ x + (f\ 0\ a - f\ n\ a)$])
apply(*rule*,*rule*) **apply**(*rule has-derivative-add-const*, *rule assms*(*2*)[*rule-format*], *assumption*)
apply(*rule* ‹$a \in s$›) **by**(*auto intro*!: *Lim-const*) **qed** *auto*

lemma *has-antiderivative-limit*: **fixes** g'::$real\hat{}'m \Rightarrow real\hat{}'m \Rightarrow real\hat{}'n$
assumes *convex s* $\forall e > 0.\ \exists f\ f'.\ \forall x \in s.\ (f\ has\text{-}derivative\ (f'\ x))\ (at\ x\ within\ s) \wedge (\forall h.\ norm(f'\ x\ h - g'\ x\ h) \leq e * norm(h))$
shows $\exists g.\ \forall x \in s.\ (g\ has\text{-}derivative\ g'(x))\ (at\ x\ within\ s)$ **proof**−
have $*$:$\forall n.\ \exists f\ f'.\ \forall x \in s.\ (f\ has\text{-}derivative\ (f'\ x))\ (at\ x\ within\ s) \wedge (\forall h.\ norm(f'\ x\ h - g'\ x\ h) \leq inverse\ (real\ (Suc\ n)) * norm(h))$
apply(*rule*) **using** *assms*(*2*) **apply**(*erule-tac x*=*inverse* (*real* (*Suc n*)) **in** *allE*) **by** *auto*
guess *f* **using** $*$[*THEN choice*] **..** **note** $* = this$ **guess** *f'* **using** $*$[*THEN choice*] **..** **note** $f = this$
show *?thesis* **apply**(*rule has-antiderivative-sequence*[*OF assms*(*1*), *of f f'*]) **defer** **proof**(*rule*,*rule*)
fix *e*::*real* **assume** $0 < e$ **guess** *N* **using** *reals-Archimedean*[*OF* ‹$e > 0$›] **..** **note** $N = this$
show $\exists N.\ \forall n \geq N.\ \forall x \in s.\ \forall h.\ norm\ (f'\ n\ x\ h - g'\ x\ h) \leq e * norm\ h$ **apply**(*rule-tac x*=*N* **in** *exI*) **proof**(*default*+) **case** *goal1*
have $*$:*inverse* (*real* (*Suc n*)) $\leq e$ **apply**(*rule order-trans*[*OF* - *N*[*THEN less-imp-le*]])
using *goal1*(*1*) **by**(*auto simp add*:*field-simps*)

show *?case* **using** *f*[*rule-format*,*THEN conjunct2*,*OF goal1*(*2*), *of n*, *THEN spec*[**where** *x*=*h*]]
apply(*rule order-trans*) **using** *N* ∗ **apply**(*cases h*=*0*) **by** *auto* **qed** **qed**(*insert f*,*auto*) **qed**

22.32 Differentiation of a series.

definition *sums-seq* :: (*nat* ⇒ *'a*::*real-normed-vector*) ⇒ *'a* ⇒ (*nat set*) ⇒ *bool* (**infixl** *sums'-seq 12*) **where** (*f sums-seq l*) *s* ≡ ((λ*n*. *setsum f* (*s* ∩ {*0*..*n*})) −−−> *l*) *sequentially*

lemma *has-derivative-series*: **fixes** *f*::*nat* ⇒ *real^'m* ⇒ *real^'n*
assumes *convex s* ∀ *n*. ∀ *x*∈*s*. ((*f n*) *has-derivative* (*f' n x*)) (*at x within s*)
∀ *e*>*0*. ∃ *N*. ∀ *n*≥*N*. ∀ *x*∈*s*. ∀ *h*. *norm*(*setsum* (λ*i*. *f' i x h*) (*k* ∩ {*0*..*n*}) − *g' x h*) ≤ *e* ∗ *norm*(*h*)
x∈*s* ((λ*n*. *f n x*) *sums-seq l*) *k*
shows ∃ *g*. ∀ *x*∈*s*. ((λ*n*. *f n x*) *sums-seq* (*g x*)) *k* ∧ (*g has-derivative g'*(*x*)) (*at x within s*)
unfolding *sums-seq-def* **apply**(*rule has-derivative-sequence*[*OF assms*(*1*) - *assms*(*3*)]) **apply**(*rule*,*rule*)
apply(*rule has-derivative-setsum*) **defer apply**(*rule*,*rule assms*(*2*)[*rule-format*],*assumption*)
using *assms*(*4*−*5*) **unfolding** *sums-seq-def* **by** *auto*

22.33 Derivative with composed bilinear function.

lemma *has-derivative-bilinear-within*: **fixes** *h*::*real^'m* ⇒ *real^'n* ⇒ *real^'p* **and** *f*::*real^'q* ⇒ *real^'m*
assumes (*f has-derivative f'*) (*at x within s*) (*g has-derivative g'*) (*at x within s*) *bounded-bilinear h*
shows ((λ*x*. *h* (*f x*) (*g x*)) *has-derivative* (λ*d*. *h* (*f x*) (*g' d*) + *h* (*f' d*) (*g x*))) (*at x within s*) **proof**−
have (*g* −−−> *g x*) (*at x within s*) **apply**(*rule differentiable-imp-continuous-within*[*unfolded continuous-within*])
using *assms*(*2*) **unfolding** *differentiable-def* **by** *auto* **moreover**
interpret *f'*:*bounded-linear f'* **using** *assms* **unfolding** *has-derivative-def* **by** *auto*
interpret *g'*:*bounded-linear g'* **using** *assms* **unfolding** *has-derivative-def* **by** *auto*
interpret *h*:*bounded-bilinear h* **using** *assms* **by** *auto*
have ((λ*y*. *f'* (*y* − *x*)) −−−> *0*) (*at x within s*) **unfolding** *f'.zero*[*THEN sym*]
apply(*rule Lim-linear*[*of* λ*y*. *y* − *x 0 at x within s f'*]) **using** *Lim-sub*[*OF Lim-within-id Lim-const*, *of x x s*]
unfolding *id-def* **using** *assms*(*1*) **unfolding** *has-derivative-def* **by** *auto*
hence ((λ*y*. *f x* + *f'* (*y* − *x*)) −−−> *f x*) (*at x within s*)
using *Lim-add*[*OF Lim-const*, *of* λ*y*. *f'* (*y* − *x*) *0 at x within s f x*] **by** *auto*
ultimately
have ∗:((λ*x'*. *h* (*f x* + *f'* (*x'* − *x*)) ((*1*/(*norm* (*x'* − *x*))) ∗$_R$ (*g x'* − (*g x* + *g'* (*x'* − *x*))))
+ *h* ((*1*/ (*norm* (*x'* − *x*))) ∗$_R$ (*f x'* − (*f x* + *f'* (*x'* − *x*)))) (*g x'*)) −−−> *h* (*f x*) *0* + *h 0* (*g x*)) (*at x within s*)

apply−**apply**(*rule Lim-add*) **apply**(*rule-tac*[!] *Lim-bilinear*[*OF - - assms*(*3*)])
using *assms*(*1*−*2*) **unfolding** *has-derivative-within* **by** *auto*
guess *B* **using** *bounded-bilinear.pos-bounded*[*OF assms*(*3*)] **..** **note** *B*=*this*
guess *C* **using** *f'.pos-bounded* **..** **note** *C*=*this*
guess *D* **using** *g'.pos-bounded* **..** **note** *D*=*this*
have *bcd*:$B * C * D > 0$ **using** *B C D* **by** (*auto intro*!: *mult-pos-pos*)
have **:$((\lambda y.\ (1/(norm(y - x))) *_R (h\ (f'(y - x))\ (g'(y - x)))) ---> 0)\ (at\ x\ within\ s)$ **unfolding** *Lim-within* **proof**(*rule*,*rule*) **case** *goal1*
hence $e/(B*C*D)>0$ **using** *B C D* **by**(*auto intro*!:*divide-pos-pos mult-pos-pos*)
thus *?case* **apply**(*rule-tac* $x=e/(B*C*D)$ **in** *exI*,*rule*) **proof**(*rule*,*rule*,*erule conjE*)
fix *y* **assume** *as*:$y \in s\ 0 < dist\ y\ x\ dist\ y\ x < e\ /\ (B * C * D)$
have $norm\ (h\ (f'\ (y - x))\ (g'\ (y - x))) \leq norm\ (f'\ (y - x)) * norm\ (g'\ (y - x)) * B$ **using** *B* **by** *auto*
also have $\ldots \leq (norm\ (y - x) * C) * (D * norm\ (y - x)) * B$ **apply**(*rule mult-right-mono*)
apply(*rule mult-mono*) **using** *B C D* **by** (*auto simp add*: *field-simps intro*!:*mult-nonneg-nonneg*)
also have $\ldots = (B * C * D * norm\ (y - x)) * norm\ (y - x)$ **by**(*auto simp add*:*field-simps*)
also have $\ldots < e * norm\ (y - x)$ **apply**(*rule mult-strict-right-mono*)
using *as*(*3*)[*unfolded dist-norm*] **and** *as*(*2*) **unfolding** *pos-less-divide-eq*[*OF bcd*] **by** (*auto simp add*:*field-simps*)
finally show $dist\ ((1\ /\ norm\ (y - x)) *_R h\ (f'\ (y - x))\ (g'\ (y - x)))\ 0 < e$
unfolding *dist-norm* **apply**−**apply**(*cases* $y = x$) **by**(*auto simp add*:*field-simps*)
qed **qed**
have *bounded-linear* $(\lambda d.\ h\ (f\ x)\ (g'\ d) + h\ (f'\ d)\ (g\ x))$ **unfolding** *linear-linear linear-def*
unfolding *smult-conv-scaleR* **unfolding** *g'.add f'.scaleR f'.add g'.scaleR*
unfolding *h.add-right h.add-left scaleR-right-distrib h.scaleR-left h.scaleR-right* **by** *auto*
thus *?thesis* **using** *Lim-add*[*OF* **] **unfolding** *has-derivative-within*
unfolding *smult-conv-scaleR* **unfolding** *g'.add f'.scaleR f'.add g'.scaleR f'.diff g'.diff*
h.add-right h.add-left scaleR-right-distrib h.scaleR-left h.scaleR-right h.diff-right h.diff-left
scaleR-right-diff-distrib h.zero-right h.zero-left **by**(*auto simp add*:*field-simps*)
qed

lemma *has-derivative-bilinear-at*: **fixes** *h*::*real^'m* ⇒ *real^'n* ⇒ *real^'p* **and** *f*::*real^'p* ⇒ *real^'m*
assumes (*f has-derivative f'*) (*at x*) (*g has-derivative g'*) (*at x*) *bounded-bilinear h*
shows $((\lambda x.\ h\ (f\ x)\ (g\ x))\ has\text{-}derivative\ (\lambda d.\ h\ (f\ x)\ (g'\ d) + h\ (f'\ d)\ (g\ x)))\ (at\ x)$
using *has-derivative-bilinear-within*[*of f f' x UNIV g g' h*] **unfolding** *within-UNIV*
using *assms* **by** *auto*

22.34 Considering derivative $(real,\ 1)\ cart \Rightarrow (real,\ 'n)\ cart$ as a vector.

definition *has-vector-derivative* :: $(real \Rightarrow 'b{::}real\text{-}normed\text{-}vector) \Rightarrow ('b) \Rightarrow (real\ net \Rightarrow bool)$
(**infixl** *has'-vector'-derivative 12*) **where**
$(f\ has\text{-}vector\text{-}derivative\ f')\ net \equiv (f\ has\text{-}derivative\ (\lambda x.\ x *_R f'))\ net$

definition *vector-derivative f net* $\equiv (SOME\ f'.\ (f\ has\text{-}vector\text{-}derivative\ f')\ net)$

lemma *vector-derivative-works*: **fixes** $f{::}real \Rightarrow 'a{::}real\text{-}normed\text{-}vector$
shows *f differentiable net* $\longleftrightarrow$ $(f\ has\text{-}vector\text{-}derivative\ (vector\text{-}derivative\ f\ net))$ *net* (**is** *?l* = *?r*)
proof assume *?l* **guess** f' **using** ‹*?l*›[*unfolded differentiable-def*] **.. note** $f' = this$
then interpret *bounded-linear* f' **by** *auto*
thus *?r* **unfolding** *vector-derivative-def has-vector-derivative-def*
apply−**apply**(*rule someI-ex*,*rule-tac x*=f' *1* **in** *exI*)
using f' **unfolding** *scaleR*[*THEN sym*] **by** *auto*
next assume *?r* **thus** *?l* **unfolding** *vector-derivative-def has-vector-derivative-def differentiable-def* **by** *auto* **qed**

lemma *vector-derivative-unique-at*: **fixes** $f{::}real \Rightarrow real\hat{}'n$
assumes $(f\ has\text{-}vector\text{-}derivative\ f')\ (at\ x)$ $(f\ has\text{-}vector\text{-}derivative\ f'')\ (at\ x)$
shows $f' = f''$ **proof**−
have $*{:}(\lambda x.\ x *_R f') \circ dest\text{-}vec1 = (\lambda x.\ x *_R f'') \circ dest\text{-}vec1$ **apply**(*rule frechet-derivative-unique-at*)
using *assms*[*unfolded has-vector-derivative-def*] **unfolding** *has-derivative-at-dest-vec1*[*THEN sym*] **by** *auto*
show *?thesis* **proof**(*rule ccontr*) **assume** $f' \neq f''$ **moreover**
hence $((\lambda x.\ x *_R f') \circ dest\text{-}vec1)\ (vec1\ 1) = ((\lambda x.\ x *_R f'') \circ dest\text{-}vec1)\ (vec1\ 1)$ **using** $*$ **by** *auto*
ultimately show *False* **unfolding** *o-def vec1-dest-vec1* **by** *auto* **qed qed**

lemma *vector-derivative-unique-within-closed-interval*: **fixes** $f{::}real \Rightarrow real\hat{}'n$
assumes $a < b\ x \in \{a..b\}$
$(f\ has\text{-}vector\text{-}derivative\ f')\ (at\ x\ within\ \{a..b\})$
$(f\ has\text{-}vector\text{-}derivative\ f'')\ (at\ x\ within\ \{a..b\})$ **shows** $f' = f''$ **proof**−
have $*{:}(\lambda x.\ x *_R f') \circ dest\text{-}vec1 = (\lambda x.\ x *_R f'') \circ dest\text{-}vec1$
apply(*rule frechet-derivative-unique-within-closed-interval*[*of vec1 a vec1 b*])
using *assms*(*3*−)[*unfolded has-vector-derivative-def*]
unfolding *has-derivative-within-dest-vec1*[*THEN sym*] *vec1-interval* **using** *assms*(*1*−*2*) **by** *auto*
show *?thesis* **proof**(*rule ccontr*) **assume** $f' \neq f''$ **moreover**
hence $((\lambda x.\ x *_R f') \circ dest\text{-}vec1)\ (vec1\ 1) = ((\lambda x.\ x *_R f'') \circ dest\text{-}vec1)\ (vec1\ 1)$ **using** $*$ **by** *auto*
ultimately show *False* **unfolding** *o-def vec1-dest-vec1* **by** *auto* **qed qed**

lemma *vector-derivative-at*: **fixes** $f{::}real \Rightarrow real\hat{}'a$ **shows**
$(f\ has\text{-}vector\text{-}derivative\ f')\ (at\ x) \Longrightarrow vector\text{-}derivative\ f\ (at\ x) = f'$
apply(*rule vector-derivative-unique-at*) **defer apply** *assumption*

unfolding *vector-derivative-works*[*THEN sym*] *differentiable-def*
unfolding *has-vector-derivative-def* **by** *auto*

lemma *vector-derivative-within-closed-interval*: **fixes** f::*real* ⇒ *real*^'*a*
assumes $a < b$ $x \in \{a..b\}$ (*f has-vector-derivative f'*) (*at x within* $\{a..b\}$)
shows *vector-derivative f* (*at x within* $\{a..b\}$) = f'
apply(*rule vector-derivative-unique-within-closed-interval*)
using *vector-derivative-works*[*unfolded differentiable-def*]
using *assms* **by**(*auto simp add*:*has-vector-derivative-def*)

lemma *has-vector-derivative-within-subset*:
(*f has-vector-derivative f'*) (*at x within s*) ⟹ $t \subseteq s$ ⟹ (*f has-vector-derivative f'*) (*at x within t*)
unfolding *has-vector-derivative-def* **apply**(*rule has-derivative-within-subset*) **by** *auto*

lemma *has-vector-derivative-const*:
((λ*x*. *c*) *has-vector-derivative 0*) *net*
unfolding *has-vector-derivative-def* **using** *has-derivative-const* **by** *auto*

lemma *has-vector-derivative-id*: ((λ*x*::*real*. *x*) *has-vector-derivative 1*) *net*
unfolding *has-vector-derivative-def* **using** *has-derivative-id* **by** *auto*

lemma *has-vector-derivative-cmul*: (*f has-vector-derivative f'*) *net* ⟹ ((λ*x*. *c* *$_R$ *f x*) *has-vector-derivative* (*c* *$_R$ *f'*)) *net*
unfolding *has-vector-derivative-def* **apply**(*drule has-derivative-cmul*) **by**(*auto simp add*:*algebra-simps*)

lemma *has-vector-derivative-cmul-eq*: **assumes** $c \neq 0$
shows (((λ*x*. *c* *$_R$ *f x*) *has-vector-derivative* (*c* *$_R$ *f'*)) *net* ⟷ (*f has-vector-derivative f'*) *net*)
apply *rule* **apply**(*drule has-vector-derivative-cmul*[**where** *c*=*1*/*c*]) **defer**
apply(*rule has-vector-derivative-cmul*) **using** *assms* **by** *auto*

lemma *has-vector-derivative-neg*:
(*f has-vector-derivative f'*) *net* ⟹ ((λ*x*. −(*f x*)) *has-vector-derivative* (− *f'*)) *net*
unfolding *has-vector-derivative-def* **apply**(*drule has-derivative-neg*) **by** *auto*

lemma *has-vector-derivative-add*:
assumes (*f has-vector-derivative f'*) *net* (*g has-vector-derivative g'*) *net*
shows ((λ*x*. *f*(*x*) + *g*(*x*)) *has-vector-derivative* (*f'* + *g'*)) *net*
using *has-derivative-add*[*OF assms*[*unfolded has-vector-derivative-def*]]
unfolding *has-vector-derivative-def* **unfolding** *scaleR-right-distrib* **by** *auto*

lemma *has-vector-derivative-sub*:
assumes (*f has-vector-derivative f'*) *net* (*g has-vector-derivative g'*) *net*
shows ((λ*x*. *f*(*x*) − *g*(*x*)) *has-vector-derivative* (*f'* − *g'*)) *net*
using *has-derivative-sub*[*OF assms*[*unfolded has-vector-derivative-def*]]
unfolding *has-vector-derivative-def scaleR-right-diff-distrib* **by** *auto*

lemma *has-vector-derivative-bilinear-within*: **fixes** *h*::*real^'m* ⇒ *real^'n* ⇒ *real^'p*
assumes (*f has-vector-derivative f'*) (*at x within s*) (*g has-vector-derivative g'*) (*at x within s*) *bounded-bilinear h*
shows ((λ*x*. *h* (*f x*) (*g x*)) *has-vector-derivative* (*h* (*f x*) *g'* + *h f'* (*g x*))) (*at x within s*) **proof**−
interpret *bounded-bilinear h* **using** *assms* **by** *auto*
show *?thesis* **using** *has-derivative-bilinear-within*[*OF assms*(*1*−*2*)[*unfolded has-vector-derivative-def has-derivative-within-dest-vec1*[*THEN sym*]], **where** *h*=*h*]
unfolding *o-def vec1-dest-vec1 has-vector-derivative-def*
unfolding *has-derivative-within-dest-vec1*[*unfolded o-def*, **where** *f*=λ*x*. *h* (*f x*) (*g x*) **and** *f'*=λ*d*. *h* (*f x*) (*d* *$_R$ *g'*) + *h* (*d* *$_R$ *f'*) (*g x*)]
using *assms*(*3*) **unfolding** *scaleR-right scaleR-left scaleR-right-distrib* **by** *auto*
qed

lemma *has-vector-derivative-bilinear-at*: **fixes** *h*::*real^'m* ⇒ *real^'n* ⇒ *real^'p*
assumes (*f has-vector-derivative f'*) (*at x*) (*g has-vector-derivative g'*) (*at x*) *bounded-bilinear h*
shows ((λ*x*. *h* (*f x*) (*g x*)) *has-vector-derivative* (*h* (*f x*) *g'* + *h f'* (*g x*))) (*at x*)
apply(*rule has-vector-derivative-bilinear-within*[**where** *s*=*UNIV*, *unfolded within-UNIV*])
using *assms* **by** *auto*

lemma *has-vector-derivative-at-within*: (*f has-vector-derivative f'*) (*at x*) ⟹ (*f has-vector-derivative f'*) (*at x within s*)
unfolding *has-vector-derivative-def* **apply**(*rule has-derivative-at-within*) **by** *auto*

lemma *has-vector-derivative-transform-within*:
assumes *0* < *d x* ∈ *s* ∀ *x'*∈*s*. *dist x' x* < *d* ⟶ *f x'* = *g x'* (*f has-vector-derivative f'*) (*at x within s*)
shows (*g has-vector-derivative f'*) (*at x within s*)
using *assms* **unfolding** *has-vector-derivative-def* **by**(*rule has-derivative-transform-within*)

lemma *has-vector-derivative-transform-at*:
assumes *0* < *d* ∀ *x'*. *dist x' x* < *d* ⟶ *f x'* = *g x'* (*f has-vector-derivative f'*) (*at x*)
shows (*g has-vector-derivative f'*) (*at x*)
using *assms* **unfolding** *has-vector-derivative-def* **by**(*rule has-derivative-transform-at*)

lemma *has-vector-derivative-transform-within-open*:
assumes *open s x* ∈ *s* ∀ *y*∈*s*. *f y* = *g y* (*f has-vector-derivative f'*) (*at x*)
shows (*g has-vector-derivative f'*) (*at x*)
using *assms* **unfolding** *has-vector-derivative-def* **by**(*rule has-derivative-transform-within-open*)

lemma *vector-diff-chain-at*:
assumes (*f has-vector-derivative f'*) (*at x*) (*g has-vector-derivative g'*) (*at* (*f x*))
shows ((*g* ∘ *f*) *has-vector-derivative* (*f'* *$_R$ *g'*)) (*at x*)
using *assms*(*2*) **unfolding** *has-vector-derivative-def* **apply**− **apply**(*drule diff-chain-at*[*OF assms*(*1*)[*unfolded has-vector-derivative-def*]])
unfolding *o-def scaleR.scaleR-left* **by** *auto*

lemma *vector-diff-chain-within*:
assumes (*f has-vector-derivative f'*) (*at x within s*) (*g has-vector-derivative g'*) (*at* (*f x*) *within f ' s*)
shows ((*g o f*) *has-vector-derivative* (*f' *$_R$ *g'*)) (*at x within s*)
using *assms*(*2*) **unfolding** *has-vector-derivative-def* **apply**− **apply**(*drule diff-chain-within*[*OF assms*(*1*)[*unfolded has-vector-derivative-def*]])
unfolding *o-def scaleR.scaleR-left* **by** *auto*

end

23 Integration: Kurzweil-Henstock gauge integration in many dimensions.

theory *Integration*
imports *Derivative* ~~/*src*/*HOL*/*Decision-Procs*/*Dense-Linear-Order*
begin

declare [[*smt-certificates*=~~/*src*/*HOL*/*Multivariate-Analysis*/*Integration.certs*]]
declare [[*smt-fixed*=*true*]]
declare [[*z3-proofs*=*true*]]

setup ⟪ *Arith-Data.add-tactic Ferrante*−*Rackoff* (*K FerranteRackoff.dlo-tac*) ⟫

23.1 Sundries

lemma *conjunctD2*: **assumes** *a* ∧ *b* **shows** *a b* **using** *assms* **by** *auto*
lemma *conjunctD3*: **assumes** *a* ∧ *b* ∧ *c* **shows** *a b c* **using** *assms* **by** *auto*
lemma *conjunctD4*: **assumes** *a* ∧ *b* ∧ *c* ∧ *d* **shows** *a b c d* **using** *assms* **by** *auto*
lemma *conjunctD5*: **assumes** *a* ∧ *b* ∧ *c* ∧ *d* ∧ *e* **shows** *a b c d e* **using** *assms* **by** *auto*

declare *smult-conv-scaleR*[*simp*]

lemma *simple-image*: {*f x* |*x* . *x* ∈ *s*} = *f ' s* **by** *blast*

lemma *linear-simps*: **assumes** *bounded-linear f*
shows *f* (*a* + *b*) = *f a* + *f b f* (*a* − *b*) = *f a* − *f b f 0* = *0 f* (− *a*) = − *f a f* (*s* *$_R$ *v*) = *s* *$_R$ (*f v*)
apply(*rule-tac*[!] *additive.add additive.minus additive.diff additive.zero bounded-linear.scaleR*)
using *assms* **unfolding** *bounded-linear-def additive-def* **by** *auto*

lemma *bounded-linearI*:**assumes** ⋀*x y*. *f* (*x* + *y*) = *f x* + *f y*
⋀*r x*. *f* (*r* *$_R$ *x*) = *r* *$_R$ *f x* ⋀*x*. *norm* (*f x*) ≤ *norm x* ∗ *K*
shows *bounded-linear f*
unfolding *bounded-linear-def additive-def bounded-linear-axioms-def* **using** *assms*
by *auto*

lemma *real-le-inf-subset*:
assumes $t \neq \{\}$ $t \subseteq s$ $\exists b.\ b <=* s$ **shows** *Inf s <= Inf* (*t*::*real set*)
apply(*rule isGlb-le-isLb*) **apply**(*rule Inf*[*OF assms*(*1*)])
using *assms* **apply**−**apply**(*erule exE*) **apply**(*rule-tac x*=*b* **in** *exI*)
unfolding *isLb-def setge-def* **by** *auto*

lemma *real-ge-sup-subset*:
assumes $t \neq \{\}$ $t \subseteq s$ $\exists b.\ s *<= b$ **shows** *Sup s >= Sup* (*t*::*real set*)
apply(*rule isLub-le-isUb*) **apply**(*rule Sup*[*OF assms*(*1*)])
using *assms* **apply**−**apply**(*erule exE*) **apply**(*rule-tac x*=*b* **in** *exI*)
unfolding *isUb-def setle-def* **by** *auto*

lemma *dist-trans*[*simp*]:*dist* (*vec1 x*) (*vec1 y*) = *dist x* (*y*::*real*)
unfolding *dist-real-def dist-vec1* **..**

lemma *Lim-trans*[*simp*]: **fixes** *f*::′*a* ⇒ *real*
shows ((λ*x*. *vec1* (*f x*)) −−−> *vec1 l*) *net* ⟷ (*f* −−−> *l*) *net*
using *assms* **unfolding** *Lim dist-trans* **..**

lemma *bounded-linear-component*[*intro*]: *bounded-linear* (λ*x*::*real*^′*n*. *x* $ *k*)
apply(*rule bounded-linearI*[**where** *K*=*1*])
using *component-le-norm*[*of - k*] **unfolding** *real-norm-def* **by** *auto*

lemma *bounded-vec1*[*intro*]: *bounded s* ⟹ *bounded* (*vec1* ‘ (*s*::*real set*))
unfolding *bounded-def* **apply** *safe* **apply**(*rule-tac x*=*vec1 x* **in** *exI*,*rule-tac x*=*e* **in** *exI*) **by** *auto*

lemma *transitive-stepwise-lt-eq*:
assumes (⋀*x y z*::*nat*. *R x y* ⟹ *R y z* ⟹ *R x z*)
shows ((∀ *m*. ∀ *n*>*m*. *R m n*) ⟷ (∀ *n*. *R n* (*Suc n*))) (**is** *?l* = *?r*)
proof(*safe*) **assume** *?r* **fix** *n m*::*nat* **assume** *m* < *n* **thus** *R m n* **apply**−
proof(*induct n arbitrary*: *m*) **case** (*Suc n*) **show** *?case*
proof(*cases m* < *n*) **case** *True*
show *?thesis* **apply**(*rule assms*[*OF Suc*(*1*)[*OF True*]]) **using** ‹*?r*› **by** *auto*
next case *False* **hence** *m* = *n* **using** *Suc*(*2*) **by** *auto*
thus *?thesis* **using** ‹*?r*› **by** *auto*
qed qed *auto* **qed** *auto*

lemma *transitive-stepwise-gt*:
assumes ⋀*x y z*. *R x y* ⟹ *R y z* ⟹ *R x z* ⋀*n*. *R n* (*Suc n*)
shows ∀ *n*>*m*. *R m n*
proof− **have** ∀ *m*. ∀ *n*>*m*. *R m n* **apply**(*subst transitive-stepwise-lt-eq*)
apply(*rule assms*) **apply**(*assumption*,*assumption*) **using** *assms*(*2*) **by** *auto*
thus *?thesis* **by** *auto* **qed**

lemma *transitive-stepwise-le-eq*:
assumes ⋀*x*. *R x x* ⋀*x y z*. *R x y* ⟹ *R y z* ⟹ *R x z*
shows (∀ *m*. ∀ *n*≥*m*. *R m n*) ⟷ (∀ *n*. *R n* (*Suc n*)) (**is** *?l* = *?r*)

proof *safe* **assume** *?r* **fix** *m n*::*nat* **assume** *m*≤*n* **thus** *R m n* **apply**−
proof(*induct n arbitrary*: *m*) **case** (*Suc n*) **show** *?case*
proof(*cases m* ≤ *n*) **case** *True* **show** *?thesis* **apply**(*rule assms*(*2*))
apply(*rule Suc*(*1*)[*OF True*]) **using** ‹*?r*› **by** *auto*
next case *False* **hence** *m* = *Suc n* **using** *Suc*(*2*) **by** *auto*
thus *?thesis* **using** *assms*(*1*) **by** *auto*
qed qed(*insert assms*(*1*), *auto*) **qed** *auto*

lemma *transitive-stepwise-le*:
assumes ⋀*x*. *R x x* ⋀*x y z*. *R x y* ⟹ *R y z* ⟹ *R x z* ⋀*n*. *R n* (*Suc n*)
shows ∀ *n*≥*m*. *R m n*
proof− **have** ∀ *m*. ∀ *n*≥*m*. *R m n* **apply**(*subst transitive-stepwise-le-eq*)
apply(*rule assms*) **apply**(*rule assms*,*assumption*,*assumption*) **using** *assms*(*3*)
by *auto*
thus *?thesis* **by** *auto* **qed**

23.2 Some useful lemmas about intervals.

lemma *empty-as-interval*: {} = {*1*..*0*::*real*^'*n*}
apply(*rule set-ext*,*rule*) **defer unfolding** *vector-le-def mem-interval*
using *UNIV-witness*[**where** *'a*='*n*] **apply**(*erule-tac exE*,*rule-tac x*=*x* **in** *allE*)
by *auto*

lemma *interior-subset-union-intervals*:
assumes *i* = {*a*..*b*::*real*^'*n*} *j* = {*c*..*d*} *interior j* ≠ {} *i* ⊆ *j* ∪ *s interior*(*i*) ∩ *interior*(*j*) = {}
shows *interior i* ⊆ *interior s* **proof**−
have {*a*<..<*b*} ∩ {*c*..*d*} = {} **using** *inter-interval-mixed-eq-empty*[*of c d a b*]
and *assms*(*3*,*5*)
unfolding *assms*(*1*,*2*) *interior-closed-interval* **by** *auto*
moreover have {*a*<..<*b*} ⊆ {*c*..*d*} ∪ *s* **apply**(*rule order-trans*,*rule interval-open-subset-closed*)
using *assms*(*4*) **unfolding** *assms*(*1*,*2*) **by** *auto*
ultimately show *?thesis* **apply**−**apply**(*rule interior-maximal*) **defer apply**(*rule open-interior*)
unfolding *assms*(*1*,*2*) *interior-closed-interval* **by** *auto* **qed**

lemma *inter-interior-unions-intervals*: **fixes** *f*::(*real*^'*n*) *set set*
assumes *finite f open s* ∀ *t*∈*f*. ∃ *a b*. *t* = {*a*..*b*} ∀ *t*∈*f*. *s* ∩ (*interior t*) = {}
shows *s* ∩ *interior*(⋃*f*) = {} **proof**(*rule ccontr*,*unfold ex-in-conv*[*THEN sym*])
case *goal1*
have *lem1*:⋀*x e s U*. *ball x e* ⊆ *s* ∩ *interior U* ⟷ *ball x e* ⊆ *s* ∩ *U* **apply** *rule* **defer apply**(*rule-tac Int-greatest*)
unfolding *open-subset-interior*[*OF open-ball*] **using** *interior-subset* **by** *auto*
have *lem2*:⋀*x s P*. ∃ *x*∈*s*. *P x* ⟹ ∃ *x*∈*insert x s*. *P x* **by** *auto*
have ⋀*f*. *finite f* ⟹ (∀ *t*∈*f*. ∃ *a b*. *t* = {*a*..*b*}) ⟹ (∃ *x*. *x* ∈ *s* ∩ *interior* (⋃*f*)) ⟹ (∃ *t*∈*f*. ∃ *x*. ∃ *e*>*0*. *ball x e* ⊆ *s* ∩ *t*) **proof**− **case** *goal1*
thus *?case* **proof**(*induct rule*:*finite-induct*)
case *empty* **from** *this*(*2*) **guess** *x* **..** **hence** *False* **unfolding** *Union-empty interior-empty* **by** *auto* **thus** *?case* **by** *auto* **next**

case (*insert i f*) **guess** *x* **using** *insert*(*5*) **..** **note** *x* = *this*
then guess *e* **unfolding** *open-contains-ball-eq*[*OF open-Int*[*OF assms*(*2*) *open-interior*],*rule-format*]
.. **note** *e*=*this*
guess *a* **using** *insert*(*4*)[*rule-format*,*OF insertI1*] **..** **then guess** *b* **..** **note** *ab* = *this*
show *?case* **proof**(*cases x*∈*i*) **case** *False* **hence** *x* ∈ *UNIV* − {*a*..*b*} **unfolding** *ab* **by** *auto*
then guess *d* **unfolding** *open-contains-ball-eq*[*OF open-Diff*[*OF open-UNIV closed-interval*],*rule-format*] **..**
hence *0* < *d ball x* (*min d e*) ⊆ *UNIV* − *i* **using** *e* **unfolding** *ab* **by** *auto*
hence *ball x* (*min d e*) ⊆ *s* ∩ *interior* (⋃*f*) **using** *e* **unfolding** *lem1* **by** *auto* **hence** *x* ∈ *s* ∩ *interior* (⋃*f*) **using** ‹*d*>*0*› *e* **by** *auto*
hence ∃ *t*∈*f*. ∃ *x e*. *0* < *e* ∧ *ball x e* ⊆ *s* ∩ *t* **apply**−**apply**(*rule insert*(*3*)) **using** *insert*(*4*) **by** *auto* **thus** *?thesis* **by** *auto* **next**
case *True* **show** *?thesis* **proof**(*cases x*∈{*a*<..<*b*})
case *True* **then guess** *d* **unfolding** *open-contains-ball-eq*[*OF open-interval*,*rule-format*] **..**
thus *?thesis* **apply**(*rule-tac x*=*i* **in** *bexI*,*rule-tac x*=*x* **in** *exI*,*rule-tac x*=*min d e* **in** *exI*)
unfolding *ab* **using** *interval-open-subset-closed*[*of a b*] **and** *e* **by** *fastsimp*+ **next**
case *False* **then obtain** *k* **where** *x*$*k* ≤ *a*$*k* ∨ *x*$*k* ≥ *b*$*k* **unfolding** *mem-interval* **by**(*auto simp add*:*not-less*)
hence *x*$*k* = *a*$*k* ∨ *x*$*k* = *b*$*k* **using** *True* **unfolding** *ab* **and** *mem-interval* **apply**(*erule-tac x*=*k* **in** *allE*) **by** *auto*
hence ∃ *x*. *ball x* (*e*/*2*) ⊆ *s* ∩ (⋃*f*) **proof**(*erule-tac disjE*)
let *?z* = *x* − (*e*/*2*) *∗R basis k* **assume** *as*:*x*$*k* = *a*$*k* **have** *ball ?z* (*e* / *2*) ∩ *i* = {} **apply**(*rule ccontr*) **unfolding** *ex-in-conv*[*THEN sym*] **proof**(*erule exE*)
fix *y* **assume** *y* ∈ *ball ?z* (*e* / *2*) ∩ *i* **hence** *dist ?z y* < *e*/*2* **and** *yi*:*y*∈*i* **by** *auto*
hence |(*?z* − *y*) $ *k*| < *e*/*2* **using** *component-le-norm*[*of ?z* − *y k*] **unfolding** *dist-norm* **by** *auto*
hence *y*$*k* < *a*$*k* **unfolding** *vector-component-simps vector-scaleR-component as* **using** *e*[*THEN conjunct1*] **by**(*auto simp add*:*field-simps*)
hence *y* ∉ *i* **unfolding** *ab mem-interval not-all* **by**(*rule-tac x*=*k* **in** *exI*,*auto*) **thus** *False* **using** *yi* **by** *auto* **qed**
moreover have *ball ?z* (*e*/*2*) ⊆ *s* ∩ (⋃ *insert i f*) **apply**(*rule order-trans*[*OF - e*[*THEN conjunct2*, *unfolded lem1*]]) **proof**
fix *y* **assume** *as*:*y*∈ *ball ?z* (*e*/*2*) **have** *norm* (*x* − *y* − (*e* / *2*) *∗R basis k*) ≤ |*e*| / *2* + *norm* (*x* − *y*) **apply**−**apply**(*rule order-trans*,*rule norm-triangle-sub*[*of x* − *y* (*e*/*2*) *∗R basis k*])
unfolding *norm-scaleR norm-basis* **by** *auto*
also have ... < |*e*| / *2* + |*e*| / *2* **apply**(*rule add-strict-left-mono*) **using** *as* **unfolding** *mem-ball dist-norm* **using** *e* **by**(*auto simp add*:*field-simps*)
finally show *y*∈*ball x e* **unfolding** *mem-ball dist-norm* **using** *e* **by**(*auto simp add*:*field-simps*) **qed**
ultimately show *?thesis* **apply**(*rule-tac x*=*?z* **in** *exI*) **unfolding** *Union-insert* **by** *auto*

next let *?z = x + (e/2) $*_R$ basis k* **assume** *as:x$k = b$k* **have** *ball ?z (e / 2) ∩ i = {}* **apply**(*rule ccontr*) **unfolding** *ex-in-conv*[*THEN sym*] **proof**(*erule exE*)
fix *y* **assume** *y ∈ ball ?z (e / 2) ∩ i* **hence** *dist ?z y < e/2* **and** *yi:y∈i* **by** *auto*
hence *|(?z − y) $ k| < e/2* **using** *component-le-norm*[*of ?z − y k*] **unfolding** *dist-norm* **by** *auto*
hence *y$k > b$k* **unfolding** *vector-component-simps vector-scaleR-component as* **using** *e*[*THEN conjunct1*] **by**(*auto simp add:field-simps*)
hence *y ∉ i* **unfolding** *ab mem-interval not-all* **by**(*rule-tac x=k* **in** *exI*,*auto*)
thus *False* **using** *yi* **by** *auto* **qed**
moreover have *ball ?z (e/2) ⊆ s ∩ (⋃ insert i f)* **apply**(*rule order-trans*[*OF - e*[*THEN conjunct2*, *unfolded lem1*]]) **proof**
fix *y* **assume** *as:y∈ ball ?z (e/2)* **have** *norm (x − y) ≤ |e| / 2 + norm (x − y + (e / 2) $*_R$ basis k)*
apply−apply(*rule order-trans*,*rule norm-triangle-sub*[*of x − y − (e/2) $*_R$ basis k*])
unfolding *norm-scaleR norm-basis* **by** *auto*
also have *... < |e| / 2 + |e| / 2* **apply**(*rule add-strict-left-mono*) **using** *as* **unfolding** *mem-ball dist-norm* **using** *e* **by**(*auto simp add:field-simps*)
finally show *y∈ball x e* **unfolding** *mem-ball dist-norm* **using** *e* **by**(*auto simp add:field-simps*) **qed**
ultimately show *?thesis* **apply**(*rule-tac x=?z* **in** *exI*) **unfolding** *Union-insert* **by** *auto* **qed**
then guess *x* **.. hence** *x ∈ s ∩ interior (⋃f)* **unfolding** *lem1*[**where** *U=⋃f*,*THEN sym*] **using** *centre-in-ball e*[*THEN conjunct1*] **by** *auto*
thus *?thesis* **apply−apply**(*rule lem2*,*rule insert(3)*) **using** *insert(4)* **by** *auto*
qed qed qed qed note *∗ = this*
guess *t* **using** *∗*[*OF assms(1,3) goal1*] **.. from** *this(2)* **guess** *x* **.. then guess** *e* **..**
hence *x ∈ s x∈interior t* **defer using** *open-subset-interior*[*OF open-ball*, *of x e t*] **by** *auto*
thus *False* **using** ‹*t∈f*› *assms(4)* **by** *auto* **qed**

23.3 Bounds on intervals where they exist.

definition *interval-upperbound* (*s*::(*real^'n*) *set*) = (*χ i. Sup* {*a. ∃ x∈s. x$i = a*})

definition *interval-lowerbound* (*s*::(*real^'n*) *set*) = (*χ i. Inf* {*a. ∃ x∈s. x$i = a*})

lemma *interval-upperbound*[*simp*]: **assumes** *∀ i. a$i ≤ b$i* **shows** *interval-upperbound {a..b} = b*
using *assms* **unfolding** *interval-upperbound-def Cart-eq Cart-lambda-beta* **apply−apply**(*rule*,*erule-tac x=i* **in** *allE*)
apply(*rule Sup-unique*) **unfolding** *setle-def* **apply** *rule* **unfolding** *mem-Collect-eq* **apply**(*erule bexE*) **unfolding** *mem-interval* **defer**
apply(*rule*,*rule*) **apply**(*rule-tac x=b$i* **in** *bexI*) **defer unfolding** *mem-Collect-eq* **apply**(*rule-tac x=b* **in** *bexI*)
unfolding *mem-interval* **using** *assms* **by** *auto*

lemma *interval-lowerbound*[*simp*]: **assumes** $\forall\, i.\ a\$i \leq b\$i$ **shows** *interval-lowerbound* $\{a..b\} = a$
using *assms* **unfolding** *interval-lowerbound-def Cart-eq Cart-lambda-beta* **apply**−**apply**(*rule*,*erule-tac x*=*i* **in** *allE*)
apply(*rule Inf-unique*) **unfolding** *setge-def* **apply** *rule* **unfolding** *mem-Collect-eq* **apply**(*erule bexE*) **unfolding** *mem-interval* **defer**
apply(*rule*,*rule*) **apply**(*rule-tac x*=*a*$*i* **in** *bexI*) **defer unfolding** *mem-Collect-eq* **apply**(*rule-tac x*=*a* **in** *bexI*)
unfolding *mem-interval* **using** *assms* **by** *auto*

lemmas *interval-bounds* = *interval-upperbound interval-lowerbound*

lemma *interval-bounds'*[*simp*]: **assumes** $\{a..b\}\neq\{\}$ **shows** *interval-upperbound* $\{a..b\}$ = *b interval-lowerbound* $\{a..b\} = a$
using *assms* **unfolding** *interval-ne-empty* **by** *auto*

lemma *interval-upperbound-1*[*simp*]: *dest-vec1 a* $\leq$ *dest-vec1 b* $\Longrightarrow$ *interval-upperbound* $\{a..b\}$ = (*b*::*real*^*1*)
apply(*rule interval-upperbound*) **by** *auto*

lemma *interval-lowerbound-1*[*simp*]: *dest-vec1 a* $\leq$ *dest-vec1 b* $\Longrightarrow$ *interval-lowerbound* $\{a..b\}$ = (*a*::*real*^*1*)
apply(*rule interval-lowerbound*) **by** *auto*

lemmas *interval-bound-1* = *interval-upperbound-1 interval-lowerbound-1*

23.4 Content (length, area, volume...) of an interval.

definition *content* (*s*::(*real*^'*n*) *set*) =
(*if s* = {} *then 0 else* ($\prod$ *i*∈*UNIV*. (*interval-upperbound s*)$*i* − (*interval-lowerbound s*)$*i*))

lemma *interval-not-empty*:$\forall\, i.\ a\$i \leq b\$i \Longrightarrow \{a..b::real\hat{}'n\} \neq \{\}$
unfolding *interval-eq-empty* **unfolding** *not-ex not-less* **by** *assumption*

lemma *content-closed-interval*: **assumes** $\forall\, i.\ a\$i \leq b\$i$
shows *content* $\{a..b\}$ = ($\prod$ *i*∈*UNIV*. *b*$*i* − *a*$*i*)
using *interval-not-empty*[*OF assms*] **unfolding** *content-def interval-upperbound*[*OF assms*] *interval-lowerbound*[*OF assms*] **by** *auto*

lemma *content-closed-interval'*: **assumes** $\{a..b\}\neq\{\}$ **shows** *content* $\{a..b\}$ = ($\prod$ *i*∈*UNIV*. *b*$*i* − *a*$*i*)
apply(*rule content-closed-interval*) **using** *assms* **unfolding** *interval-ne-empty* **.**

lemma *content-1*:*dest-vec1 a* $\leq$ *dest-vec1 b* $\Longrightarrow$ *content* $\{a..b\}$ = *dest-vec1 b* − *dest-vec1 a*
using *content-closed-interval*[*of a b*] **by** *auto*

lemma *content-1'*:$a \leq b \Longrightarrow$ *content* $\{$*vec1 a*..*vec1 b*$\} = b - a$ **using** *content-1*[*of vec a vec b*] **by** *auto*

lemma *content-unit*[*intro*]: *content*$\{0..1::real\hat{}'n\} = 1$ **proof**−
have $*:\forall$ *i*. *0*$i \leq (1::real\hat{}'n::finite)\$i$ **by** *auto*
have *0* $\in \{0..1::real\hat{}'n::finite\}$ **unfolding** *mem-interval* **by** *auto*
thus *?thesis* **unfolding** *content-def interval-bounds*[*OF* $*$] **using** *setprod-1* **by** *auto* **qed**

lemma *content-pos-le*[*intro*]: *0* $\leq$ *content* $\{a..b\}$ **proof**(*cases* $\{a..b\}$=$\{\}$)
case *False* **hence** $*:\forall$ *i*. *a* $ *i* $\leq$ *b* $ *i* **unfolding** *interval-ne-empty* **by** *assumption*
have $(\prod i \in UNIV$. *interval-upperbound* $\{a..b\}$ $ *i* − *interval-lowerbound* $\{a..b\}$ $ *i*$) \geq 0$
apply(*rule setprod-nonneg*) **unfolding** *interval-bounds*[*OF* $*$] **using** $*$ **apply**(*erule-tac x*=*x* **in** *allE*) **by** *auto*
thus *?thesis* **unfolding** *content-def* **by**(*auto simp del*:*interval-bounds'*) **qed**(*unfold content-def*, *auto*)

lemma *content-pos-lt*: **assumes** $\forall$ *i*. *a*$*i* < *b*$*i* **shows** *0* < *content* $\{a..b\}$
proof− **have** *help-lemma1*: $\forall$ *i*. *a*$*i* < *b*$*i* $\Longrightarrow \forall$ *i*. *a*$*i* $\leq$ ((*b*$*i*)::*real*) **apply**(*rule*,*erule-tac x*=*i* **in** *allE*) **by** *auto*
show *?thesis* **unfolding** *content-closed-interval*[*OF help-lemma1*[*OF assms*]] **apply**(*rule setprod-pos*)
using *assms* **apply**(*erule-tac x*=*x* **in** *allE*) **by** *auto* **qed**

lemma *content-pos-lt-1*: *dest-vec1 a* < *dest-vec1 b* $\Longrightarrow$ *0* < *content*($\{a..b\}$)
apply(*rule content-pos-lt*) **by** *auto*

lemma *content-eq-0*: *content*($\{a..b::real\hat{}'n\}$) = *0* $\longleftrightarrow$ ($\exists$ *i*. *b*$*i* $\leq$ *a*$*i*) **proof**(*cases* $\{a..b\} = \{\}$)
case *True* **thus** *?thesis* **unfolding** *content-def if-P*[*OF True*] **unfolding** *interval-eq-empty* **apply**−
apply(*rule*,*erule exE*) **apply**(*rule-tac x*=*i* **in** *exI*) **by** *auto* **next**
guess *a* **using** *UNIV-witness*[**where** *'a*=*'n*] **..** **case** *False* **note** *as*=*this*[*unfolded interval-eq-empty not-ex not-less*]
show *?thesis* **unfolding** *content-def if-not-P*[*OF False*] *setprod-zero-iff*[*OF finite-UNIV*]
apply(*rule*) **apply**(*erule-tac*[!] *exE bexE*) **unfolding** *interval-bounds*[*OF as*] **apply**(*rule-tac x*=*x* **in** *exI*) **defer**
apply(*rule-tac x*=*i* **in** *bexI*) **using** *as* **apply**(*erule-tac x*=*i* **in** *allE*) **by** *auto* **qed**

lemma *cond-cases*:($P \Longrightarrow Q\ x$) $\Longrightarrow$ ($\neg\ P \Longrightarrow Q\ y$) $\Longrightarrow Q$ (*if P then x else y*) **by** *auto*

lemma *content-closed-interval-cases*:
content $\{a..b\}$ = (*if* $\forall$ *i*. *a*$*i* $\leq$ *b*$*i* *then setprod* ($\lambda$*i*. *b*$*i* − *a*$*i*) *UNIV else 0*) **apply**(*rule cond-cases*)
apply(*rule content-closed-interval*) **unfolding** *content-eq-0 not-all not-le* **defer**
apply(*erule exE*,*rule-tac x*=*x* **in** *exI*) **by** *auto*

lemma *content-eq-0-interior*: *content* $\{a..b\} = 0 \longleftrightarrow$ *interior*$(\{a..b\}) = \{\}$
unfolding *content-eq-0 interior-closed-interval interval-eq-empty* **by** *auto*

lemma *content-eq-0-1*: *content* $\{a..b::real\hat{}1\} = 0 \longleftrightarrow$ *dest-vec1* $b \leq$ *dest-vec1* a
unfolding *content-eq-0* **by** *auto*

lemma *content-pos-lt-eq*: $0 <$ *content* $\{a..b\} \longleftrightarrow (\forall i.\ a\$i < b\$i)$
apply(*rule*) **defer apply**(*rule content-pos-lt*,*assumption*) **proof**− **assume** $0 <$ *content* $\{a..b\}$
hence *content* $\{a..b\} \neq 0$ **by** *auto* **thus** $\forall i.\ a\$i < b\$i$ **unfolding** *content-eq-0 not-ex not-le* **by** *auto* **qed**

lemma *content-empty*[*simp*]: *content* $\{\} = 0$ **unfolding** *content-def* **by** *auto*

lemma *content-subset*: **assumes** $\{a..b\} \subseteq \{c..d\}$ **shows** *content* $\{a..b::real\hat{}'n\} \leq$ *content* $\{c..d\}$ **proof**(*cases* $\{a..b\}=\{\}$)
case *True* **thus** *?thesis* **using** *content-pos-le*[*of c d*] **by** *auto* **next**
case *False* **hence** *ab-ne*:$\forall i.\ a\ \$\ i \leq b\ \$\ i$ **unfolding** *interval-ne-empty* **by** *auto*
hence *ab-ab*:$a\in\{a..b\}$ $b\in\{a..b\}$ **unfolding** *mem-interval* **by** *auto*
have $\{c..d\} \neq \{\}$ **using** *assms False* **by** *auto*
hence *cd-ne*:$\forall i.\ c\ \$\ i \leq d\ \$\ i$ **using** *assms* **unfolding** *interval-ne-empty* **by** *auto*
show *?thesis* **unfolding** *content-def* **unfolding** *interval-bounds*[*OF ab-ne*] *interval-bounds*[*OF cd-ne*]
unfolding *if-not-P*[*OF False*] *if-not-P*[*OF* ‹$\{c..d\} \neq \{\}$›] **apply**(*rule setprod-mono*,*rule*)
proof fix $i::'n$
show $0 \leq b\ \$\ i - a\ \$\ i$ **using** *ab-ne*[*THEN spec*[**where** $x=i$]] **by** *auto*
show $b\ \$\ i - a\ \$\ i \leq d\ \$\ i - c\ \$\ i$
using *assms*[*unfolded subset-eq mem-interval*,*rule-format*,*OF ab-ab*(*2*),*of i*]
using *assms*[*unfolded subset-eq mem-interval*,*rule-format*,*OF ab-ab*(*1*),*of i*]
by *auto* **qed qed**

lemma *content-lt-nz*: $0 <$ *content* $\{a..b\} \longleftrightarrow$ *content* $\{a..b\} \neq 0$
unfolding *content-pos-lt-eq content-eq-0* **unfolding** *not-ex not-le* **by** *auto*

23.5 The notion of a gauge — simply an open set containing the point.

definition *gauge* **where** *gauge* $d \longleftrightarrow (\forall x.\ x\in(d\ x) \wedge$ *open*$(d\ x))$

lemma *gaugeI*:**assumes** $\bigwedge x.\ x\in g\ x$ $\bigwedge x.$ *open* $(g\ x)$ **shows** *gauge* g
using *assms* **unfolding** *gauge-def* **by** *auto*

lemma *gaugeD*[*dest*]: **assumes** *gauge* d **shows** $x\in d\ x$ *open* $(d\ x)$ **using** *assms* **unfolding** *gauge-def* **by** *auto*

lemma *gauge-ball-dependent*: $\forall x.\ 0 < e\ x \Longrightarrow$ *gauge* $(\lambda x.$ *ball* $x\ (e\ x))$
unfolding *gauge-def* **by** *auto*

lemma *gauge-ball*[*intro*]: $0 < e \Longrightarrow$ *gauge* $(\lambda x.$ *ball* $x\ e)$ **unfolding** *gauge-def* **by** *auto*

lemma *gauge-trivial*[*intro*]: *gauge* $(\lambda x.$ *ball* $x\ 1)$ **apply**(*rule gauge-ball*) **by** *auto*

lemma *gauge-inter*[*intro*]: *gauge d1* $\Longrightarrow$ *gauge d2* $\Longrightarrow$ *gauge* $(\lambda x.\ (d1\ x) \cap (d2\ x))$
unfolding *gauge-def* **by** *auto*

lemma *gauge-inters*: **assumes** *finite* $s\ \forall\, d{\in}s.$ *gauge* $(f\ d)$ **shows** *gauge*$(\lambda x. \bigcap \{f\ d\ x \mid d.\ d \in s\})$ **proof**−
have $*{:}\bigwedge x.\ \{f\ d\ x \mid d.\ d \in s\} = (\lambda d.\ f\ d\ x)\ `\ s$ **by** *auto* **show** *?thesis*
unfolding *gauge-def* **unfolding** $*$
using *assms* **unfolding** *Ball-def Inter-iff mem-Collect-eq gauge-def* **by** *auto* **qed**

lemma *gauge-existence-lemma*: $(\forall\, x.\ \exists\, d{::}real.\ p\ x \longrightarrow 0 < d \wedge q\ d\ x) \longleftrightarrow (\forall\, x.\ \exists\, d{>}0.\ p\ x \longrightarrow q\ d\ x)$ **by**(*meson zero-less-one*)

23.6 Divisions.

definition *division-of* (**infixl** *division'-of 40*) **where**
s division-of i ≡
finite s ∧
$(\forall\, k{\in}s.\ k \subseteq i \wedge k \neq \{\} \wedge (\exists\, a\ b.\ k = \{a..b\})) \wedge$
$(\forall\, k1{\in}s.\ \forall\, k2{\in}s.\ k1 \neq k2 \longrightarrow interior(k1) \cap interior(k2) = \{\}) \wedge$
$(\bigcup s = i)$

lemma *division-ofD*[*dest*]: **assumes** *s division-of i*
shows*finite s* $\bigwedge k.\ k{\in}s \Longrightarrow k \subseteq i$ $\bigwedge k.\ k{\in}s \Longrightarrow k \neq \{\}$ $\bigwedge k.\ k{\in}s \Longrightarrow (\exists\, a\ b.\ k = \{a..b\})$
$\bigwedge k1\ k2.\ [\![k1{\in}s;\ k2{\in}s;\ k1 \neq k2]\!] \Longrightarrow interior(k1) \cap interior(k2) = \{\}$ $\bigcup s = i$
using *assms* **unfolding** *division-of-def* **by** *auto*

lemma *division-ofI*:
assumes *finite* s $\bigwedge k.\ k{\in}s \Longrightarrow k \subseteq i$ $\bigwedge k.\ k{\in}s \Longrightarrow k \neq \{\}$ $\bigwedge k.\ k{\in}s \Longrightarrow (\exists\, a\ b.\ k = \{a..b\})$
$\bigwedge k1\ k2.\ [\![k1{\in}s;\ k2{\in}s;\ k1 \neq k2]\!] \Longrightarrow interior(k1) \cap interior(k2) = \{\}$ $\bigcup s = i$
shows *s division-of i* **using** *assms* **unfolding** *division-of-def* **by** *auto*

lemma *division-of-finite*: *s division-of i* $\Longrightarrow$ *finite s*
unfolding *division-of-def* **by** *auto*

lemma *division-of-self*[*intro*]: $\{a..b\} \neq \{\} \Longrightarrow \{\{a..b\}\}$ *division-of* $\{a..b\}$
unfolding *division-of-def* **by** *auto*

lemma *division-of-trivial*[*simp*]: *s division-of* $\{\} \longleftrightarrow s = \{\}$ **unfolding** *division-of-def* **by** *auto*

lemma *division-of-sing*[*simp*]: *s division-of* {*a*..*a*::*real*^'*n*} ⟷ *s* = {{*a*..*a*}} (**is** *?l* = *?r*) **proof**
assume *?r* **moreover { assume** *s* = {{*a*}} **moreover fix** *k* **assume** *k*∈*s*
ultimately have∃ *x y*. *k* = {*x*..*y*} **apply**(*rule-tac x*=*a* **in** *exI*)+ **unfolding** *interval-sing*[*THEN conjunct1*] **by** *auto* **}**
ultimately show *?l* **unfolding** *division-of-def interval-sing*[*THEN conjunct1*] **by** *auto* **next**
assume *?l* **note** *as*=*conjunctD4*[*OF this*[*unfolded division-of-def interval-sing*[*THEN conjunct1*]]]
{ fix *x* **assume** *x*:*x*∈*s* **have** *x*={*a*} **using** *as*(*2*)[*rule-format*,*OF x*] **by** *auto* **}**
moreover have *s* ≠ {} **using** *as*(*4*) **by** *auto* **ultimately show** *?r* **unfolding** *interval-sing*[*THEN conjunct1*] **by** *auto* **qed**

lemma *elementary-empty*: **obtains** *p* **where** *p division-of* {}
unfolding *division-of-trivial* **by** *auto*

lemma *elementary-interval*: **obtains** *p* **where** *p division-of* {*a*..*b*}
by(*metis division-of-trivial division-of-self*)

lemma *division-contains*: *s division-of i* ⟹ ∀ *x*∈*i*. ∃ *k*∈*s*. *x* ∈ *k*
unfolding *division-of-def* **by** *auto*

lemma *forall-in-division*:
d division-of i ⟹ ((∀ *x*∈*d*. *P x*) ⟷ (∀ *a b*. {*a*..*b*} ∈ *d* ⟶ *P* {*a*..*b*}))
unfolding *division-of-def* **by** *fastsimp*

lemma *division-of-subset*: **assumes** *p division-of* (⋃ *p*) *q* ⊆ *p* **shows** *q division-of* (⋃ *q*)
apply(*rule division-ofI*) **proof**− **note** *as*=*division-ofD*[*OF assms*(*1*)]
show *finite q* **apply**(*rule finite-subset*) **using** *as*(*1*) *assms*(*2*) **by** *auto*
{ fix *k* **assume** *k* ∈ *q* **hence** *kp*:*k*∈*p* **using** *assms*(*2*) **by** *auto* **show** *k*⊆⋃ *q* **using** ‹*k* ∈ *q*› **by** *auto*
show ∃ *a b*. *k* = {*a*..*b*} **using** *as*(*4*)[*OF kp*] **by** *auto* **show** *k* ≠ {} **using** *as*(*3*)[*OF kp*] **by** *auto* **}**
fix *k1 k2* **assume** *k1* ∈ *q k2* ∈ *q k1* ≠ *k2* **hence** ∗:*k1*∈*p k2*∈*p k1*≠*k2* **using** *assms*(*2*) **by** *auto*
show *interior k1* ∩ *interior k2* = {} **using** *as*(*5*)[*OF* ∗] **by** *auto* **qed** *auto*

lemma *division-of-union-self*[*intro*]: *p division-of s* ⟹ *p division-of* (⋃ *p*) **unfolding** *division-of-def* **by** *auto*

lemma *division-of-content-0*: **assumes** *content* {*a*..*b*} = *0 d division-of* {*a*..*b*} **shows** ∀ *k*∈*d*. *content k* = *0*
unfolding *forall-in-division*[*OF assms*(*2*)] **apply**(*rule*,*rule*,*rule*) **apply**(*drule division-ofD*(*2*)[*OF assms*(*2*)])
apply(*drule content-subset*) **unfolding** *assms*(*1*) **proof**− **case** *goal1* **thus** *?case* **using** *content-pos-le*[*of a b*] **by** *auto* **qed**

lemma *division-inter*: **assumes** *p1 division-of s1 p2 division-of* (*s2*::(*real*^'*a*) *set*)

shows $\{k1 \cap k2 \mid k1\ k2\ .k1 \in p1 \wedge k2 \in p2 \wedge k1 \cap k2 \neq \{\}\}$ *division-of* $(s1 \cap s2)$ (**is** *?A' division-of* -) **proof**−
let *?A* $= \{s.\ s \in\ (\lambda(k1,k2).\ k1 \cap k2)\ `\ (p1 \times p2) \wedge s \neq \{\}\}$ **have** $*$:*?A'* = *?A* **by** *auto*
show *?thesis* **unfolding** $*$ **proof**(*rule division-ofI*) **have** *?A* $\subseteq (\lambda(x,\ y).\ x \cap y)\ `\ (p1 \times p2)$ **by** *auto*
moreover have *finite* $(p1 \times p2)$ **using** *assms* **unfolding** *division-of-def* **by** *auto* **ultimately show** *finite ?A* **by** *auto*
have $*$:$\bigwedge s.\ \bigcup\{x{\in}s.\ x \neq \{\}\} = \bigcup s$ **by** *auto* **show** $\bigcup$ *?A* = *s1* $\cap$ *s2* **apply**(*rule set-ext*) **unfolding** $*$ **and** *Union-image-eq UN-iff*
using *division-ofD*(*6*)[*OF assms*(*1*)] **and** *division-ofD*(*6*)[*OF assms*(*2*)] **by** *auto*
{ fix *k* **assume** *k*∈*?A* **then obtain** *k1 k2* **where** *k*:*k* = *k1* ∩ *k2 k1*∈*p1 k2*∈*p2 k*≠{} **by** *auto* **thus** *k* ≠ {} **by** *auto*
show *k* ⊆ *s1* ∩ *s2* **using** *division-ofD*(*2*)[*OF assms*(*1*) *k*(*2*)] **and** *division-ofD*(*2*)[*OF assms*(*2*) *k*(*3*)] **unfolding** *k* **by** *auto*
guess *a1* **using** *division-ofD*(*4*)[*OF assms*(*1*) *k*(*2*)] **.. then guess** *b1* **.. note** *ab1*=*this*
guess *a2* **using** *division-ofD*(*4*)[*OF assms*(*2*) *k*(*3*)] **.. then guess** *b2* **.. note** *ab2*=*this*
show ∃ *a b.* *k* = {*a*..*b*} **unfolding** *k ab1 ab2* **unfolding** *inter-interval* **by** *auto*
} fix *k1 k2*
assume *k1*∈*?A* **then obtain** *x1 y1* **where** *k1*:*k1* = *x1* ∩ *y1 x1*∈*p1 y1*∈*p2 k1*≠{} **by** *auto*
assume *k2*∈*?A* **then obtain** *x2 y2* **where** *k2*:*k2* = *x2* ∩ *y2 x2*∈*p1 y2*∈*p2 k2*≠{} **by** *auto*
assume *k1* ≠ *k2* **hence** *th*:*x1*≠*x2* ∨ *y1*≠*y2* **unfolding** *k1 k2* **by** *auto*
have $*$:(*interior x1* ∩ *interior x2* = {} ∨ *interior y1* ∩ *interior y2* = {}) ⟹
interior(*x1* ∩ *y1*) ⊆ *interior*(*x1*) ⟹ *interior*(*x1* ∩ *y1*) ⊆ *interior*(*y1*) ⟹
interior(*x2* ∩ *y2*) ⊆ *interior*(*x2*) ⟹ *interior*(*x2* ∩ *y2*) ⊆ *interior*(*y2*)
⟹ *interior*(*x1* ∩ *y1*) ∩ *interior*(*x2* ∩ *y2*) = {} **by** *auto*
show *interior k1* ∩ *interior k2* = {} **unfolding** *k1 k2* **apply**(*rule* $*$) **defer apply**(*rule-tac*[*1*−*4*] *subset-interior*)
using *division-ofD*(*5*)[*OF assms*(*1*) *k1*(*2*) *k2*(*2*)]
using *division-ofD*(*5*)[*OF assms*(*2*) *k1*(*3*) *k2*(*3*)] **using** *th* **by** *auto* **qed qed**

lemma *division-inter-1*: **assumes** *d division-of i* {*a*..*b*::*real*^'*n*} ⊆ *i*
shows { {*a*..*b*} ∩ *k* |*k*. *k* ∈ *d* ∧ {*a*..*b*} ∩ *k* ≠ {} } *division-of* {*a*..*b*} **proof**(*cases* {*a*..*b*} = {})
case *True* **show** *?thesis* **unfolding** *True* **and** *division-of-trivial* **by** *auto* **next**
have $*$:{*a*..*b*} ∩ *i* = {*a*..*b*} **using** *assms*(*2*) **by** *auto*
case *False* **show** *?thesis* **using** *division-inter*[*OF division-of-self*[*OF False*] *assms*(*1*)] **unfolding** $*$ **by** *auto* **qed**

lemma *elementary-inter*: **assumes** *p1 division-of s p2 division-of* (*t*::(*real*^'*n*) *set*)
shows ∃ *p.* *p division-of* (*s* ∩ *t*)
by(*rule*,*rule division-inter*[*OF assms*])

lemma *elementary-inters*: **assumes** *finite f f*≠{} ∀ *s*∈*f*. ∃ *p.* *p division-of* (*s*::(*real*^'*n*)

set)
shows ∃ *p*. *p division-of* (⋂ *f*) **using** *assms* **apply**−**proof**(*induct f rule*:*finite-induct*)
case (*insert x f*) **show** *?case* **proof**(*cases f*={})
case *True* **thus** *?thesis* **unfolding** *True* **using** *insert* **by** *auto* **next**
case *False* **guess** *p* **using** *insert*(*3*)[*OF False insert*(*5*)[*unfolded ball-simps*,*THEN conjunct2*]] **..**
moreover guess *px* **using** *insert*(*5*)[*rule-format*,*OF insertI1*] **.. ultimately**
show *?thesis* **unfolding** *Inter-insert* **apply**(*rule-tac elementary-inter*) **by** *assumption*+ **qed qed** *auto*

lemma *division-disjoint-union*:
assumes *p1 division-of s1 p2 division-of s2 interior s1* ∩ *interior s2* = {}
shows (*p1* ∪ *p2*) *division-of* (*s1* ∪ *s2*) **proof**(*rule division-ofI*)
note *d1* = *division-ofD*[*OF assms*(*1*)] **and** *d2* = *division-ofD*[*OF assms*(*2*)]
show *finite* (*p1* ∪ *p2*) **using** *d1*(*1*) *d2*(*1*) **by** *auto*
show ⋃(*p1* ∪ *p2*) = *s1* ∪ *s2* **using** *d1*(*6*) *d2*(*6*) **by** *auto*
{ fix *k1 k2* **assume** *as*:*k1* ∈ *p1* ∪ *p2 k2* ∈ *p1* ∪ *p2 k1* ≠ *k2* **moreover let** *?g*=*interior k1* ∩ *interior k2* = {}
{ assume *as*:*k1*∈*p1 k2*∈*p2* **have** *?g* **using** *subset-interior*[*OF d1*(*2*)[*OF as*(*1*)]] *subset-interior*[*OF d2*(*2*)[*OF as*(*2*)]]
using *assms*(*3*) **by** *blast* **}** **moreover**
{ assume *as*:*k1*∈*p2 k2*∈*p1* **have** *?g* **using** *subset-interior*[*OF d1*(*2*)[*OF as*(*2*)]] *subset-interior*[*OF d2*(*2*)[*OF as*(*1*)]]
using *assms*(*3*) **by** *blast***}** **ultimately**
show *?g* **using** *d1*(*5*)[*OF - - as*(*3*)] **and** *d2*(*5*)[*OF - - as*(*3*)] **by** *auto* **}**
fix *k* **assume** *k*:*k* ∈ *p1* ∪ *p2* **show** *k* ⊆ *s1* ∪ *s2* **using** *k d1*(*2*) *d2*(*2*) **by** *auto*
show *k* ≠ {} **using** *k d1*(*3*) *d2*(*3*) **by** *auto* **show** ∃ *a b*. *k* = {*a*..*b*} **using** *k d1*(*4*) *d2*(*4*) **by** *auto* **qed**

lemma *partial-division-extend-1*:
assumes {*c*..*d*} ⊆ {*a*..*b*::*real*^'*n*} {*c*..*d*} ≠ {}
obtains *p* **where** *p division-of* {*a*..*b*} {*c*..*d*} ∈ *p*
proof− **def** *n* ≡ *CARD*('*n*) **have** *n*:*1* ≤ *n 0* < *n n* ≠ *0* **unfolding** *n-def* **by** *auto*
guess π **using** *ex-bij-betw-nat-finite-1*[*OF finite-UNIV*[**where** '*a*='*n*]] **..** **note** π=*this*
def π' ≡ *inv-into* {*1*..*n*} π
have π'*:bij-betw* π' *UNIV* {*1*..*n*} **using** *bij-betw-inv-into*[*OF* π] **unfolding** π'*-def n-def* **by** *auto*
hence π'*i*:⋀*i*. π' *i* ∈ {*1*..*n*} **unfolding** *bij-betw-def* **by** *auto*
have ππ'[*simp*]:⋀*i*. π (π' *i*) = *i* **unfolding** π'*-def* **apply**(*rule f-inv-into-f*) **unfolding** *n-def* **using** π **unfolding** *bij-betw-def* **by** *auto*
have π'π[*simp*]:⋀*i*. *i*∈{*1*..*n*} ⟹ π' (π *i*) = *i* **unfolding** π'*-def* **apply**(*rule inv-into-f-eq*) **using** π **unfolding** *n-def bij-betw-def* **by** *auto*
have {*c*..*d*} ≠ {} **using** *assms* **by** *auto*
let *?p1* = λ*l*. {(χ *i*. *if* π' *i* < *l then c*$*i else a*$*i*) .. (χ *i*. *if* π' *i* < *l then d*$*i else if* π' *i* = *l then c*$π *l else b*$*i*)}
let *?p2* = λ*l*. {(χ *i*. *if* π' *i* < *l then c*$*i else if* π' *i* = *l then d*$π *l else a*$*i*) .. (χ *i*. *if* π' *i* < *l then d*$*i else b*$*i*)}

let *?p* = $\{?p1\ l\ |l.\ l \in \{1..n+1\}\} \cup \{?p2\ l\ |l.\ l \in \{1..n+1\}\}$
have *abcd*:$\bigwedge i.\ a\ \$\ i \le c\ \$\ i \wedge c\$i \le d\$i \wedge d\ \$\ i \le b\ \$\ i$ **using** *assms* **unfolding** *subset-interval interval-eq-empty* **by**(*auto simp add*:*not-le not-less*)
show *?thesis* **apply**(*rule that*[*of ?p*]) **apply**(*rule division-ofI*)
proof− **have** $\bigwedge i.\ \pi'\ i < Suc\ n$
proof(*rule ccontr*,*unfold not-less*) **fix** *i* **assume** $Suc\ n \le \pi'\ i$
hence $\pi'\ i \notin \{1..n\}$ **by** *auto* **thus** *False* **using** π' **unfolding** *bij-betw-def* **by** *auto*
qed hence $c = (\chi\ i.\ if\ \pi'\ i < Suc\ n\ then\ c\ \$\ i\ else\ a\ \$\ i)$
$d = (\chi\ i.\ if\ \pi'\ i < Suc\ n\ then\ d\ \$\ i\ else\ if\ \pi'\ i = n + 1\ then\ c\ \$\ \pi\ (n + 1)\ else\ b\ \$\ i)$
unfolding *Cart-eq Cart-lambda-beta* **using** π' **unfolding** *bij-betw-def* **by** *auto*
thus *cdp*:$\{c..d\} \in ?p$ **apply**−**apply**(*rule UnI1*) **unfolding** *mem-Collect-eq* **apply**(*rule-tac x*=*n + 1* **in** *exI*) **by** *auto*
have $\bigwedge l.\ l \in \{1..n+1\} \Longrightarrow ?p1\ l \subseteq \{a..b\}$ $\bigwedge l.\ l \in \{1..n+1\} \Longrightarrow ?p2\ l \subseteq \{a..b\}$
unfolding *subset-eq* **apply**(*rule-tac*[!] *ballI*,*rule-tac*[!] *ccontr*)
proof− **fix** *l* **assume** *l*:$l \in \{1..n+1\}$ **fix** *x* **assume** $x \notin \{a..b\}$
then guess *i* **unfolding** *mem-interval not-all* **..** **note** *i*=*this*
show $x \in ?p1\ l \Longrightarrow False$ $x \in ?p2\ l \Longrightarrow False$ **unfolding** *mem-interval* **apply**(*erule-tac*[!] *x*=*i* **in** *allE*)
apply(*case-tac*[!] $\pi'\ i < l$, *case-tac*[!] $\pi'\ i = l$) **using** *abcd*[*of i*] *i* **by** *auto*
qed moreover have $\bigwedge x.\ x \in \{a..b\} \Longrightarrow x \in \bigcup ?p$
proof− **fix** *x* **assume** *x*:$x \in \{a..b\}$
{ presume $x \notin \{c..d\} \Longrightarrow x \in \bigcup ?p$ **thus** $x \in \bigcup ?p$ **using** *cdp* **by** *blast* **}**
let $?M = \{i.\ i \in \{1..n+1\} \wedge \neg\ (c\ \$\ \pi\ i \le x\ \$\ \pi\ i \wedge x\ \$\ \pi\ i \le d\ \$\ \pi\ i)\}$
assume $x \notin \{c..d\}$ **then guess** *i0* **unfolding** *mem-interval not-all* **..**
hence $\pi'\ i0 \in ?M$ **using** π' **unfolding** *bij-betw-def* **by**(*auto intro*!:*le-SucI*)
hence *M*:*finite ?M* $?M \neq \{\}$ **by** *auto*
def $l \equiv Min\ ?M$ **note** *l* = *Min-less-iff*[*OF M*,*unfolded l-def*[*symmetric*]] *Min-in*[*OF M*,*unfolded mem-Collect-eq l-def*[*symmetric*]]
Min-gr-iff[*OF M*,*unfolded l-def*[*symmetric*]]
have $x \in ?p1\ l \vee x \in ?p2\ l$ **using** *l*(*2*)[*THEN conjunct2*] **unfolding** *de-Morgan-conj not-le*
apply− **apply**(*erule disjE*) **apply**(*rule disjI1*) **defer apply**(*rule disjI2*)
proof− **assume** *as*:$x\ \$\ \pi\ l < c\ \$\ \pi\ l$
show $x \in ?p1\ l$ **unfolding** *mem-interval Cart-lambda-beta*
proof case *goal1* **have** $\pi'\ i \in \{1..n\}$ **using** π' **unfolding** *bij-betw-def not-le* **by** *auto*
thus *?case* **using** *as x*[*unfolded mem-interval*,*rule-format*,*of i*]
apply *auto* **using** *l*(*3*)[*of* $\pi'\ i$] **by**(*auto elim*!:*ballE*[**where** *x*=$\pi'\ i$])
qed
next assume *as*:$x\ \$\ \pi\ l > d\ \$\ \pi\ l$
show $x \in ?p2\ l$ **unfolding** *mem-interval Cart-lambda-beta*
proof case *goal1* **have** $\pi'\ i \in \{1..n\}$ **using** π' **unfolding** *bij-betw-def not-le* **by** *auto*
thus *?case* **using** *as x*[*unfolded mem-interval*,*rule-format*,*of i*]
apply *auto* **using** *l*(*3*)[*of* $\pi'\ i$] **by**(*auto elim*!:*ballE*[**where** *x*=$\pi'\ i$])
qed qed
thus $x \in \bigcup ?p$ **using** *l*(*2*) **by** *blast*

qed ultimately show $\bigcup ?p = \{a..b\}$ **apply**−**apply**(*rule*) **defer apply**(*rule*) **by**(*assumption*,*blast*)

show *finite ?p* **by** *auto*
fix k **assume** k:$k \in ?p$ **then obtain** l **where** l:$k = ?p1\ l \vee k = ?p2\ l\ l \in \{1..n + 1\}$ **by** *auto*
show $k \subseteq \{a..b\}$ **apply**(*rule*,*unfold mem-interval*,*rule*,*rule*)
proof− **fix** i::$'n$ **and** x **assume** $x \in k$ **moreover have** $\pi'\ i < l \vee \pi'\ i = l \vee \pi'\ i > l$ **by** *auto*
ultimately show $a\$i \leq x\$i\ x\$i \leq b\$i$ **using** *abcd*[*of i*] **using** l **by**(*auto elim*:*disjE elim*!:*allE*[**where** x=i] *simp add*:*vector-le-def*)
qed have $\bigwedge l.\ ?p1\ l \neq \{\}\ \bigwedge l.\ ?p2\ l \neq \{\}$ **unfolding** *interval-eq-empty not-ex* **apply**(*rule-tac*[!] *allI*)
proof− **case** *goal1* **thus** *?case* **using** *abcd*[*of x*] **by** *auto*
next **case** *goal2* **thus** *?case* **using** *abcd*[*of x*] **by** *auto*
qed thus $k \neq \{\}$ **using** k **by** *auto*
show $\exists\ a\ b.\ k = \{a..b\}$ **using** k **by** *auto*
fix k' **assume** k':$k' \in ?p\ k \neq k'$ **then obtain** l' **where** l':$k' = ?p1\ l' \vee k' = ?p2\ l'\ l' \in \{1..n + 1\}$ **by** *auto*
{ fix $k\ k'\ l\ l'$
assume k:$k \in ?p$ **and** l:$k = ?p1\ l \vee k = ?p2\ l\ l \in \{1..n + 1\}$
assume k':$k' \in ?p\ k \neq k'$ **and** l':$k' = ?p1\ l' \vee k' = ?p2\ l'\ l' \in \{1..n + 1\}$
assume $l \leq l'$ **fix** x
have $x \notin interior\ k \cap interior\ k'$
proof(*rule*,*cases* $l' = n+1$) **assume** x:$x \in interior\ k \cap interior\ k'$
case *True* **hence** $\bigwedge i.\ \pi'\ i < l'$ **using** $\pi'i$ **by**(*auto simp add*:*less-Suc-eq-le*)
hence k':$k' = \{c..d\}$ **using** $l'(1)\ \pi'i$ **by**(*auto simp add*:*Cart-nth-inverse*)
have ln:$l < n + 1$
proof(*rule ccontr*) **case** *goal1* **hence** $l2$:$l = n+1$ **using** l **by** *auto*
hence $\bigwedge i.\ \pi'\ i < l$ **using** $\pi'i$ **by**(*auto simp add*:*less-Suc-eq-le*)
hence $k = \{c..d\}$ **using** $l(1)\ \pi'i$ **by**(*auto simp add*:*Cart-nth-inverse*)
thus *False* **using** $\langle k \neq k' \rangle\ k'$ **by** *auto*
qed have $**$:$\pi'\ (\pi\ l) = l$ **using** $\pi'\pi$[*of l*] **using** $l\ ln$ **by** *auto*
have $x\ \$\ \pi\ l < c\ \$\ \pi\ l \vee d\ \$\ \pi\ l < x\ \$\ \pi\ l$ **using** $l(1)$ **apply**−
proof(*erule disjE*)
assume *as*:$k = ?p1\ l$ **note** $* = conjunct1$[*OF x*[*unfolded as Int-iff interior-closed-interval mem-interval*],*rule-format*]
show *?thesis* **using** $*$[*of* $\pi\ l$] **using** ln **unfolding** *Cart-lambda-beta* $**$ **by** *auto*
next assume *as*:$k = ?p2\ l$ **note** $* = conjunct1$[*OF x*[*unfolded as Int-iff interior-closed-interval mem-interval*],*rule-format*]
show *?thesis* **using** $*$[*of* $\pi\ l$] **using** ln **unfolding** *Cart-lambda-beta* $**$ **by** *auto*
qed thus *False* **using** x **unfolding** k' **unfolding** *Int-iff interior-closed-interval mem-interval*
by(*auto elim*!:*allE*[**where** x=$\pi\ l$])
next case *False* **hence** $l < n + 1$ **using** $l'(2)$ **using** $\langle l \leq l' \rangle$ **by** *auto*
hence ln:$l \in \{1..n\}\ l' \in \{1..n\}$ **using** $l\ l'$ *False* **by** *auto*
note $\pi l = \pi'\pi$[*OF* $ln(1)$] $\pi'\pi$[*OF* $ln(2)$]

assume *x*:*x* ∈ *interior k* ∩ *interior k′*
show *False* **using** *l*(*1*) *l′*(*1*) **apply**−
proof(*erule-tac*[!] *disjE*)+
assume *as*:*k* = *?p1 l k′* = *?p1 l′*
note ∗ = *x*[*unfolded as Int-iff interior-closed-interval mem-interval*]
have *l* ≠ *l′* **using** *k′*(*2*)[*unfolded as*] **by** *auto*
thus *False* **using** ∗ **by**(*smt Cart-lambda-beta* π*l*)
next assume *as*:*k* = *?p2 l k′* = *?p2 l′*
note ∗ = *conjunctD2*[*OF x*[*unfolded as Int-iff interior-closed-interval mem-interval*],*rule-format*]
have *l* ≠ *l′* **apply**(*rule*) **using** *k′*(*2*)[*unfolded as*] **by** *auto*
thus *False* **using** ∗[*of* π *l*] ∗[*of* π *l′*]
unfolding *Cart-lambda-beta* π*l* **using** ‹*l* ≤ *l′*› **by** *auto*
next assume *as*:*k* = *?p1 l k′* = *?p2 l′*
note ∗ = *conjunctD2*[*OF x*[*unfolded as Int-iff interior-closed-interval mem-interval*],*rule-format*]
show *False* **using** ∗[*of* π *l*] ∗[*of* π *l′*]
unfolding *Cart-lambda-beta* π*l* **using** ‹*l* ≤ *l′*› **using** *abcd*[*of* π *l′*] **by** *smt*
next assume *as*:*k* = *?p2 l k′* = *?p1 l′*
note ∗ = *conjunctD2*[*OF x*[*unfolded as Int-iff interior-closed-interval mem-interval*],*rule-format*]
show *False* **using** ∗[*of* π *l*] ∗[*of* π *l′*]
unfolding *Cart-lambda-beta* π*l* **using** ‹*l* ≤ *l′*› **using** *abcd*[*of* π *l′*] **by** *smt*
qed qed }
from *this*[*OF k l k′ l′*] *this*[*OF k′*(*1*) *l′ k - l*] **have** ⋀*x*. *x* ∉ *interior k* ∩ *interior k′*
apply − **apply**(*cases l′* ≤ *l*) **using** *k′*(*2*) **by** *auto*
thus *interior k* ∩ *interior k′* = {} **by** *auto*
qed qed

lemma *partial-division-extend-interval*: **assumes** *p division-of* (⋃ *p*) (⋃ *p*) ⊆ {*a*..*b*}
obtains *q* **where** *p* ⊆ *q q division-of* {*a*..*b*::*real*^′*n*} **proof**(*cases p* = {})
case *True* **guess** *q* **apply**(*rule elementary-interval*[*of a b*]) **.**
thus *?thesis* **apply**− **apply**(*rule that*[*of q*]) **unfolding** *True* **by** *auto* **next**
case *False* **note** *p* = *division-ofD*[*OF assms*(*1*)]
have ∗:∀ *k*∈*p*. ∃ *q*. *q division-of* {*a*..*b*} ∧ *k*∈*q* **proof case** *goal1*
guess *c* **using** *p*(*4*)[*OF goal1*] **.. then guess** *d* **.. note** *cd*- = *this*
have ∗:{*c*..*d*} ⊆ {*a*..*b*} {*c*..*d*} ≠ {} **using** *p*(*2*,*3*)[*OF goal1*, *unfolded cd*-] **using** *assms*(*2*) **by** *auto*
guess *q* **apply**(*rule partial-division-extend-1*[*OF* ∗]) **. thus** *?case* **unfolding** *cd*- **by** *auto* **qed**
guess *q* **using** *bchoice*[*OF* ∗] **.. note** *q* = *conjunctD2*[*OF this*[*rule-format*]]
have ⋀*x*. *x*∈*p* ⟹ ∃ *d*. *d division-of* ⋃(*q x* − {*x*}) **apply**(*rule*,*rule-tac p*=*q x* **in** *division-of-subset*) **proof**−
fix *x* **assume** *x*:*x*∈*p* **show** *q x division-of* ⋃ *q x* **apply**−**apply**(*rule division-ofI*)
using *division-ofD*[*OF q*(*1*)[*OF x*]] **by** *auto* **show** *q x* − {*x*} ⊆ *q x* **by** *auto*
qed

hence $\exists\, d.\ d$ *division-of* $\bigcap\ ((\lambda i.\ \bigcup(q\ i - \{i\}))\ `\ p)$ **apply**$-$ **apply**(*rule elementary-inters*)
apply(*rule finite-imageI*[*OF p*(*1*)]) **unfolding** *image-is-empty* **apply**(*rule False*) **by** *auto*
then guess d **..** **note** $d = this$
show *?thesis* **apply**(*rule that*[*of* $d \cup p$]) **proof**$-$
have $*$:$\bigwedge s\ f\ t.\ s \neq \{\} \Longrightarrow (\forall i{\in}s.\ f\ i \cup i = t) \Longrightarrow t = \bigcap\ (f\ `\ s) \cup (\bigcup s)$ **by** *auto*
have $*$:$\{a..b\} = \bigcap\ (\lambda i.\ \bigcup(q\ i\ -\ \{i\}))\ `\ p \cup \bigcup p$ **apply**(*rule* $*$[*OF False*]) **proof fix** i **assume** i:$i{\in}p$
show $\bigcup(q\ i - \{i\}) \cup i = \{a..b\}$ **using** *division-ofD*(*6*)[*OF q*(*1*)[*OF i*]] **using** *q*(*2*)[*OF i*] **by** *auto* **qed**
show $d \cup p$ *division-of* $\{a..b\}$ **unfolding** $*$ **apply**(*rule division-disjoint-union*[*OF d assms*(*1*)])
apply(*rule inter-interior-unions-intervals*) **apply**(*rule p open-interior ballI*)$+$ **proof**(*assumption*,*rule*)
fix k **assume** k:$k{\in}p$ **have** $*$:$\bigwedge u\ t\ s.\ u \subseteq s \Longrightarrow s \cap t = \{\} \Longrightarrow u \cap t = \{\}$ **by** *auto*
show *interior* $(\bigcap(\lambda i.\ \bigcup(q\ i\ -\ \{i\}))\ `\ p) \cap$ *interior* $k = \{\}$ **apply**(*rule* $*$[*of - interior* $(\bigcup(q\ k\ -\ \{k\}))$])
defer apply(*subst Int-commute*) **apply**(*rule inter-interior-unions-intervals*) **proof**$-$ **note** *qk*=*division-ofD*[*OF q*(*1*)[*OF k*]]
show *finite* $(q\ k\ -\ \{k\})$ *open* $(interior\ k)$ $\forall t{\in}q\ k\ -\ \{k\}.\ \exists\, a\ b.\ t = \{a..b\}$ **using** *qk* **by** *auto*
show $\forall\, t{\in}q\ k\ -\ \{k\}.\ interior\ k \cap interior\ t = \{\}$ **using** *qk*(*5*) **using** *q*(*2*)[*OF k*] **by** *auto*
have $*$:$\bigwedge x\ s.\ x \in s \Longrightarrow \bigcap s \subseteq x$ **by** *auto* **show** *interior* $(\bigcap(\lambda i.\ \bigcup(q\ i\ -\ \{i\}))\ `\ p) \subseteq$ *interior* $(\bigcup(q\ k\ -\ \{k\}))$
apply(*rule subset-interior* $*$)$+$ **using** k **by** *auto* **qed qed qed** *auto* **qed**

lemma *elementary-bounded*[*dest*]: p *division-of* $s \Longrightarrow$ *bounded* $(s{::}(real\hat{}'n)\ set)$
unfolding *division-of-def* **by**(*metis bounded-Union bounded-interval*)

lemma *elementary-subset-interval*: p *division-of* $s \Longrightarrow \exists\, a\ b.\ s \subseteq \{a..b{::}real\hat{}'n\}$
by(*meson elementary-bounded bounded-subset-closed-interval*)

lemma *division-union-intervals-exists*: **assumes** $\{a..b{::}real\hat{}'n\} \neq \{\}$
obtains p **where** $(insert\ \{a..b\}\ p)$ *division-of* $(\{a..b\} \cup \{c..d\})$ **proof**(*cases* $\{c..d\} = \{\}$)
case *True* **show** *?thesis* **apply**(*rule that*[*of* $\{\}$]) **unfolding** *True* **using** *assms* **by** *auto* **next**
case *False* **note** *false*=*this* **show** *?thesis* **proof**(*cases* $\{a..b\} \cap \{c..d\} = \{\}$)
have $*$:$\bigwedge a\ b.\ \{a,b\} = \{a\} \cup \{b\}$ **by** *auto*
case *True* **show** *?thesis* **apply**(*rule that*[*of* $\{\{c..d\}\}$]) **unfolding** $*$ **apply**(*rule division-disjoint-union*)
using *false True assms* **using** *interior-subset* **by** *auto* **next**
case *False* **obtain** $u\ v$ **where** uv:$\{a..b\} \cap \{c..d\} = \{u..v\}$ **unfolding** *inter-interval* **by** *auto*
have $*$:$\{u..v\} \subseteq \{c..d\}$ **using** uv **by** *auto*
guess p **apply**(*rule partial-division-extend-1*[*OF* $*$ *False*[*unfolded uv*]]) **. note**

p=this division-ofD[OF this(1)]
have *∗:{a..b} ∪ {c..d} = {a..b} ∪ ⋃(p − {{u..v}}) ⋀x s. insert x s = {x} ∪ s*
using *p(8)* **unfolding** *uv[THEN sym]* **by** *auto*
show *thesis* **apply**(*rule that[of p − {{u..v}}]*) **unfolding** *∗(1)* **apply**(*subst ∗(2)*)
apply(*rule division-disjoint-union*)
apply(*rule*,*rule assms*) **apply**(*rule division-of-subset[of p]*) **apply**(*rule division-of-union-self[OF p(1)]*) **defer**
unfolding *interior-inter[THEN sym]* **proof**−
have *∗:⋀cd p uv ab. p ⊆ cd ⟹ ab ∩ cd = uv ⟹ ab ∩ p = uv ∩ p* **by** *auto*
have *interior ({a..b} ∩ ⋃(p − {{u..v}})) = interior({u..v} ∩ ⋃(p − {{u..v}}))*

apply(*rule arg-cong[of - - interior]*) **apply**(*rule ∗[OF - uv]*) **using** *p(8)* **by** *auto*
also have *. . . = {}* **unfolding** *interior-inter* **apply**(*rule inter-interior-unions-intervals*) **using** *p(6) p(7)[OF p(2)] p(3)* **by** *auto*
finally show *interior ({a..b} ∩ ⋃(p − {{u..v}})) = {}* **by** *assumption* **qed** *auto* **qed qed**

lemma *division-of-unions*: **assumes** *finite f ⋀p. p∈f ⟹ p division-of (⋃p)*
⋀k1 k2. ⟦k1 ∈ ⋃f; k2 ∈ ⋃f; k1 ≠ k2⟧ ⟹ interior k1 ∩ interior k2 = {}
shows *⋃f division-of ⋃⋃f* **apply**(*rule division-ofI*) **prefer** *5* **apply**(*rule assms(3)|assumption*)+
apply(*rule finite-Union assms(1)*)+ **prefer** *3* **apply**(*erule UnionE*) **apply**(*rule-tac s=X* **in** *division-ofD(3)[OF assms(2)]*)
using *division-ofD[OF assms(2)]* **by** *auto*

lemma *elementary-union-interval*: **assumes** *p division-of ⋃p*
obtains *q* **where** *q division-of ({a..b::real^'n} ∪ ⋃p)* **proof**−
note *assm=division-ofD[OF assms]*
have *lem1:⋀f s. ⋃⋃ (f ‘ s) = ⋃(λx.⋃(f x)) ‘ s* **by** *auto*
have *lem2:⋀f s. f ≠ {} ⟹ ⋃{s ∪ t |t. t ∈ f} = s ∪ ⋃f* **by** *auto*
{ presume *p={} ⟹ thesis {a..b} = {} ⟹ thesis {a..b} ≠ {} ⟹ interior {a..b} = {} ⟹ thesis*
p≠{} ⟹ interior {a..b}≠{} ⟹ {a..b} ≠ {} ⟹ thesis
thus *thesis* **by** *auto*
next assume *as:p={}* **guess** *p* **apply**(*rule elementary-interval[of a b]*) **.**
thus *thesis* **apply**(*rule-tac that[of p]*) **unfolding** *as* **by** *auto*
next assume *as:{a..b}={}* **show** *thesis* **apply**(*rule that*) **unfolding** *as* **using** *assms* **by** *auto*
next assume *as:interior {a..b} = {} {a..b} ≠ {}*
show *thesis* **apply**(*rule that[of insert {a..b} p]*,*rule division-ofI*)
unfolding *finite-insert* **apply**(*rule assm(1)*) **unfolding** *Union-insert*
using *assm(2−4) as* **apply**− **by**(*fastsimp dest: assm(5)*)+
next assume *as:p ≠ {} interior {a..b} ≠ {} {a..b}≠{}*
have *∀k∈p. ∃q. (insert {a..b} q) division-of ({a..b} ∪ k)* **proof case** *goal1*
from *assm(4)[OF this]* **guess** *c* **.. then guess** *d* **..**
thus *?case* **apply**−**apply**(*rule division-union-intervals-exists[OF as(3)*,*of c d]*)
by *auto*
qed from *bchoice[OF this]* **guess** *q* **.. note** *q=division-ofD[OF this[rule-format]]*
let *?D = ⋃{insert {a..b} (q k) | k. k ∈ p}*

show *thesis* **apply**(*rule that*[*of ?D*]) **proof**(*rule division-ofI*)
have ∗:{*insert* {*a..b*} (*q k*) |*k*. *k* ∈ *p*} = (λ*k*. *insert* {*a..b*} (*q k*)) ‘ *p* **by** *auto*
show *finite ?D* **apply**(*rule finite-Union*) **unfolding** ∗ **apply**(*rule finite-imageI*)
using *assm*(*1*) *q*(*1*) **by** *auto*
show ⋃ *?D* = {*a..b*} ∪ ⋃ *p* **unfolding** ∗ *lem1* **unfolding** *lem2*[*OF as*(*1*), *of* {*a..b*},*THEN sym*]
using *q*(*6*) **by** *auto*
fix *k* **assume** *k*:*k*∈*?D* **thus** *k* ⊆ {*a..b*} ∪ ⋃ *p* **using** *q*(*2*) **by** *auto*
show *k* ≠ {} **using** *q*(*3*) *k* **by** *auto* **show** ∃ *a b*. *k* = {*a..b*} **using** *q*(*4*) *k* **by** *auto*
fix *k′* **assume** *k′*:*k′*∈*?D k*≠*k′*
obtain *x* **where** *x*: *k* ∈*insert* {*a..b*} (*q x*) *x*∈*p* **using** *k* **by** *auto*
obtain *x′* **where** *x′*:*k′*∈*insert* {*a..b*} (*q x′*) *x′*∈*p* **using** *k′* **by** *auto*
show *interior k* ∩ *interior k′* = {} **proof**(*cases x*=*x′*)
case *True* **show** *?thesis* **apply**(*rule q*(*5*)) **using** *x x′ k′* **unfolding** *True* **by** *auto*
next case *False*
{ presume *k* = {*a..b*} ⟹ *?thesis k′* = {*a..b*} ⟹ *?thesis*
k ≠ {*a..b*} ⟹ *k′* ≠ {*a..b*} ⟹ *?thesis*
thus *?thesis* **by** *auto* **}**
{ assume *as′*:*k* = {*a..b*} **show** *?thesis* **apply**(*rule q*(*5*)) **using** *x′ k′*(*2*) **unfolding** *as′* **by** *auto* **}**
{ assume *as′*:*k′* = {*a..b*} **show** *?thesis* **apply**(*rule q*(*5*)) **using** *x k′*(*2*) **unfolding** *as′* **by** *auto* **}**
assume *as′*:*k* ≠ {*a..b*} *k′* ≠ {*a..b*}
guess *c* **using** *q*(*4*)[*OF x*(*2*,*1*)] **.. then guess** *d* **.. note** *c-d*=*this*
have *interior k* ∩ *interior* {*a..b*} = {} **apply**(*rule q*(*5*)) **using** *x k′*(*2*) **using** *as′* **by** *auto*
hence *interior k* ⊆ *interior x* **apply**−
apply(*rule interior-subset-union-intervals*[*OF c-d - as*(*2*) *q*(*2*)[*OF x*(*2*,*1*)]]) **by** *auto* **moreover**
guess *c* **using** *q*(*4*)[*OF x′*(*2*,*1*)] **.. then guess** *d* **.. note** *c-d*=*this*
have *interior k′* ∩ *interior* {*a..b*} = {} **apply**(*rule q*(*5*)) **using** *x′ k′*(*2*) **using** *as′* **by** *auto*
hence *interior k′* ⊆ *interior x′* **apply**−
apply(*rule interior-subset-union-intervals*[*OF c-d - as*(*2*) *q*(*2*)[*OF x′*(*2*,*1*)]]) **by** *auto*
ultimately show *?thesis* **using** *assm*(*5*)[*OF x*(*2*) *x′*(*2*) *False*] **by** *auto*
qed qed } qed

lemma *elementary-unions-intervals*:
assumes *finite f* ⋀*s*. *s* ∈ *f* ⟹ ∃ *a b*. *s* = {*a..b*::*real*^′*n*}
obtains *p* **where** *p division-of* (⋃ *f*) **proof**−
have ∃ *p*. *p division-of* (⋃ *f*) **proof**(*induct-tac f rule*:*finite-subset-induct*)
show ∃ *p*. *p division-of* ⋃ {} **using** *elementary-empty* **by** *auto*
fix *x F* **assume** *as*:*finite F x* ∉ *F* ∃ *p*. *p division-of* ⋃ *F x*∈*f*
from *this*(*3*) **guess** *p* **.. note** *p*=*this*
from *assms*(*2*)[*OF as*(*4*)] **guess** *a* **.. then guess** *b* **.. note** *ab*=*this*
have ∗:⋃ *F* = ⋃ *p* **using** *division-ofD*[*OF p*] **by** *auto*

show $\exists p.\ p$ *division-of* $\bigcup$*insert* $x\ F$ **using** *elementary-union-interval*[*OF p*[*unfolded* $*$], *of a b*]
unfolding *Union-insert ab* $*$ **by** *auto*
qed(*insert assms*,*auto*) **thus** *?thesis* **apply**−**apply**(*erule exE*,*rule that*) **by** *auto*
qed

lemma *elementary-union*: **assumes** *ps division-of s pt division-of* (*t*::(*real*^'*n*) *set*)
obtains *p* **where** *p division-of* (*s* ∪ *t*)
proof− **have** *s* ∪ *t* = ⋃*ps* ∪ ⋃*pt* **using** *assms* **unfolding** *division-of-def* **by** *auto*
hence $*$:⋃(*ps* ∪ *pt*) = *s* ∪ *t* **by** *auto*
show *?thesis* **apply**−**apply**(*rule elementary-unions-intervals*[*of ps*∪*pt*])
unfolding $*$ **prefer** *3* **apply**(*rule-tac p*=*p* **in** *that*)
using *assms*[*unfolded division-of-def*] **by** *auto* **qed**

lemma *partial-division-extend*: **fixes** *t*::(*real*^'*n*) *set*
assumes *p division-of s q division-of t s* ⊆ *t*
obtains *r* **where** *p* ⊆ *r r division-of t* **proof**−
note *divp* = *division-ofD*[*OF assms*(*1*)] **and** *divq* = *division-ofD*[*OF assms*(*2*)]
obtain *a b* **where** *ab*:*t*⊆{*a*..*b*} **using** *elementary-subset-interval*[*OF assms*(*2*)] **by** *auto*
guess *r1* **apply**(*rule partial-division-extend-interval*) **apply**(*rule assms*(*1*)[*unfolded divp*(*6*)[*THEN sym*]])
apply(*rule subset-trans*) **by**(*rule ab assms*[*unfolded divp*(*6*)[*THEN sym*]])+
note *r1* = *this division-ofD*[*OF this*(*2*)]
guess *p'* **apply**(*rule elementary-unions-intervals*[*of r1* − *p*]) **using** *r1*(*3*,*6*) **by** *auto*
then obtain *r2* **where** *r2*:*r2 division-of* (⋃(*r1* − *p*)) ∩ (⋃*q*)
apply− **apply**(*drule elementary-inter*[*OF - assms*(*2*)[*unfolded divq*(*6*)[*THEN sym*]]]) **by** *auto*
{ fix *x* **assume** *x*:*x*∈*t x*∉*s*
hence *x*∈⋃*r1* **unfolding** *r1* **using** *ab* **by** *auto*
then guess *r* **unfolding** *Union-iff* **.. note** *r*=*this* **moreover**
have *r* ∉ *p* **proof assume** *r*∈*p* **hence** *x*∈*s* **using** *divp*(*2*) *r* **by** *auto*
thus *False* **using** *x* **by** *auto* **qed**
ultimately have *x*∈⋃(*r1* − *p*) **by** *auto* **}**
hence $*$:*t* = ⋃*p* ∪ (⋃(*r1* − *p*) ∩ ⋃*q*) **unfolding** *divp divq* **using** *assms*(*3*) **by** *auto*
show *?thesis* **apply**(*rule that*[*of p* ∪ *r2*]) **unfolding** $*$ **defer apply**(*rule division-disjoint-union*)
unfolding *divp*(*6*) **apply**(*rule assms r2*)+
proof− **have** *interior s* ∩ *interior* (⋃(*r1*−*p*)) = {}
proof(*rule inter-interior-unions-intervals*)
show *finite* (*r1* − *p*) *open* (*interior s*) ∀ *t*∈*r1*−*p*. ∃ *a b*. *t* = {*a*..*b*} **using** *r1* **by** *auto*
have $*$:⋀*s*. (⋀*x*. *x* ∈ *s* ⟹ *False*) ⟹ *s* = {} **by** *auto*
show ∀ *t*∈*r1*−*p*. *interior s* ∩ *interior t* = {} **proof**(*rule*)
fix *m x* **assume** *as*:*m*∈*r1*−*p*
have *interior m* ∩ *interior* (⋃*p*) = {} **proof**(*rule inter-interior-unions-intervals*)

show *finite p open (interior m)* $\forall\, t{\in}p.\ \exists\, a\ b.\ t = \{a..b\}$ **using** *divp* **by** *auto*
show $\forall\, t{\in}p.$ *interior m* $\cap$ *interior t* $= \{\}$ **apply**(*rule*, *rule r1*(*7*)) **using** *as* **using** *r1* **by** *auto*
qed thus *interior s* $\cap$ *interior m* $= \{\}$ **unfolding** *divp* **by** *auto*
qed qed
thus *interior s* $\cap$ *interior* $(\bigcup(r1{-}p) \cap (\bigcup q)) = \{\}$ **using** *interior-subset* **by** *auto*
qed *auto* **qed**

23.7 Tagged (partial) divisions.

definition *tagged-partial-division-of* (**infixr** *tagged'-partial'-division'-of 40*) **where**
(*s tagged-partial-division-of i*) $\equiv$
finite s $\wedge$
$(\forall\, x\ k.\ (x,k) \in s \longrightarrow x \in k \wedge k \subseteq i \wedge (\exists\, a\ b.\ k = \{a..b\})) \wedge$
$(\forall\, x1\ k1\ x2\ k2.\ (x1,k1) \in s \wedge (x2,k2) \in s \wedge ((x1,k1) \neq (x2,k2))$
$\longrightarrow$ *interior*$(k1) \cap$ *interior*$(k2) = \{\})$

lemma *tagged-partial-division-ofD*[*dest*]: **assumes** *s tagged-partial-division-of i*
shows *finite s* $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow x \in k$ $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow k \subseteq i$
$\bigwedge x\ k.\ (x,k) \in s \Longrightarrow \exists\, a\ b.\ k = \{a..b\}$
$\bigwedge x1\ k1\ x2\ k2.\ (x1,k1) \in s \Longrightarrow (x2,k2) \in s \Longrightarrow (x1,k1) \neq (x2,k2) \Longrightarrow$ *interior*$(k1) \cap$ *interior*$(k2) = \{\}$
using *assms* **unfolding** *tagged-partial-division-of-def* **apply**$-$ **by** *blast*$+$

definition *tagged-division-of* (**infixr** *tagged'-division'-of 40*) **where**
(*s tagged-division-of i*) $\equiv$
(*s tagged-partial-division-of i*) $\wedge$ $(\bigcup\{k.\ \exists\, x.\ (x,k) \in s\} = i)$

lemma *tagged-division-of-finite*[*dest*]: *s tagged-division-of i* $\Longrightarrow$ *finite s*
unfolding *tagged-division-of-def tagged-partial-division-of-def* **by** *auto*

lemma *tagged-division-of*:
(*s tagged-division-of i*) $\longleftrightarrow$
finite s $\wedge$
$(\forall\, x\ k.\ (x,k) \in s$
$\longrightarrow x \in k \wedge k \subseteq i \wedge (\exists\, a\ b.\ k = \{a..b\})) \wedge$
$(\forall\, x1\ k1\ x2\ k2.\ (x1,k1) \in s \wedge (x2,k2) \in s \wedge {\sim}((x1,k1) = (x2,k2))$
$\longrightarrow$ *interior*$(k1) \cap$ *interior*$(k2) = \{\})) \wedge$
$(\bigcup\{k.\ \exists\, x.\ (x,k) \in s\} = i)$
unfolding *tagged-division-of-def tagged-partial-division-of-def* **by** *auto*

lemma *tagged-division-ofI*: **assumes**
finite s $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow x \in k$ $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow k \subseteq i$ $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow \exists\, a\ b.\ k = \{a..b\}$
$\bigwedge x1\ k1\ x2\ k2.\ (x1,k1) \in s \Longrightarrow (x2,k2) \in s \Longrightarrow {\sim}((x1,k1) = (x2,k2)) \Longrightarrow$ *interior*$(k1) \cap$ *interior*$(k2) = \{\})$
$(\bigcup\{k.\ \exists\, x.\ (x,k) \in s\} = i)$

shows *s tagged-division-of i*
unfolding *tagged-division-of* **apply**(*rule*) **defer apply** *rule*
apply(*rule allI impI conjI assms*)+ **apply** *assumption*
apply(*rule*, *rule assms*, *assumption*) **apply**(*rule assms*, *assumption*)
using *assms*(*1*,*5*−) **apply**− **by** *blast*+

lemma *tagged-division-ofD*[*dest*]: **assumes** *s tagged-division-of i*
shows *finite s* $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow x \in k$ $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow k \subseteq i$ $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow \exists\, a\ b.\ k = \{a..b\}$
$\bigwedge x1\ k1\ x2\ k2.\ (x1,k1) \in s \Longrightarrow (x2,k2) \in s \Longrightarrow \neg((x1,k1) = (x2,k2)) \Longrightarrow (interior(k1) \cap interior(k2) = \{\})$
$(\bigcup \{k.\ \exists\, x.\ (x,k) \in s\} = i)$ **using** *assms* **unfolding** *tagged-division-of* **apply**− **by** *blast*+

lemma *division-of-tagged-division*: **assumes***s tagged-division-of i* **shows** (*snd* ‘ *s*) *division-of i*
proof(*rule division-ofI*) **note** *assm*=*tagged-division-ofD*[*OF assms*]
show $\bigcup$ *snd* ‘ *s* = *i finite* (*snd* ‘ *s*) **using** *assm* **by** *auto*
fix *k* **assume** *k*:*k* ∈ *snd* ‘ *s* **then obtain** *xk* **where** *xk*:(*xk*, *k*) ∈ *s* **by** *auto*
thus *k* ⊆ *i k* ≠ {} ∃ *a b*. *k* = {*a*..*b*} **using** *assm* **apply**− **by** *fastsimp*+
fix *k*′ **assume** *k*′:*k*′ ∈ *snd* ‘ *s k* ≠ *k*′ **from** *this*(*1*) **obtain** *xk*′ **where** *xk*′:(*xk*′, *k*′) ∈ *s* **by** *auto*
thus *interior k* ∩ *interior k*′ = {} **apply**−**apply**(*rule assm*(*5*)) **apply**(*rule xk xk*′)+ **using** *k*′ **by** *auto*
qed

lemma *partial-division-of-tagged-division*: **assumes** *s tagged-partial-division-of i*
shows (*snd* ‘ *s*) *division-of* $\bigcup$(*snd* ‘ *s*)
proof(*rule division-ofI*) **note** *assm*=*tagged-partial-division-ofD*[*OF assms*]
show *finite* (*snd* ‘ *s*) $\bigcup$ *snd* ‘ *s* = $\bigcup$ *snd* ‘ *s* **using** *assm* **by** *auto*
fix *k* **assume** *k*:*k* ∈ *snd* ‘ *s* **then obtain** *xk* **where** *xk*:(*xk*, *k*) ∈ *s* **by** *auto*
thus *k*≠{} ∃ *a b*. *k* = {*a*..*b*} *k* ⊆ $\bigcup$ *snd* ‘ *s* **using** *assm* **by** *auto*
fix *k*′ **assume** *k*′:*k*′ ∈ *snd* ‘ *s k* ≠ *k*′ **from** *this*(*1*) **obtain** *xk*′ **where** *xk*′:(*xk*′, *k*′) ∈ *s* **by** *auto*
thus *interior k* ∩ *interior k*′ = {} **apply**−**apply**(*rule assm*(*5*)) **apply**(*rule xk xk*′)+ **using** *k*′ **by** *auto*
qed

lemma *tagged-partial-division-subset*: **assumes** *s tagged-partial-division-of i t* ⊆ *s*
shows *t tagged-partial-division-of i*
using *assms* **unfolding** *tagged-partial-division-of-def* **using** *finite-subset*[*OF assms*(*2*)] **by** *blast*

lemma *setsum-over-tagged-division-lemma*: **fixes** *d*::(*real*^′*m*) *set* ⇒ ′*a*::*real-normed-vector*
assumes *p tagged-division-of i* $\bigwedge u\ v.\ \{u..v\} \neq \{\} \Longrightarrow content\ \{u..v\} = 0 \Longrightarrow d\ \{u..v\} = 0$
shows *setsum* (λ(*x*,*k*). *d k*) *p* = *setsum d* (*snd* ‘ *p*)
proof− **note** *assm*=*tagged-division-ofD*[*OF assms*(*1*)]
have ∗:(λ(*x*,*k*). *d k*) = *d* ∘ *snd* **unfolding** *o-def* **apply**(*rule ext*) **by** *auto*

show *?thesis* **unfolding** $*$ **apply**(*subst eq-commute*) **proof**(*rule setsum-reindex-nonzero*)
show *finite p* **using** *assm* **by** *auto*
fix *x y* **assume** *as*:$x\in p$ $y\in p$ $x\neq y$ *snd x = snd y*
obtain *a b* **where** *ab*:*snd x* = $\{a..b\}$ **using** *assm*(*4*)[*of fst x snd x*] *as*(*1*) **by** *auto*
have (*fst x*, *snd y*) $\in$ *p* (*fst x*, *snd y*) $\neq$ *y* **unfolding** *as*(*4*)[*THEN sym*] **using** *as*(*1*−*3*) **by** *auto*
hence *interior* (*snd x*) $\cap$ *interior* (*snd y*) = {} **apply**−**apply**(*rule assm*(*5*)[*of fst x - fst y*]) **using** *as* **by** *auto*
hence *content* $\{a..b\}$ = *0* **unfolding** *as*(*4*)[*THEN sym*] *ab content-eq-0-interior* **by** *auto*
hence *d* $\{a..b\}$ = *0* **apply**−**apply**(*rule assms*(*2*)) **using** *assm*(*2*)[*of fst x snd x*] *as*(*1*) **unfolding** *ab*[*THEN sym*] **by** *auto*
thus *d* (*snd x*) = *0* **unfolding** *ab* **by** *auto* **qed qed**

lemma *tag-in-interval*: *p tagged-division-of i* $\Longrightarrow$ $(x,k) \in p \Longrightarrow x \in i$ **by** *auto*

lemma *tagged-division-of-empty*: {} *tagged-division-of* {}
unfolding *tagged-division-of* **by** *auto*

lemma *tagged-partial-division-of-trivial*[*simp*]:
p tagged-partial-division-of {} $\longleftrightarrow$ *p* = {}
unfolding *tagged-partial-division-of-def* **by** *auto*

lemma *tagged-division-of-trivial*[*simp*]:
p tagged-division-of {} $\longleftrightarrow$ *p* = {}
unfolding *tagged-division-of* **by** *auto*

lemma *tagged-division-of-self*:
$x \in \{a..b\} \Longrightarrow \{(x,\{a..b\})\}$ *tagged-division-of* $\{a..b\}$
apply(*rule tagged-division-ofI*) **by** *auto*

lemma *tagged-division-union*:
assumes *p1 tagged-division-of s1* *p2 tagged-division-of s2 interior s1* $\cap$ *interior s2* = {}
shows (*p1* $\cup$ *p2*) *tagged-division-of* (*s1* $\cup$ *s2*)
proof(*rule tagged-division-ofI*) **note** *p1*=*tagged-division-ofD*[*OF assms*(*1*)] **and** *p2*=*tagged-division-ofD*[*OF assms*(*2*)]
show *finite* (*p1* $\cup$ *p2*) **using** *p1*(*1*) *p2*(*1*) **by** *auto*
show $\bigcup\{k.\ \exists x.\ (x,\ k) \in p1 \cup p2\}$ = *s1* $\cup$ *s2* **using** *p1*(*6*) *p2*(*6*) **by** *blast*
fix *x k* **assume** *xk*:$(x,k)\in p1\cup p2$ **show** $x\in k$ $\exists a\ b.\ k = \{a..b\}$ **using** *xk p1*(*2*,*4*) *p2*(*2*,*4*) **by** *auto*
show $k\subseteq s1\cup s2$ **using** *xk p1*(*3*) *p2*(*3*) **by** *blast*
fix *x′ k′* **assume** *xk′*:$(x',k')\in p1\cup p2$ $(x,k) \neq (x',k')$
have $*$:$\bigwedge a\ b.\ a\subseteq s1 \Longrightarrow b\subseteq s2 \Longrightarrow$ *interior a* $\cap$ *interior b* = {} **using** *assms*(*3*) *subset-interior* **by** *blast*
show *interior k* $\cap$ *interior k′* = {} **apply**(*cases* $(x,k)\in p1$, *case-tac*[!] $(x',k')\in p1$)
apply(*rule p1*(*5*)) **prefer** *4* **apply**(*rule* $*$) **prefer** *6* **apply**(*subst Int-commute*,*rule* $*$) **prefer** *8* **apply**(*rule p2*(*5*))

using *p1(3) p2(3)* **using** *xk xk'* **by** *auto* **qed**

lemma *tagged-division-unions*:
assumes *finite iset* $\forall i{\in}iset.$ (*pfn*(*i*) *tagged-division-of i*)
$\forall i1 \in iset.\ \forall i2 \in iset.\ {}^\sim(i1 = i2) \longrightarrow (interior(i1) \cap interior(i2) = \{\})$
shows $\bigcup$(*pfn* ' *iset*) *tagged-division-of* ($\bigcup$ *iset*)
proof(*rule tagged-division-ofI*)
note *assm* = *tagged-division-ofD*[*OF assms*(*2*)[*rule-format*]]
show *finite* ($\bigcup$ *pfn* ' *iset*) **apply**(*rule finite-Union*) **using** *assms* **by** *auto*
have $\bigcup\{k.\ \exists x.\ (x,\ k) \in \bigcup pfn\ `\ iset\} = \bigcup(\lambda i.\ \bigcup\{k.\ \exists x.\ (x,\ k) \in pfn\ i\})\ `\ iset$
by *blast*
also have ... = $\bigcup$ *iset* **using** *assm*(*6*) **by** *auto*
finally show $\bigcup\{k.\ \exists x.\ (x,\ k) \in \bigcup pfn\ `\ iset\} = \bigcup iset$ **.**
fix *x k* **assume** *xk*:$(x,k){\in}\bigcup pfn\ `\ iset$ **then obtain** *i* **where** *i*:$i \in iset\ (x,\ k) \in pfn\ i$ **by** *auto*
show $x{\in}k\ \exists a\ b.\ k = \{a..b\}\ k \subseteq \bigcup iset$ **using** *assm*(*2−4*)[*OF i*] **using** *i*(*1*) **by** *auto*
fix $x'\ k'$ **assume** $xk'{:}(x',k'){\in}\bigcup pfn\ `\ iset\ (x,\ k) \neq (x',\ k')$ **then obtain** i' **where** $i'{:}i' \in iset\ (x',\ k') \in pfn\ i'$ **by** *auto*
have $*{:}\bigwedge a\ b.\ i{\neq}i' \Longrightarrow a{\subseteq}\ i \Longrightarrow b{\subseteq}\ i' \Longrightarrow interior\ a \cap interior\ b = \{\}$ **using** *i*(*1*) *i'*(*1*)
using *assms*(*3*)[*rule-format*] *subset-interior* **by** *blast*
show *interior k* $\cap$ *interior k'* = $\{\}$ **apply**(*cases i=i'*)
using *assm*(*5*)[*OF i - xk'*(*2*)] *i'*(*2*) **using** *assm*(*3*)[*OF i*] *assm*(*3*)[*OF i'*] **defer**
apply−apply(*rule* $*$) **by** *auto*
qed

lemma *tagged-partial-division-of-union-self*:
assumes *p tagged-partial-division-of s* **shows** *p tagged-division-of* ($\bigcup$(*snd* ' *p*))
apply(*rule tagged-division-ofI*) **using** *tagged-partial-division-ofD*[*OF assms*] **by** *auto*

lemma *tagged-division-of-union-self*: **assumes** *p tagged-division-of s*
shows *p tagged-division-of* ($\bigcup$(*snd* ' *p*))
apply(*rule tagged-division-ofI*) **using** *tagged-division-ofD*[*OF assms*] **by** *auto*

23.8 Fine-ness of a partition w.r.t. a gauge.

definition *fine* (**infixr** *fine 46*) **where**
d fine s $\longleftrightarrow$ $(\forall (x,k) \in s.\ k \subseteq d(x))$

lemma *fineI*: **assumes** $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow k \subseteq d\ x$
shows *d fine s* **using** *assms* **unfolding** *fine-def* **by** *auto*

lemma *fineD*[*dest*]: **assumes** *d fine s*
shows $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow k \subseteq d\ x$ **using** *assms* **unfolding** *fine-def* **by** *auto*

lemma *fine-inter*: $(\lambda x.\ d1\ x \cap d2\ x)$ *fine p* $\longleftrightarrow$ *d1 fine p* $\wedge$ *d2 fine p*
unfolding *fine-def* **by** *auto*

lemma *fine-inters*:
($\lambda x.\ \bigcap\ \{f\ d\ x \mid d.\ \ d \in s\}$) *fine p* $\longleftrightarrow$ ($\forall\, d{\in}s.$ ($f\ d$) *fine p*)
unfolding *fine-def* **by** *blast*

lemma *fine-union*:
d fine p1 $\Longrightarrow$ *d fine p2* $\Longrightarrow$ *d fine* ($p1 \cup p2$)
unfolding *fine-def* **by** *blast*

lemma *fine-unions*:($\bigwedge p.\ p \in ps \Longrightarrow d$ *fine p*) $\Longrightarrow$ *d fine* ($\bigcup ps$)
unfolding *fine-def* **by** *auto*

lemma *fine-subset*: $p \subseteq q \Longrightarrow d$ *fine q* $\Longrightarrow d$ *fine p*
unfolding *fine-def* **by** *blast*

23.9 Gauge integral. Define on compact intervals first, then use a limit.

definition *has-integral-compact-interval* (**infixr** *has′-integral′-compact′-interval 46*)
where
(*f has-integral-compact-interval y*) $i \equiv$
($\forall\, e{>}0.\ \exists\, d.$ *gauge d* $\wedge$
($\forall\, p.$ *p tagged-division-of i* $\wedge$ *d fine p*
$\longrightarrow$ *norm*(*setsum* ($\lambda(x,k).$ *content k* $*_R$ *f x*) $p - y$) $< e$))

definition *has-integral* (**infixr** *has′-integral 46*) **where**
((f::(*real*^′*n* $\Rightarrow$ ′*b*::*real-normed-vector*)) *has-integral y*) $i \equiv$
if ($\exists\, a\ b.\ i = \{a..b\}$) *then* (*f has-integral-compact-interval y*) *i*
else ($\forall\, e{>}0.\ \exists\, B{>}0.\ \forall\, a\ b.$ *ball 0 B* $\subseteq \{a..b\}$
$\longrightarrow$ ($\exists\, z.$ (($\lambda x.$ *if* $x \in i$ *then f x else 0*) *has-integral-compact-interval z*)
$\{a..b\} \wedge$
norm($z - y$) $< e$))

lemma *has-integral*:
(*f has-integral y*) ($\{a..b\}$) $\longleftrightarrow$
($\forall\, e{>}0.\ \exists\, d.$ *gauge d* $\wedge$ ($\forall\, p.$ *p tagged-division-of* $\{a..b\} \wedge$ *d fine p*
$\longrightarrow$ *norm*(*setsum* ($\lambda(x,k).$ *content*(k) $*_R$ *f x*) $p - y$) $< e$))
unfolding *has-integral-def has-integral-compact-interval-def* **by** *auto*

lemma *has-integralD*[*dest*]: **assumes**
(*f has-integral y*) ($\{a..b\}$) $e{>}0$
obtains *d* **where** *gauge d* $\bigwedge p.$ *p tagged-division-of* $\{a..b\} \Longrightarrow$ *d fine p*
$\Longrightarrow$ *norm*(*setsum* ($\lambda(x,k).$ *content*(k) $*_R$ $f(x)$) $p - y$) $< e$
using *assms* **unfolding** *has-integral* **by** *auto*

lemma *has-integral-alt*:
(*f has-integral y*) $i \longleftrightarrow$
(*if* ($\exists\, a\ b.\ i = \{a..b\}$) *then* (*f has-integral y*) *i*
else ($\forall\, e{>}0.\ \exists\, B{>}0.\ \forall\, a\ b.$ *ball 0 B* $\subseteq \{a..b\}$

$\longrightarrow$ ($\exists\, z.$ (($\lambda x.$ *if* $x \in i$ *then* $f(x)$ *else* 0) *has-integral* z) ($\{a..b\}$) $\wedge$ $norm(z - y) < e$)))
unfolding *has-integral* **unfolding** *has-integral-compact-interval-def has-integral-def* **by** *auto*

lemma *has-integral-altD*:
assumes (f *has-integral* y) i $\neg$ ($\exists\, a\ b.\ i = \{a..b\}$) $e>0$
obtains B **where** $B>0$ $\forall\, a\ b.$ *ball* 0 $B \subseteq \{a..b\} \longrightarrow$ ($\exists\, z.$ (($\lambda x.$ *if* $x \in i$ *then* $f(x)$ *else* 0) *has-integral* z) ($\{a..b\}$) $\wedge$ $norm(z - y) < e$)
using *assms* **unfolding** *has-integral* **unfolding** *has-integral-compact-interval-def has-integral-def* **by** *auto*

definition *integrable-on* (**infixr** *integrable'-on* 46) **where**
(f *integrable-on* i) $\equiv \exists\, y.$ (f *has-integral* y) i

definition *integral* i $f \equiv$ *SOME* $y.$ (f *has-integral* y) i

lemma *integrable-integral*[*dest*]:
f *integrable-on* $i \Longrightarrow$ (f *has-integral* (*integral* i f)) i
unfolding *integrable-on-def integral-def* **by**(*rule someI-ex*)

lemma *has-integral-integrable*[*intro*]: (f *has-integral* i) $s \Longrightarrow$ f *integrable-on* s
unfolding *integrable-on-def* **by** *auto*

lemma *has-integral-integral*:f *integrable-on* $s \longleftrightarrow$ (f *has-integral* (*integral* s f)) s
by *auto*

lemma *has-integral-vec1*: **assumes** (f *has-integral* k) $\{a..b\}$
shows (($\lambda x.$ *vec1* (f x)) *has-integral* (*vec1* k)) $\{a..b\}$
proof− **have** $*$:$\bigwedge p.$ ($\sum$ (x, k)$\in p.$ *content* k $*_R$ *vec1* (f x)) − *vec1* k = *vec1* (($\sum$ (x, k)$\in p.$ *content* k $*_R$ f x) − k)
unfolding *vec-sub Cart-eq* **by**(*auto simp add*: *split-beta*)
show *?thesis* **using** *assms* **unfolding** *has-integral* **apply** *safe*
apply(*erule-tac* x=e **in** *allE*,*safe*) **apply**(*rule-tac* x=d **in** *exI*,*safe*)
apply(*erule-tac* x=p **in** *allE*,*safe*) **unfolding** $*$ *norm-vector-1* **by** *auto* **qed**

lemma *setsum-content-null*:
assumes *content*($\{a..b\}$) = 0 p *tagged-division-of* $\{a..b\}$
shows *setsum* (λ(x,k). *content* k $*_R$ f x) p = (0::'a::*real-normed-vector*)
proof(*rule setsum-0'*,*rule*) **fix** y **assume** y:$y\in p$
obtain x k **where** xk:y = (x,k) **using** *surj-pair*[*of* y] **by** *blast*
note *assm* = *tagged-division-ofD*(3−4)[*OF assms*(2) y[*unfolded* xk]]
from *this*(2) **guess** c **.. then guess** d **.. note** *c-d*=*this*
have (λ(x, k). *content* k $*_R$ f x) y = *content* k $*_R$ f x **unfolding** xk **by** *auto*
also have ... = 0 **using** *content-subset*[*OF assm*(1)[*unfolded c-d*]] *content-pos-le*[*of* c d]
unfolding *assms*(1) *c-d* **by** *auto*
finally show (λ(x, k). *content* k $*_R$ f x) y = 0 **.**

qed

23.10 Some basic combining lemmas.

lemma *tagged-division-unions-exists*:
assumes *finite iset* $\forall i \in iset.\ \exists p.\ p$ *tagged-division-of* $i \wedge d$ *fine* p
$\forall i1 \in iset.\ \forall i2 \in iset.\ \sim(i1 = i2) \longrightarrow (interior(i1) \cap interior(i2) = \{\})$ $(\bigcup iset = i)$
obtains p **where** p *tagged-division-of* i d *fine* p
proof− **guess** *pfn* **using** *bchoice*[*OF assms*(*2*)] **..** **note** *pfn* = *conjunctD2*[*OF this*[*rule-format*]]
show *thesis* **apply**(*rule-tac* p=$\bigcup$(*pfn* ' *iset*) **in** *that*) **unfolding** *assms*(*4*)[*THEN sym*]
apply(*rule tagged-division-unions*[*OF assms*(*1*) - *assms*(*3*)]) **defer**
apply(*rule fine-unions*) **using** *pfn* **by** *auto*
qed

23.11 The set we're concerned with must be closed.

lemma *division-of-closed*: s *division-of* $i \Longrightarrow$ *closed* $(i::(real\hat{}'n)\ set)$
unfolding *division-of-def* **by**(*fastsimp intro*!: *closed-Union closed-interval*)

23.12 General bisection principle for intervals; might be useful elsewhere.

lemma *interval-bisection-step*:
assumes $P\ \{\}$ $(\forall s\ t.\ P\ s \wedge P\ t \wedge interior(s) \cap interior(t) = \{\} \longrightarrow P(s \cup t))$
$\sim(P\ \{a..b::real\hat{}'n\})$
obtains $c\ d$ **where** $\sim(P\{c..d\})$
$\forall i.\ a\$i \le c\$i \wedge c\$i \le d\$i \wedge d\$i \le b\$i \wedge 2 * (d\$i - c\$i) \le b\$i - a\$i$
proof− **have** $\{a..b\} \neq \{\}$ **using** *assms*(*1*,*3*) **by** *auto*
note *ab*=*this*[*unfolded interval-eq-empty not-ex not-less*]
{ fix f **have** *finite* $f \Longrightarrow$
$(\forall s \in f.\ P\ s) \Longrightarrow$
$(\forall s \in f.\ \exists a\ b.\ s = \{a..b\}) \Longrightarrow$
$(\forall s \in f. \forall t \in f.\ \sim(s = t) \longrightarrow interior(s) \cap interior(t) = \{\}) \Longrightarrow P(\bigcup f)$
proof(*induct f rule*:*finite-induct*)
case *empty* **show** *?case* **using** *assms*(*1*) **by** *auto*
next case (*insert x f*) **show** *?case* **unfolding** *Union-insert* **apply**(*rule assms*(*2*)[*rule-format*])
apply *rule* **defer apply** *rule* **defer apply**(*rule inter-interior-unions-intervals*)
using *insert* **by** *auto*
qed } note $* = this$
let *?A* = $\{\{c..d\} \mid c\ d.\ \forall i.\ (c\$i = a\$i) \wedge (d\$i = (a\$i + b\$i)\ /\ 2) \vee (c\$i = (a\$i + b\$i)\ /\ 2) \wedge (d\$i = b\$i)\}$
let *?PP* = $\lambda c\ d.\ \forall i.\ a\$i \le c\$i \wedge c\$i \le d\$i \wedge d\$i \le b\$i \wedge 2 * (d\$i - c\$i) \le b\$i - a\$i$
{ presume $\forall c\ d.$ *?PP c d* $\longrightarrow P\ \{c..d\} \Longrightarrow$ *False*
thus *thesis* **unfolding** *atomize-not not-all* **apply**−**apply**(*erule exE*)+ **apply**(*rule-tac* c=x **and** d=xa **in** *that*) **by** *auto* **}**
assume *as*:$\forall c\ d.$ *?PP c d* $\longrightarrow P\ \{c..d\}$

have P ($\bigcup$ *?A*) **proof**(*rule* $*$, *rule-tac*[*2*−] *ballI*, *rule-tac*[*4*] *ballI*, *rule-tac*[*4*] *impI*)
let *?B* = ($\lambda s.\{(\chi\ i.\ if\ i \in s\ then\ a\$i\ else\ (a\$i + b\$i)\ /\ 2)\ ..$
$(\chi\ i.\ if\ i \in s\ then\ (a\$i + b\$i)\ /\ 2\ else\ b\$i)\}$) ‘ $\{s.\ s \subseteq UNIV\}$
have *?A* $\subseteq$ *?B* **proof case** *goal1*
then guess c **unfolding** *mem-Collect-eq* **.. then guess** d **apply**− **by**(*erule exE*,(*erule conjE*)+) **note** *c-d=this*[*rule-format*]
have $*$:$\bigwedge a\ b\ c\ d.\ a = c \Longrightarrow b = d \Longrightarrow \{a..b\} = \{c..d\}$ **by** *auto*
show *x*∈*?B* **unfolding** *image-iff* **apply**(*rule-tac x*={*i. c*$*i* = *a*$*i*} **in** *bexI*)
unfolding *c-d* **apply**(*rule* $*$) **unfolding** *Cart-eq cond-component Cart-lambda-beta*
proof(*rule-tac*[*1*−*2*] *allI*) **fix** i **show** $c\ \$\ i = (if\ i \in \{i.\ c\ \$\ i = a\ \$\ i\}\ then\ a\ \$\ i\ else\ (a\ \$\ i + b\ \$\ i)\ /\ 2)$
$d\ \$\ i = (if\ i \in \{i.\ c\ \$\ i = a\ \$\ i\}\ then\ (a\ \$\ i + b\ \$\ i)\ /\ 2\ else\ b\ \$\ i)$
using *c-d*(*2*)[*of i*] *ab*[*THEN spec*[**where** *x*=*i*]] **by**(*auto simp add*:*field-simps*)
qed *auto* **qed**
thus *finite ?A* **apply**(*rule finite-subset*[*of - ?B*]) **by** *auto*
fix s **assume** *s*∈*?A* **then guess** c **unfolding** *mem-Collect-eq* **.. then guess** d **apply**− **by**(*erule exE*,(*erule conjE*)+)
note *c-d=this*[*rule-format*]
show $P\ s$ **unfolding** *c-d* **apply**(*rule as*[*rule-format*]) **proof**− **case** *goal1* **show** *?case*
using *c-d*(*2*)[*of i*] **using** *ab*[*THEN spec*[**where** *x*=*i*]] **by** *auto* **qed**
show $\exists\ a\ b.\ s = \{a..b\}$ **unfolding** *c-d* **by** *auto*
fix t **assume** *t*∈*?A* **then guess** e **unfolding** *mem-Collect-eq* **.. then guess** f **apply**− **by**(*erule exE*,(*erule conjE*)+)
note *e-f=this*[*rule-format*]
assume $s \neq t$ **hence** ¬ $(c = e \wedge d = f)$ **unfolding** *c-d e-f* **by** *auto*
then obtain i **where** $c\$i \neq e\$i \vee d\$i \neq f\$i$ **unfolding** *de-Morgan-conj Cart-eq* **by** *auto*
hence *i*:$c\$i \neq e\$i\ d\$i \neq f\$i$ **apply**− **apply**(*erule-tac*[!] *disjE*)
proof− **assume** $c\$i \neq e\$i$ **thus** $d\$i \neq f\$i$ **using** *c-d*(*2*)[*of i*] *e-f*(*2*)[*of i*] **by** *fastsimp*
next **assume** $d\$i \neq f\$i$ **thus** $c\$i \neq e\$i$ **using** *c-d*(*2*)[*of i*] *e-f*(*2*)[*of i*] **by** *fastsimp*
qed have $*$:$\bigwedge s\ t.\ (\bigwedge a.\ a \in s \Longrightarrow a \in t \Longrightarrow False) \Longrightarrow s \cap t = \{\}$ **by** *auto*
show *interior s* ∩ *interior t* = {} **unfolding** *e-f c-d interior-closed-interval* **proof**(*rule* $*$)
fix x **assume** *x*∈{*c*<..<*d*} *x*∈{*e*<..<*f*}
hence *x*:$c\$i < d\$i\ e\$i < f\$i\ c\$i < f\$i\ e\$i < d\$i$ **unfolding** *mem-interval* **apply**−**apply**(*erule-tac*[!] *x*=*i* **in** *allE*)+ **by** *auto*
show *False* **using** *c-d*(*2*)[*of i*] **apply**− **apply**(*erule-tac disjE*)
proof(*erule-tac*[!] *conjE*) **assume** *as*:$c\ \$\ i = a\ \$\ i\ d\ \$\ i = (a\ \$\ i + b\ \$\ i)\ /\ 2$
show *False* **using** *e-f*(*2*)[*of i*] **and** *i x* **unfolding** *as* **by**(*fastsimp simp add*:*field-simps*)
next assume *as*:$c\ \$\ i = (a\ \$\ i + b\ \$\ i)\ /\ 2\ d\ \$\ i = b\ \$\ i$
show *False* **using** *e-f*(*2*)[*of i*] **and** *i x* **unfolding** *as* **by**(*fastsimp simp add*:*field-simps*)
qed qed qed

also have $\bigcup$ *?A* = {*a..b*} **proof**(*rule set-ext*,*rule*)
fix *x* **assume** *x*∈$\bigcup$ *?A* **then guess** *Y* **unfolding** *Union-iff* **..**
from *this*(*1*) **guess** *c* **unfolding** *mem-Collect-eq* **.. then guess** *d* **..**
note *c-d* = *this*[*THEN conjunct2*,*rule-format*] ‹*x*∈*Y*›[*unfolded this*[*THEN conjunct1*]]
show *x*∈{*a..b*} **unfolding** *mem-interval* **proof**
fix *i* **show** *a* $ *i* ≤ *x* $ *i* ∧ *x* $ *i* ≤ *b* $ *i*
using *c-d*(*1*)[*of i*] *c-d*(*2*)[*unfolded mem-interval*,*THEN spec*[**where** *x*=*i*]]
by *auto* **qed**
next fix *x* **assume** *x*:*x*∈{*a..b*}
have ∀ *i*. ∃ *c d*. (*c* = *a*$*i* ∧ *d* = (*a*$*i* + *b*$*i*) / *2* ∨ *c* = (*a*$*i* + *b*$*i*) / *2* ∧ *d* = *b*$*i*) ∧ *c*≤*x*$*i* ∧ *x*$*i* ≤ *d*
(**is** ∀ *i*. ∃ *c d*. *?P i c d*) **unfolding** *mem-interval* **proof fix** *i*
have *?P i* (*a*$*i*) ((*a* $ *i* + *b* $ *i*) / *2*) ∨ *?P i* ((*a* $ *i* + *b* $ *i*) / *2*) (*b*$*i*)
using *x*[*unfolded mem-interval*,*THEN spec*[**where** *x*=*i*]] **by** *auto* **thus** ∃ *c d*. *?P i c d* **by** *blast*
qed thus *x*∈$\bigcup$ *?A* **unfolding** *Union-iff lambda-skolem* **unfolding** *Bex-def mem-Collect-eq*
apply−**apply**(*erule exE*)+ **apply**(*rule-tac x*={*xa..xaa*} **in** *exI*) **unfolding** *mem-interval* **by** *auto*
qed finally show *False* **using** *assms* **by** *auto* **qed**

lemma *interval-bisection*:
assumes *P* {} (∀ *s t*. *P s* ∧ *P t* ∧ *interior*(*s*) ∩ *interior*(*t*) = {} ⟶ *P*(*s* ∪ *t*)) ¬ *P* {*a..b*::*real*^′*n*}
obtains *x* **where** *x* ∈ {*a..b*} ∀ *e*>*0*. ∃ *c d*. *x* ∈ {*c..d*} ∧ {*c..d*} ⊆ *ball x e* ∧ {*c..d*} ⊆ {*a..b*} ∧ ~*P*({*c..d*})
proof−
have ∀ *x*. ∃ *y*. ¬ *P* {*fst x..snd x*} ⟶ (¬ *P* {*fst y..snd y*} ∧ (∀ *i*. *fst x*$*i* ≤ *fst y*$*i* ∧ *fst y*$*i* ≤ *snd y*$*i* ∧ *snd y*$*i* ≤ *snd x*$*i* ∧
2 ∗ (*snd y*$*i* − *fst y*$*i*) ≤ *snd x*$*i* − *fst x*$*i*)) **proof case** *goal1* **thus** *?case* **proof**−
presume ¬ *P* {*fst x..snd x*} ⟹ *?thesis*
thus *?thesis* **apply**(*cases P* {*fst x..snd x*}) **by** *auto*
next assume *as*:¬ *P* {*fst x..snd x*} **from** *interval-bisection-step*[*of P*, *OF assms*(*1*−*2*) *as*] **guess** *c d* **.**
thus *?thesis* **apply**− **apply**(*rule-tac x*=(*c*,*d*) **in** *exI*) **by** *auto*
qed qed then guess *f* **apply**−**apply**(*drule choice*) **by**(*erule exE*) **note** *f*=*this*
def *AB* ≡ λ*n*. (*f* ^^ *n*) (*a*,*b*) **def** *A* ≡ λ*n*. *fst*(*AB n*) **and** *B* ≡ λ*n*. *snd*(*AB n*)
note *ab-def* = *this AB-def*
have *A 0* = *a B 0* = *b* ⋀*n*. ¬ *P* {*A*(*Suc n*)..*B*(*Suc n*)} ∧
(∀ *i*. *A*(*n*)$*i* ≤ *A*(*Suc n*)$*i* ∧ *A*(*Suc n*)$*i* ≤ *B*(*Suc n*)$*i* ∧ *B*(*Suc n*)$*i* ≤ *B*(*n*)$*i* ∧
2 ∗ (*B*(*Suc n*)$*i* − *A*(*Suc n*)$*i*) ≤ *B*(*n*)$*i* − *A*(*n*)$*i*) (**is** ⋀*n*. *?P n*)
proof− **show** *A 0* = *a B 0* = *b* **unfolding** *ab-def* **by** *auto*
case *goal3* **note** *S* = *ab-def funpow.simps o-def id-apply* **show** *?case*
proof(*induct n*) **case** *0* **thus** *?case* **unfolding** *S* **apply**(*rule f*[*rule-format*]) **using** *assms*(*3*) **by** *auto*
next case (*Suc n*) **show** *?case* **unfolding** *S* **apply**(*rule f*[*rule-format*]) **using**

Suc **unfolding** *S* **by** *auto*
qed qed note *AB* = *this*(*1−2*) *conjunctD2*[*OF this*(*3*),*rule-format*]

have *interv*:⋀*e*. *0* < *e* ⟹ ∃ *n*. ∀ *x*∈{*A n..B n*}. ∀ *y*∈{*A n..B n*}. *dist x y* < *e*
proof− **case** *goal1* **guess** *n* **using** *real-arch-pow2*[*of* (*setsum* (λ*i*. *b*$*i* − *a*$*i*) *UNIV*) / *e*] **.. note** *n*=*this*
show *?case* **apply**(*rule-tac x*=*n* **in** *exI*) **proof**(*rule*,*rule*)
fix *x y* **assume** *xy*:*x*∈{*A n..B n*} *y*∈{*A n..B n*}
have *dist x y* ≤ *setsum* (λ*i*. *abs*((*x* − *y*)$*i*)) *UNIV* **unfolding** *dist-norm* **by**(*rule norm-le-l1*)
also have ... ≤ *setsum* (λ*i*. *B n*$*i* − *A n*$*i*) *UNIV*
proof(*rule setsum-mono*) **fix** *i* **show** |(*x* − *y*) $ *i*| ≤ *B n* $ *i* − *A n* $ *i*
using *xy*[*unfolded mem-interval*,*THEN spec*[**where** *x*=*i*]]
unfolding *vector-minus-component* **by** *auto* **qed**
also have ... ≤ *setsum* (λ*i*. *b*$*i* − *a*$*i*) *UNIV* / *2*^*n* **unfolding** *setsum-divide-distrib*
proof(*rule setsum-mono*) **case** *goal1* **thus** *?case*
proof(*induct n*) **case** *0* **thus** *?case* **unfolding** *AB* **by** *auto*
next case (*Suc n*) **have** *B* (*Suc n*) $ *i* − *A* (*Suc n*) $ *i* ≤ (*B n* $ *i* − *A n* $ *i*) / *2* **using** *AB*(*4*)[*of n i*] **by** *auto*
also have ... ≤ (*b* $ *i* − *a* $ *i*) / *2* ^ *Suc n* **using** *Suc* **by**(*auto simp add*:*field-simps*) **finally show** *?case* **.**
qed qed
also have ... < *e* **using** *n* **using** *goal1* **by**(*auto simp add*:*field-simps*) **finally show** *dist x y* < *e* **.**
qed qed
{ fix *n m* ::*nat* **assume** *m* ≤ *n* **then guess** *d* **unfolding** *le-Suc-ex-iff* **.. note** *d*=*this*
have {*A n..B n*} ⊆ {*A m..B m*} **unfolding** *d*
proof(*induct d*) **case** *0* **thus** *?case* **by** *auto*
next case (*Suc d*) **show** *?case* **apply**(*rule subset-trans*[*OF - Suc*])
apply(*rule*) **unfolding** *mem-interval* **apply**(*rule*,*erule-tac x*=*i* **in** *allE*)
proof− **case** *goal1* **thus** *?case* **using** *AB*(*4*)[*of m* + *d i*] **by**(*auto simp add*:*field-simps*)
qed qed } note *ABsubset* = *this*
have ∃ *a*. ∀ *n*. *a*∈{*A n..B n*} **apply**(*rule decreasing-closed-nest*[*rule-format*,*OF closed-interval - ABsubset interv*])
proof− **fix** *n* **show** {*A n..B n*} ≠ {} **apply**(*cases 0*<*n*) **using** *AB*(*3*)[*of n* − *1*] *assms*(*1*,*3*) *AB*(*1−2*) **by** *auto* **qed** *auto*
then guess *x0* **.. note** *x0*=*this*[*rule-format*]
show *thesis* **proof**(*rule that*[*rule-format*,*of x0*])
show *x0*∈{*a..b*} **using** *x0*[*of 0*] **unfolding** *AB* **.**
fix *e* **assume** *0* < (*e*::*real*) **from** *interv*[*OF this*] **guess** *n* **.. note** *n*=*this*
show ∃ *c d*. *x0* ∈ {*c..d*} ∧ {*c..d*} ⊆ *ball x0 e* ∧ {*c..d*} ⊆ {*a..b*} ∧ ¬ *P* {*c..d*}
apply(*rule-tac x*=*A n* **in** *exI*,*rule-tac x*=*B n* **in** *exI*) **apply**(*rule*,*rule x0*) **apply** *rule* **defer**
proof show ¬ *P* {*A n..B n*} **apply**(*cases 0*<*n*) **using** *AB*(*3*)[*of n* − *1*] *assms*(*3*) *AB*(*1−2*) **by** *auto*
show {*A n..B n*} ⊆ *ball x0 e* **using** *n* **using** *x0*[*of n*] **by** *auto*
show {*A n..B n*} ⊆ {*a..b*} **unfolding** *AB*(*1−2*)[*symmetric*] **apply**(*rule*

ABsubset) **by** *auto*
qed qed qed

23.13 Cousin's lemma.

lemma *fine-division-exists*: **assumes** *gauge g*
obtains *p* **where** *p tagged-division-of* {*a*..*b*::*real*^'*n*} *g fine p*
proof− **presume** ¬ (∃*p*. *p tagged-division-of* {*a*..*b*} ∧ *g fine p*) ⟹ *False*
then guess *p* **unfolding** *atomize-not not-not* **..** **thus** *thesis* **apply**−**apply**(*rule that*[*of p*]) **by** *auto*
next assume *as*:¬ (∃*p*. *p tagged-division-of* {*a*..*b*} ∧ *g fine p*)
guess *x* **apply**(*rule interval-bisection*[*of* λ*s*. ∃*p*. *p tagged-division-of s* ∧ *g fine p*,*rule-format*,*OF - - as*])
apply(*rule-tac x*={} **in** *exI*) **defer apply**(*erule conjE exE*)+
proof− **show** {} *tagged-division-of* {} ∧ *g fine* {} **unfolding** *fine-def* **by** *auto*
fix *s t p p'* **assume** *p tagged-division-of s g fine p p' tagged-division-of t g fine p' interior s* ∩ *interior t* = {}
thus ∃*p*. *p tagged-division-of s* ∪ *t* ∧ *g fine p* **apply**−**apply**(*rule-tac x*=*p* ∪ *p'* **in** *exI*) **apply** *rule*
apply(*rule tagged-division-union*) **prefer** *4* **apply**(*rule fine-union*) **by** *auto*
qed note *x*=*this*
obtain *e* **where** *e*:*e*>*0 ball x e* ⊆ *g x* **using** *gaugeD*[*OF assms*, *of x*] **unfolding** *open-contains-ball* **by** *auto*
from *x*(*2*)[*OF e*(*1*)] **guess** *c d* **apply**−**apply**(*erule exE conjE*)+ **. note** *c-d* = *this*
have *g fine* {(*x*, {*c*..*d*})} **unfolding** *fine-def* **using** *e* **using** *c-d*(*2*) **by** *auto*
thus *False* **using** *tagged-division-of-self*[*OF c-d*(*1*)] **using** *c-d* **by** *auto* **qed**

23.14 Basic theorems about integrals.

lemma *has-integral-unique*: **fixes** *f*::*real*^'*n* ⇒ '*a*::*real-normed-vector*
assumes (*f has-integral k1*) *i* (*f has-integral k2*) *i* **shows** *k1* = *k2*
proof(*rule ccontr*) **let** *?e* = *norm*(*k1* − *k2*) / *2* **assume** *as*:*k1* ≠ *k2* **hence** *e*:*?e* > *0* **by** *auto*
have *lem*:⋀*f*::*real*^'*n* ⇒ '*a*. ⋀ *a b k1 k2*.
(*f has-integral k1*) ({*a*..*b*}) ⟹ (*f has-integral k2*) ({*a*..*b*}) ⟹ *k1* ≠ *k2* ⟹ *False*
proof− **case** *goal1* **let** *?e* = *norm*(*k1* − *k2*) / *2* **from** *goal1*(*3*) **have** *e*:*?e* > *0* **by** *auto*
guess *d1* **by**(*rule has-integralD*[*OF goal1*(*1*) *e*]) **note** *d1*=*this*
guess *d2* **by**(*rule has-integralD*[*OF goal1*(*2*) *e*]) **note** *d2*=*this*
guess *p* **by**(*rule fine-division-exists*[*OF gauge-inter*[*OF d1*(*1*) *d2*(*1*)],*of a b*]) **note** *p*=*this*
let *?c* = (∑(*x*, *k*)∈*p*. *content k* ∗*R* *f x*) **have** *norm* (*k1* − *k2*) ≤ *norm* (*?c* − *k2*) + *norm* (*?c* − *k1*)
using *norm-triangle-ineq4*[*of k1* − *?c k2* − *?c*] **by**(*auto simp add*:*algebra-simps norm-minus-commute*)
also have ... < *norm* (*k1* − *k2*) / *2* + *norm* (*k1* − *k2*) / *2*
apply(*rule add-strict-mono*) **apply**(*rule-tac*[!] *d2*(*2*) *d1*(*2*)) **using** *p* **unfolding** *fine-def* **by** *auto*

finally show *False* **by** *auto*
qed { presume ¬ (∃ *a b*. *i* = {*a*..*b*}) ⟹ *False*
thus *False* **apply**−**apply**(*cases* ∃ *a b*. *i* = {*a*..*b*})
using *assms* **by**(*auto simp add*:*has-integral intro*:*lem*[*OF - - as*]) **}**
assume *as*:¬ (∃ *a b*. *i* = {*a*..*b*})
guess *B1* **by**(*rule has-integral-altD*[*OF assms*(*1*) *as*,*OF e*]) **note** *B1*=*this*[*rule-format*]
guess *B2* **by**(*rule has-integral-altD*[*OF assms*(*2*) *as*,*OF e*]) **note** *B2*=*this*[*rule-format*]
have ∃ *a b*::*real*^'*n*. *ball 0 B1* ∪ *ball 0 B2* ⊆ {*a*..*b*} **apply**(*rule bounded-subset-closed-interval*)
using *bounded-Un bounded-ball* **by** *auto* **then guess** *a b* **apply**−**by**(*erule exE*)+
note *ab*=*conjunctD2*[*OF this*[*unfolded Un-subset-iff*]]
guess *w* **using** *B1*(*2*)[*OF ab*(*1*)] **.. note** *w*=*conjunctD2*[*OF this*]
guess *z* **using** *B2*(*2*)[*OF ab*(*2*)] **.. note** *z*=*conjunctD2*[*OF this*]
have *z* = *w* **using** *lem*[*OF w*(*1*) *z*(*1*)] **by** *auto*
hence *norm* (*k1* − *k2*) ≤ *norm* (*z* − *k2*) + *norm* (*w* − *k1*)
using *norm-triangle-ineq4* [*of k1* − *w k2* − *z*] **by**(*auto simp add*: *norm-minus-commute*)

also have ... < *norm* (*k1* − *k2*) / *2* + *norm* (*k1* − *k2*) / *2* **apply**(*rule add-strict-mono*) **by**(*rule-tac*[!] *z*(*2*) *w*(*2*))
finally show *False* **by** *auto* **qed**

lemma *integral-unique*[*intro*]:
(*f has-integral y*) *k* ⟹ *integral k f* = *y*
unfolding *integral-def* **apply**(*rule some-equality*) **by**(*auto intro*: *has-integral-unique*)

lemma *has-integral-is-0*: **fixes** *f*::*real*^'*n* ⇒ '*a*::*real-normed-vector*
assumes ∀ *x*∈*s*. *f x* = *0* **shows** (*f has-integral 0*) *s*
proof− **have** *lem*:⋀*a b*. ⋀*f*::*real*^'*n* ⇒ '*a*.
(∀ *x*∈{*a*..*b*}. *f*(*x*) = *0*) ⟹ (*f has-integral 0*) ({*a*..*b*}) **unfolding** *has-integral*
proof(*rule*,*rule*) **fix** *a b e* **and** *f*::*real*^'*n* ⇒ '*a*
assume *as*:∀ *x*∈{*a*..*b*}. *f x* = *0* *0* < (*e*::*real*)
show ∃ *d*. *gauge d* ∧ (∀ *p*. *p tagged-division-of* {*a*..*b*} ∧ *d fine p* ⟶ *norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗$_R$ *f x*) − *0*) < *e*)
apply(*rule-tac x*=λ*x*. *ball x 1* **in** *exI*) **apply**(*rule*,*rule gaugeI*) **unfolding** *centre-in-ball* **defer apply**(*rule open-ball*)
proof(*rule*,*rule*,*erule conjE*) **case** *goal1*
have (∑ (*x*, *k*)∈*p*. *content k* ∗$_R$ *f x*) = *0* **proof**(*rule setsum-0'*,*rule*)
fix *x* **assume** *x*:*x*∈*p* **have** *f* (*fst x*) = *0* **using** *tagged-division-ofD*(*2*−*3*)[*OF goal1*(*1*), *of fst x snd x*] **using** *as x* **by** *auto*
thus (λ(*x*, *k*). *content k* ∗$_R$ *f x*) *x* = *0* **apply**(*subst surjective-pairing*[*of x*]) **unfolding** *split-conv* **by** *auto*
qed thus *?case* **using** *as* **by** *auto*
qed *auto* **qed** **{ presume** ¬ (∃ *a b*. *s* = {*a*..*b*}) ⟹ *?thesis*
thus *?thesis* **apply**−**apply**(*cases* ∃ *a b*. *s* = {*a*..*b*})
using *assms* **by**(*auto simp add*:*has-integral intro*:*lem*) **}**
have ∗:(λ*x*. *if x* ∈ *s then f x else 0*) = (λ*x*. *0*) **apply**(*rule ext*) **using** *assms* **by** *auto*
assume ¬ (∃ *a b*. *s* = {*a*..*b*}) **thus** *?thesis* **apply**(*subst has-integral-alt*) **unfold-**

ing *if-not-P* ∗
apply(*rule*,*rule*,*rule-tac x=1* **in** *exI*,*rule*) **defer apply**(*rule*,*rule*,*rule*)
proof− **fix** *e*::*real* **and** *a b* **assume** *e>0*
thus ∃ *z*. ((λ*x*::*real*^′*n*. *0*::′*a*) *has-integral z*) {*a*..*b*} ∧ *norm* (*z* − *0*) < *e*
apply(*rule-tac x=0* **in** *exI*) **apply**(*rule*,*rule lem*) **by** *auto*
qed *auto* **qed**

lemma *has-integral-0*[*simp*]: ((λ*x*::*real*^′*n*. *0*) *has-integral 0*) *s*
apply(*rule has-integral-is-0*) **by** *auto*

lemma *has-integral-0-eq*[*simp*]: ((λ*x*. *0*) *has-integral i*) *s* ⟷ *i* = *0*
using *has-integral-unique*[*OF has-integral-0*] **by** *auto*

lemma *has-integral-linear*: **fixes** *f*::*real*^′*n* ⇒ ′*a*::*real-normed-vector*
assumes (*f has-integral y*) *s bounded-linear h* **shows** ((*h o f*) *has-integral* ((*h y*))) *s*
proof− **interpret** *bounded-linear h* **using** *assms*(*2*) **. from** *pos-bounded* **guess** *B* **.. note** *B=conjunctD2*[*OF this*,*rule-format*]
have *lem*:⋀*f*::*real*^′*n* ⇒ ′*a*. ⋀ *y a b*.
(*f has-integral y*) ({*a*..*b*}) ⟹ ((*h o f*) *has-integral h*(*y*)) ({*a*..*b*})
proof(*subst has-integral*,*rule*,*rule*) **case** *goal1*
from *pos-bounded* **guess** *B* **.. note** *B=conjunctD2*[*OF this*,*rule-format*]
have ∗:*e* / *B* > *0* **apply**(*rule divide-pos-pos*) **using** *goal1*(*2*) *B* **by** *auto*
guess *g* **using** *has-integralD*[*OF goal1*(*1*) ∗] **. note** *g=this*
show *?case* **apply**(*rule-tac x=g* **in** *exI*) **apply**(*rule*,*rule g*(*1*))
proof(*rule*,*rule*,*erule conjE*) **fix** *p* **assume** *as*:*p tagged-division-of* {*a*..*b*} *g fine p*
have ∗:⋀*x k*. *h* ((λ(*x*, *k*). *content k* ∗*R f x*) *x*) = (λ(*x*, *k*). *h* (*content k* ∗*R f x*)) *x* **by** *auto*
have (∑ (*x*, *k*)∈*p*. *content k* ∗*R* (*h* ∘ *f*) *x*) = *setsum* (*h* ∘ (λ(*x*, *k*). *content k* ∗*R f x*)) *p*
unfolding *o-def* **unfolding** *scaleR*[*THEN sym*] ∗ **by** *simp*
also have . . . = *h* (∑ (*x*, *k*)∈*p*. *content k* ∗*R f x*) **using** *setsum*[*of* λ(*x*,*k*). *content k* ∗*R f x p*] **using** *as* **by** *auto*
finally have ∗:(∑ (*x*, *k*)∈*p*. *content k* ∗*R* (*h* ∘ *f*) *x*) = *h* (∑ (*x*, *k*)∈*p*. *content k* ∗*R f x*) **.**
show *norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗*R* (*h* ∘ *f*) *x*) − *h y*) < *e* **unfolding** ∗ *diff*[*THEN sym*]
apply(*rule le-less-trans*[*OF B*(*2*)]) **using** *g*(*2*)[*OF as*] *B*(*1*) **by**(*auto simp add*:*field-simps*)
qed qed { **presume** ¬ (∃ *a b*. *s* = {*a*..*b*}) ⟹ *?thesis*
thus *?thesis* **apply**−**apply**(*cases* ∃ *a b*. *s* = {*a*..*b*}) **using** *assms* **by**(*auto simp add*:*has-integral intro*!:*lem*) }
assume *as*:¬ (∃ *a b*. *s* = {*a*..*b*}) **thus** *?thesis* **apply**(*subst has-integral-alt*) **unfolding** *if-not-P*
proof(*rule*,*rule*) **fix** *e*::*real* **assume** *e*:*0*<*e*
have ∗:*0* < *e*/*B* **by**(*rule divide-pos-pos*,*rule e*,*rule B*(*1*))
guess *M* **using** *has-integral-altD*[*OF assms*(*1*) *as* ∗,*rule-format*] **. note** *M=this*
show ∃ *B*>*0*. ∀ *a b*. *ball 0 B* ⊆ {*a*..*b*} ⟶ (∃ *z*. ((λ*x*. *if x* ∈ *s then* (*h* ∘ *f*) *x*

else 0) *has-integral z*) {*a..b*} ∧ *norm* (*z* − *h y*) < *e*)
apply(*rule-tac x=M* **in** *exI*) **apply**(*rule*,*rule M*(*1*))
proof(*rule*,*rule*,*rule*) **case** *goal1* **guess** *z* **using** *M*(*2*)[*OF goal1*(*1*)] **.. note** *z=conjunctD2*[*OF this*]
have ∗:(λ*x. if x* ∈ *s then* (*h* ∘ *f*) *x else 0*) = *h* ∘ (λ*x. if x* ∈ *s then f x else 0*)
unfolding *o-def* **apply**(*rule ext*) **using** *zero* **by** *auto*
show *?case* **apply**(*rule-tac x=h z* **in** *exI*,*rule*) **unfolding** ∗ **apply**(*rule lem*[*OF z*(*1*)]) **unfolding** *diff*[*THEN sym*]
apply(*rule le-less-trans*[*OF B*(*2*)]) **using** *B*(*1*) *z*(*2*) **by**(*auto simp add*:*field-simps*)
qed qed qed

lemma *has-integral-cmul*:
shows (*f has-integral k*) *s* ⟹ ((λ*x. c* ∗*R f x*) *has-integral* (*c* ∗*R k*)) *s*
unfolding *o-def*[*THEN sym*] **apply**(*rule has-integral-linear*,*assumption*)
by(*rule scaleR.bounded-linear-right*)

lemma *has-integral-neg*:
shows (*f has-integral k*) *s* ⟹ ((λ*x.* −(*f x*)) *has-integral* (−*k*)) *s*
apply(*drule-tac c=−1* **in** *has-integral-cmul*) **by** *auto*

lemma *has-integral-add*: **fixes** *f*::*real*^*'n* ⇒ *'a*::*real-normed-vector*
assumes (*f has-integral k*) *s* (*g has-integral l*) *s*
shows ((λ*x. f x* + *g x*) *has-integral* (*k* + *l*)) *s*
proof− **have** *lem*:⋀*f g*::*real*^*'n* ⇒ *'a*. ⋀*a b k l*.
(*f has-integral k*) ({*a..b*}) ⟹ (*g has-integral l*) ({*a..b*}) ⟹
((λ*x. f*(*x*) + *g*(*x*)) *has-integral* (*k* + *l*)) ({*a..b*}) **proof**− **case** *goal1*
show *?case* **unfolding** *has-integral* **proof**(*rule*,*rule*) **fix** *e*::*real* **assume** *e*:*e>0*
hence ∗:*e/2>0* **by** *auto*
guess *d1* **using** *has-integralD*[*OF goal1*(*1*) ∗] **. note** *d1=this*
guess *d2* **using** *has-integralD*[*OF goal1*(*2*) ∗] **. note** *d2=this*
show ∃ *d. gauge d* ∧ (∀ *p. p tagged-division-of* {*a..b*} ∧ *d fine p* ⟶ *norm* ((∑ (*x*, *k*)∈*p. content k* ∗*R* (*f x* + *g x*)) − (*k* + *l*)) < *e*)
apply(*rule-tac x=*λ*x.* (*d1 x*) ∩ (*d2 x*) **in** *exI*) **apply**(*rule*,*rule gauge-inter*[*OF d1*(*1*) *d2*(*1*)])
proof(*rule*,*rule*,*erule conjE*) **fix** *p* **assume** *as*:*p tagged-division-of* {*a..b*} (λ*x. d1 x* ∩ *d2 x*) *fine p*
have ∗:(∑ (*x*, *k*)∈*p. content k* ∗*R* (*f x* + *g x*)) = (∑ (*x*, *k*)∈*p. content k* ∗*R f x*) + (∑ (*x*, *k*)∈*p. content k* ∗*R g x*)
unfolding *scaleR-right-distrib setsum-addf*[*of* λ(*x*,*k*). *content k* ∗*R f x* λ(*x*,*k*). *content k* ∗*R g x p*,*THEN sym*]
by(*rule setsum-cong2*,*auto*)
have *norm* ((∑ (*x*, *k*)∈*p. content k* ∗*R* (*f x* + *g x*)) − (*k* + *l*)) = *norm* (((∑ (*x*, *k*)∈*p. content k* ∗*R f x*) − *k*) + ((∑ (*x*, *k*)∈*p. content k* ∗*R g x*) − *l*))
unfolding ∗ **by**(*auto simp add*:*algebra-simps*) **also let** *?res* = ...
from *as* **have** ∗:*d1 fine p d2 fine p* **unfolding** *fine-inter* **by** *auto*
have *?res* < *e/2* + *e/2* **apply**(*rule le-less-trans*[*OF norm-triangle-ineq*])
apply(*rule add-strict-mono*) **using** *d1*(*2*)[*OF as*(*1*) ∗(*1*)] **and** *d2*(*2*)[*OF as*(*1*) ∗(*2*)] **by** *auto*
finally show *norm* ((∑ (*x*, *k*)∈*p. content k* ∗*R* (*f x* + *g x*)) − (*k* + *l*)) <

e **by** *auto*
qed qed qed { **presume** ¬ (∃ *a b*. *s* = {*a*..*b*}) ⟹ *?thesis*
thus *?thesis* **apply**−**apply**(*cases* ∃ *a b*. *s* = {*a*..*b*}) **using** *assms* **by**(*auto simp add*:*has-integral intro*!:*lem*) }
assume *as*:¬ (∃ *a b*. *s* = {*a*..*b*}) **thus** *?thesis* **apply**(*subst has-integral-alt*) **unfolding** *if-not-P*
proof(*rule*,*rule*) **case** *goal1* **hence** ∗:*e*/2 > 0 **by** *auto*
from *has-integral-altD*[*OF assms*(1) *as* ∗] **guess** *B1* **. note** *B1*=*this*[*rule-format*]
from *has-integral-altD*[*OF assms*(2) *as* ∗] **guess** *B2* **. note** *B2*=*this*[*rule-format*]
show *?case* **apply**(*rule-tac x*=*max B1 B2* **in** *exI*) **apply**(*rule*,*rule min-max.less-supI1*,*rule B1*)
proof(*rule*,*rule*,*rule*) **fix** *a b* **assume** *ball 0* (*max B1 B2*) ⊆ {*a*..*b*::*real*^′*n*}
hence ∗:*ball 0 B1* ⊆ {*a*..*b*::*real*^′*n*} *ball 0 B2* ⊆ {*a*..*b*::*real*^′*n*} **by** *auto*
guess *w* **using** *B1*(2)[*OF* ∗(1)] **.. note** *w*=*conjunctD2*[*OF this*]
guess *z* **using** *B2*(2)[*OF* ∗(2)] **.. note** *z*=*conjunctD2*[*OF this*]
have ∗:⋀*x*. (*if x* ∈ *s then f x* + *g x else* 0) = (*if x* ∈ *s then f x else* 0) + (*if x* ∈ *s then g x else* 0) **by** *auto*
show ∃ *z*. ((λ*x*. *if x* ∈ *s then f x* + *g x else* 0) *has-integral z*) {*a*..*b*} ∧ *norm* (*z* − (*k* + *l*)) < *e*
apply(*rule-tac x*=*w* + *z* **in** *exI*) **apply**(*rule*,*rule lem*[*OF w*(1) *z*(1), *unfolded* ∗[*THEN sym*]])
using *norm-triangle-ineq*[*of w* − *k z* − *l*] *w*(2) *z*(2) **by**(*auto simp add*:*field-simps*)
qed qed qed

lemma *has-integral-sub*:
shows (*f has-integral k*) *s* ⟹ (*g has-integral l*) *s* ⟹ ((λ*x*. *f*(*x*) − *g*(*x*)) *has-integral* (*k* − *l*)) *s*
using *has-integral-add*[*OF* - *has-integral-neg*,*of f k s g l*] **unfolding** *algebra-simps* **by** *auto*

lemma *integral-0*: *integral s* (λ*x*::*real*^′*n*. 0::*real*^′*m*) = 0
by(*rule integral-unique has-integral-0*)+

lemma *integral-add*:
shows *f integrable-on s* ⟹ *g integrable-on s* ⟹
integral s (λ*x*. *f x* + *g x*) = *integral s f* + *integral s g*
apply(*rule integral-unique*) **apply**(*drule integrable-integral*)+
apply(*rule has-integral-add*) **by** *assumption*+

lemma *integral-cmul*:
shows *f integrable-on s* ⟹ *integral s* (λ*x*. *c* ∗$_R$ *f x*) = *c* ∗$_R$ *integral s f*
apply(*rule integral-unique*) **apply**(*drule integrable-integral*)+
apply(*rule has-integral-cmul*) **by** *assumption*+

lemma *integral-neg*:
shows *f integrable-on s* ⟹ *integral s* (λ*x*. − *f x*) = − *integral s f*
apply(*rule integral-unique*) **apply**(*drule integrable-integral*)+
apply(*rule has-integral-neg*) **by** *assumption*+

lemma *integral-sub*:
shows *f integrable-on s* $\Longrightarrow$ *g integrable-on s* $\Longrightarrow$ *integral s* $(\lambda x.\ f\ x - g\ x) =$ *integral s f* $-$ *integral s g*
apply(*rule integral-unique*) **apply**(*drule integrable-integral*)+
apply(*rule has-integral-sub*) **by** *assumption*+

lemma *integrable-0*: $(\lambda x.\ 0)$ *integrable-on s*
unfolding *integrable-on-def* **using** *has-integral-0* **by** *auto*

lemma *integrable-add*:
shows *f integrable-on s* $\Longrightarrow$ *g integrable-on s* $\Longrightarrow$ $(\lambda x.\ f\ x + g\ x)$ *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-add*)

lemma *integrable-cmul*:
shows *f integrable-on s* $\Longrightarrow$ $(\lambda x.\ c *_R f(x))$ *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-cmul*)

lemma *integrable-neg*:
shows *f integrable-on s* $\Longrightarrow$ $(\lambda x.\ -f(x))$ *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-neg*)

lemma *integrable-sub*:
shows *f integrable-on s* $\Longrightarrow$ *g integrable-on s* $\Longrightarrow$ $(\lambda x.\ f\ x - g\ x)$ *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-sub*)

lemma *integrable-linear*:
shows *f integrable-on s* $\Longrightarrow$ *bounded-linear h* $\Longrightarrow$ $(h\ o\ f)$ *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-linear*)

lemma *integral-linear*:
shows *f integrable-on s* $\Longrightarrow$ *bounded-linear h* $\Longrightarrow$ *integral s* $(h\ o\ f) = h($*integral s f*$)$
apply(*rule has-integral-unique*) **defer unfolding** *has-integral-integral*
apply(*drule has-integral-linear*,*assumption*,*assumption*) **unfolding** *has-integral-integral*[*THEN sym*]
apply(*rule integrable-linear*) **by** *assumption*+

lemma *integral-component-eq*[*simp*]: **fixes** *f*::*real^'n* $\Rightarrow$ *real^'m*
assumes *f integrable-on s* **shows** *integral s* $(\lambda x.\ f\ x\ \$\ k) =$ *integral s f* $\$\ k$
using *integral-linear*[*OF assms*(*1*) *bounded-linear-component*,*unfolded o-def*] **.**

lemma *has-integral-setsum*:
assumes *finite t* $\forall\, a{\in}t.\ ((f\ a)$ *has-integral* $(i\ a))\ s$
shows $((\lambda x.$ *setsum* $(\lambda a.\ f\ a\ x)\ t)$ *has-integral* $($*setsum i t*$))\ s$
proof(*insert assms*(*1*) *subset-refl*[*of t*],*induct rule*:*finite-subset-induct*)
case (*insert x F*) **show** *?case* **unfolding** *setsum-insert*[*OF insert*(*1*,*3*)]
apply(*rule has-integral-add*) **using** *insert assms* **by** *auto*
qed *auto*

lemma *integral-setsum*:
shows *finite t* $\Longrightarrow$ $\forall$ *a*$\in$*t*. (*f a*) *integrable-on s* $\Longrightarrow$
integral s (λ*x*. *setsum* (λ*a*. *f a x*) *t*) = *setsum* (λ*a*. *integral s* (*f a*)) *t*
apply(*rule integral-unique*) **apply**(*rule has-integral-setsum*)
using *integrable-integral* **by** *auto*

lemma *integrable-setsum*:
shows *finite t* $\Longrightarrow$ $\forall$ *a* $\in$ *t*.(*f a*) *integrable-on s* $\Longrightarrow$ (λ*x*. *setsum* (λ*a*. *f a x*) *t*) *integrable-on s*
unfolding *integrable-on-def* **apply**(*drule bchoice*) **using** *has-integral-setsum*[*of t*] **by** *auto*

lemma *has-integral-eq*:
assumes $\forall$ *x*$\in$*s*. *f x* = *g x* (*f has-integral k*) *s* **shows** (*g has-integral k*) *s*
using *has-integral-sub*[*OF assms*(*2*), *of* λ*x*. *f x* − *g x 0*]
using *has-integral-is-0*[*of s* λ*x*. *f x* − *g x*] **using** *assms*(*1*) **by** *auto*

lemma *integrable-eq*:
shows $\forall$ *x*$\in$*s*. *f x* = *g x* $\Longrightarrow$ *f integrable-on s* $\Longrightarrow$ *g integrable-on s*
unfolding *integrable-on-def* **using** *has-integral-eq*[*of s f g*] **by** *auto*

lemma *has-integral-eq-eq*:
shows $\forall$ *x*$\in$*s*. *f x* = *g x* $\Longrightarrow$ ((*f has-integral k*) *s* $\longleftrightarrow$ (*g has-integral k*) *s*)
using *has-integral-eq*[*of s f g*] *has-integral-eq*[*of s g f*] **by** *rule auto*

lemma *has-integral-null*[*dest*]:
assumes *content*({*a*..*b*}) = *0* **shows** (*f has-integral 0*) ({*a*..*b*})
unfolding *has-integral* **apply**(*rule*,*rule*,*rule-tac x*=λ*x*. *ball x 1* **in** *exI*,*rule*) **defer**
proof(*rule*,*rule*,*erule conjE*) **fix** *e*::*real* **assume** *e*:*e*>*0* **thus** *gauge* (λ*x*. *ball x 1*)
by *auto*
fix *p* **assume** *p*:*p tagged-division-of* {*a*..*b*}
have *norm* ((∑ (*x*, *k*)∈*p*. *content k* $*_R$ *f x*) − *0*) = *0* **unfolding** *norm-eq-zero diff-0-right*
using *setsum-content-null*[*OF assms*(*1*) *p*, *of f*] .
thus *norm* ((∑ (*x*, *k*)∈*p*. *content k* $*_R$ *f x*) − *0*) < *e* **using** *e* **by** *auto* **qed**

lemma *has-integral-null-eq*[*simp*]:
shows *content*({*a*..*b*}) = *0* $\Longrightarrow$ ((*f has-integral i*) ({*a*..*b*}) $\longleftrightarrow$ *i* = *0*)
apply *rule* **apply**(*rule has-integral-unique*,*assumption*)
apply(*drule has-integral-null*,*assumption*)
apply(*drule has-integral-null*) **by** *auto*

lemma *integral-null*[*dest*]: **shows** *content*({*a*..*b*}) = *0* $\Longrightarrow$ *integral*({*a*..*b*}) *f* = *0*
by(*rule integral-unique*,*drule has-integral-null*)

lemma *integrable-on-null*[*dest*]: **shows** *content*({*a*..*b*}) = *0* $\Longrightarrow$ *f integrable-on* {*a*..*b*}
unfolding *integrable-on-def* **apply**(*drule has-integral-null*) **by** *auto*

lemma *has-integral-empty*[*intro*]: **shows** (*f has-integral 0*) {}
unfolding *empty-as-interval* **apply**(*rule has-integral-null*)
using *content-empty* **unfolding** *empty-as-interval* **.**

lemma *has-integral-empty-eq*[*simp*]: **shows** (*f has-integral i*) {} ⟷ *i* = *0*
apply(*rule*,*rule has-integral-unique*,*assumption*) **by** *auto*

lemma *integrable-on-empty*[*intro*]: **shows** *f integrable-on* {} **unfolding** *integrable-on-def*
by *auto*

lemma *integral-empty*[*simp*]: **shows** *integral* {} *f* = *0*
apply(*rule integral-unique*) **using** *has-integral-empty* **.**

lemma *has-integral-refl*[*intro*]: **shows** (*f has-integral 0*) {*a*..*a*} (*f has-integral 0*)
{*a*}
proof− **have** ∗:{*a*} = {*a*..*a*} **apply**(*rule set-ext*) **unfolding** *mem-interval singleton-iff*
Cart-eq
apply *safe* **prefer** *3* **apply**(*erule-tac x*=*i* **in** *allE*) **by**(*auto simp add*: *field-simps*)
show (*f has-integral 0*) {*a*..*a*} (*f has-integral 0*) {*a*} **unfolding** ∗
apply(*rule-tac*[!] *has-integral-null*) **unfolding** *content-eq-0-interior*
unfolding *interior-closed-interval* **using** *interval-sing* **by** *auto* **qed**

lemma *integrable-on-refl*[*intro*]: **shows** *f integrable-on* {*a*..*a*} **unfolding** *integrable-on-def*
by *auto*

lemma *integral-refl*: **shows** *integral* {*a*..*a*} *f* = *0* **apply**(*rule integral-unique*) **by**
auto

23.15 Cauchy-type criterion for integrability.

lemma *integrable-cauchy*: **fixes** *f*::*real*^'*n* ⇒ '*a*::{*real-normed-vector*,*complete-space*}

shows *f integrable-on* {*a*..*b*} ⟷
(∀ *e*>*0*.∃ *d*. *gauge d* ∧ (∀ *p1 p2*. *p1 tagged-division-of* {*a*..*b*} ∧ *d fine p1* ∧
p2 tagged-division-of {*a*..*b*} ∧ *d fine p2*
⟶ *norm*(*setsum* (λ(*x*,*k*). *content k* ∗*R f x*) *p1* −
setsum (λ(*x*,*k*). *content k* ∗*R f x*) *p2*) < *e*)) (**is** *?l* =
(∀ *e*>*0*. ∃ *d*. *?P e d*))
proof assume *?l*
then guess *y* **unfolding** *integrable-on-def has-integral* **..** **note** *y*=*this*
show ∀ *e*>*0*. ∃ *d*. *?P e d* **proof**(*rule*,*rule*) **case** *goal1* **hence** *e*/*2* > *0* **by** *auto*
then guess *d* **apply**− **apply**(*drule y*[*rule-format*]) **by**(*erule exE*,*erule conjE*)
note *d*=*this*[*rule-format*]
show *?case* **apply**(*rule-tac x*=*d* **in** *exI*,*rule*,*rule d*) **apply**(*rule*,*rule*,*rule*,(*erule*
conjE)+)
proof− **fix** *p1 p2* **assume** *as*:*p1 tagged-division-of* {*a*..*b*} *d fine p1 p2 tagged-division-of*
{*a*..*b*} *d fine p2*
show *norm* ((∑ (*x*, *k*)∈*p1*. *content k* ∗*R f x*) − (∑ (*x*, *k*)∈*p2*. *content k* ∗*R*

```
f x)) < e
      apply(rule dist-triangle-half-l[where y=y,unfolded dist-norm])
      using d(2)[OF conjI[OF as(1-2)]] d(2)[OF conjI[OF as(3-4)]] .
   qed qed
next assume ∀ e>0. ∃ d. ?P e d hence ∀ n::nat. ∃ d. ?P (inverse(real (n + 1)))
d by auto
 from choice[OF this] guess d .. note d=conjunctD2[OF this[rule-format],rule-format]
  have ⋀n. gauge (λx. ⋂{d i x |i. i ∈ {0..n}}) apply(rule gauge-inters) using
d(1) by auto
  hence ∀ n. ∃ p. p tagged-division-of {a..b} ∧ (λx. ⋂{d i x |i. i ∈ {0..n}}) fine
p apply-
 proof case goal1 from this[of n] show ?case apply(drule-tac fine-division-exists)
by auto qed
  from choice[OF this] guess p .. note p = conjunctD2[OF this[rule-format]]
  have dp:⋀i n. i≤n ⟹ d i fine p n using p(2) unfolding fine-inters by auto
  have Cauchy (λn. setsum (λ(x,k). content k *R (f x)) (p n))
  proof(rule CauchyI) case goal1 then guess N unfolding real-arch-inv[of e] ..
note N=this
    show ?case apply(rule-tac x=N in exI)
    proof(rule,rule,rule,rule) fix m n assume mn:N ≤ m N ≤ n have *:N = (N
- 1) + 1 using N by auto
      show norm ((∑(x, k)∈p m. content k *R f x) - (∑(x, k)∈p n. content k
*R f x)) < e
        apply(rule less-trans[OF - N[THEN conjunct2,THEN conjunct2]]) ap-
ply(subst *) apply(rule d(2))
       using dp p(1) using mn by auto
    qed qed
 then guess y unfolding convergent-eq-cauchy[THEN sym] .. note y=this[unfolded
Lim-sequentially,rule-format]
  show ?l unfolding integrable-on-def has-integral apply(rule-tac x=y in exI)
  proof(rule,rule) fix e::real assume e>0 hence *:e/2 > 0 by auto
   then guess N1 unfolding real-arch-inv[of e/2] .. note N1=this hence N1':N1
= N1 - 1 + 1 by auto
    guess N2 using y[OF *] .. note N2=this
     show ∃ d. gauge d ∧ (∀ p. p tagged-division-of {a..b} ∧ d fine p ⟶ norm
((∑(x, k)∈p. content k *R f x) - y) < e)
      apply(rule-tac x=d (N1 + N2) in exI) apply rule defer
    proof(rule,rule,erule conjE) show gauge (d (N1 + N2)) using d by auto
      fix q assume as:q tagged-division-of {a..b} d (N1 + N2) fine q
      have *:inverse (real (N1 + N2 + 1)) < e / 2 apply(rule less-trans) using
N1 by auto
    show norm ((∑(x, k)∈q. content k *R f x) - y) < e apply(rule norm-triangle-half-r)
       apply(rule less-trans[OF - *]) apply(subst N1', rule d(2)[of p (N1+N2)])
defer
       using N2[rule-format,unfolded dist-norm,of N1+N2]
        using as dp[of N1 - 1 + 1 + N2 N1 + N2] using p(1)[of N1 + N2]
using N1 by auto qed qed qed
```

23.16 Additivity of integral on abutting intervals.

lemma *interval-split*:
 $\{a..b::real\hat{}'n\} \cap \{x.\ x\$k \le c\} = \{a\ ..\ (\chi\ i.\ if\ i = k\ then\ min\ (b\$k)\ c\ else\ b\$i)\}$
 $\{a..b\} \cap \{x.\ x\$k \ge c\} = \{(\chi\ i.\ if\ i = k\ then\ max\ (a\$k)\ c\ else\ a\$i)\ ..\ b\}$
 apply(*rule-tac*[!] *set-ext*) **unfolding** *Int-iff mem-interval mem-Collect-eq*
 unfolding *Cart-lambda-beta* **by** *auto*

lemma *content-split*:
 content $\{a..b::real\hat{}'n\} = content(\{a..b\} \cap \{x.\ x\$k \le c\}) + content(\{a..b\} \cap \{x.\ x\$k >= c\})$
proof− **note** *simps* = *interval-split content-closed-interval-cases Cart-lambda-beta vector-le-def*
 { presume $a{\le}b \Longrightarrow$ *?thesis* **thus** *?thesis* **apply**(*cases* $a{\le}b$) **unfolding** *simps* **by** *auto* **}**
 have $*$:$UNIV = insert\ k\ (UNIV - \{k\})\ \bigwedge x.\ finite\ (UNIV - \{x::'n\})\ \bigwedge x.\ x \notin UNIV - \{x\}$ **by** *auto*
 have $*$:$\bigwedge X\ Y\ Z.\ (\prod i{\in}UNIV.\ Z\ i\ (if\ i = k\ then\ X\ else\ Y\ i)) = Z\ k\ X\ *\ (\prod i{\in}UNIV - \{k\}.\ Z\ i\ (Y\ i))$
 $(\prod i{\in}UNIV.\ b\$i - a\$i) = (\prod i{\in}UNIV - \{k\}.\ b\$i - a\$i) * (b\$k - a\$k)$
 apply(*subst* $*(1)$) **defer apply**(*subst* $*(1)$) **unfolding** *setprod-insert*[*OF* $*(2-)$] **by** *auto*
 assume *as*:$a{\le}b$ **moreover have** $\bigwedge x.\ min\ (b\ \$\ k)\ c = max\ (a\ \$\ k)\ c \Longrightarrow x*\ (b\$k - a\$k) = x*(max\ (a\ \$\ k)\ c - a\ \$\ k) + x*(b\ \$\ k - max\ (a\ \$\ k)\ c)$
 by (*auto simp add*:*field-simps*)
 moreover have $\neg\ a\ \$\ k \le c \Longrightarrow \neg\ c \le b\ \$\ k \Longrightarrow False$
 unfolding *not-le* **using** *as*[*unfolded vector-le-def*,*rule-format*,*of k*] **by** *auto*
 ultimately show *?thesis*
 unfolding *simps* **unfolding** $*(1)$[*of* $\lambda i\ x.\ b\$i - x$] $*(1)$[*of* $\lambda i\ x.\ x - a\$i$] $*(2)$ **by**(*auto*)
qed

lemma *division-split-left-inj*:
 assumes *d division-of i k1* $\in$ *d k2* $\in$ *d* $k1 \neq k2$
 $k1 \cap \{x::real\hat{}'n.\ x\$k \le c\} = k2 \cap \{x.\ x\$k \le c\}$
 shows $content(k1 \cap \{x.\ x\$k \le c\}) = 0$
proof− **note** *d*=*division-ofD*[*OF assms*(*1*)]
 have $*$:$\bigwedge a\ b::real\hat{}'n.\ \bigwedge\ c\ k.\ (content(\{a..b\} \cap \{x.\ x\$k \le c\}) = 0 \longleftrightarrow interior(\{a..b\} \cap \{x.\ x\$k \le c\}) = \{\})$
 unfolding *interval-split content-eq-0-interior* **by** *auto*
 guess *u1 v1* **using** *d*(*4*)[*OF assms*(*2*)] **apply**−**by**(*erule exE*)+ **note** *uv1*=*this*
 guess *u2 v2* **using** *d*(*4*)[*OF assms*(*3*)] **apply**−**by**(*erule exE*)+ **note** *uv2*=*this*
 have $**$:$\bigwedge s\ t\ u.\ s \cap t = \{\} \Longrightarrow u \subseteq s \Longrightarrow u \subseteq t \Longrightarrow u = \{\}$ **by** *auto*
 show *?thesis* **unfolding** *uv1 uv2* $*$ **apply**(*rule* $**$[*OF d*(*5*)[*OF assms*(*2-4*)]])
 defer apply(*subst assms*(*5*)[*unfolded uv1 uv2*]) **unfolding** *uv1 uv2* **by** *auto*
qed

lemma *division-split-right-inj*:
 assumes *d division-of i k1* $\in$ *d k2* $\in$ *d* $k1 \neq k2$

k1 ∩ {*x*::*real*^'*n*. *x*$*k* ≥ *c*} = *k2* ∩ {*x*. *x*$*k* ≥ *c*}
shows *content*(*k1* ∩ {*x*. *x*$*k* ≥ *c*}) = *0*
proof− **note** *d*=*division-ofD*[*OF assms*(*1*)]
have ∗:⋀*a b*::*real*^'*n*. ⋀ *c k*. (*content*({*a*..*b*} ∩ {*x*. *x*$*k* >= *c*}) = *0* ⟷ *interior*({*a*..*b*} ∩ {*x*. *x*$*k* >= *c*}) = {})
unfolding *interval-split content-eq-0-interior* **by** *auto*
guess *u1 v1* **using** *d*(*4*)[*OF assms*(*2*)] **apply**−**by**(*erule exE*)+ **note** *uv1*=*this*
guess *u2 v2* **using** *d*(*4*)[*OF assms*(*3*)] **apply**−**by**(*erule exE*)+ **note** *uv2*=*this*
have ∗∗:⋀*s t u*. *s* ∩ *t* = {} ⟹ *u* ⊆ *s* ⟹ *u* ⊆ *t* ⟹ *u* = {} **by** *auto*
show *?thesis* **unfolding** *uv1 uv2* ∗ **apply**(*rule* ∗∗[*OF d*(*5*)[*OF assms*(*2*−*4*)]])
defer apply(*subst assms*(*5*)[*unfolded uv1 uv2*]) **unfolding** *uv1 uv2* **by** *auto*
qed

lemma *tagged-division-split-left-inj*:
assumes *d tagged-division-of i* (*x1*,*k1*) ∈ *d* (*x2*,*k2*) ∈ *d k1* ≠ *k2* *k1* ∩ {*x*. *x*$*k* ≤ *c*} = *k2* ∩ {*x*. *x*$*k* ≤ *c*}
shows *content*(*k1* ∩ {*x*. *x*$*k* ≤ *c*}) = *0*
proof− **have** ∗:⋀*a b c*. (*a*,*b*) ∈ *c* ⟹ *b* ∈ *snd* ' *c* **unfolding** *image-iff* **apply**(*rule-tac x*=(*a*,*b*) **in** *bexI*) **by** *auto*
show *?thesis* **apply**(*rule division-split-left-inj*[*OF division-of-tagged-division*[*OF assms*(*1*)]])
apply(*rule-tac*[*1*−*2*] ∗) **using** *assms*(*2*−) **by** *auto* **qed**

lemma *tagged-division-split-right-inj*:
assumes *d tagged-division-of i* (*x1*,*k1*) ∈ *d* (*x2*,*k2*) ∈ *d k1* ≠ *k2* *k1* ∩ {*x*. *x*$*k* ≥ *c*} = *k2* ∩ {*x*. *x*$*k* ≥ *c*}
shows *content*(*k1* ∩ {*x*. *x*$*k* ≥ *c*}) = *0*
proof− **have** ∗:⋀*a b c*. (*a*,*b*) ∈ *c* ⟹ *b* ∈ *snd* ' *c* **unfolding** *image-iff* **apply**(*rule-tac x*=(*a*,*b*) **in** *bexI*) **by** *auto*
show *?thesis* **apply**(*rule division-split-right-inj*[*OF division-of-tagged-division*[*OF assms*(*1*)]])
apply(*rule-tac*[*1*−*2*] ∗) **using** *assms*(*2*−) **by** *auto* **qed**

lemma *division-split*:
assumes *p division-of* {*a*..*b*::*real*^'*n*}
shows {*l* ∩ {*x*. *x*$*k* ≤ *c*} | *l*. *l* ∈ *p* ∧ ~(*l* ∩ {*x*. *x*$*k* ≤ *c*} = {})} *division-of* ({*a*..*b*} ∩ {*x*. *x*$*k* ≤ *c*}) (**is** *?p1 division-of ?I1*) **and**
{*l* ∩ {*x*. *x*$*k* ≥ *c*} | *l*. *l* ∈ *p* ∧ ~(*l* ∩ {*x*. *x*$*k* ≥ *c*} = {})} *division-of* ({*a*..*b*} ∩ {*x*. *x*$*k* ≥ *c*}) (**is** *?p2 division-of ?I2*)
proof(*rule-tac*[!] *division-ofI*) **note** *p*=*division-ofD*[*OF assms*]
show *finite ?p1 finite ?p2* **using** *p*(*1*) **by** *auto* **show** ⋃ *?p1* = *?I1* ⋃ *?p2* = *?I2* **unfolding** *p*(*6*)[*THEN sym*] **by** *auto*
{ fix *k* **assume** *k*∈*?p1* **then guess** *l* **unfolding** *mem-Collect-eq* **apply**−**by**(*erule exE*,(*erule conjE*)+) **note** *l*=*this*
guess *u v* **using** *p*(*4*)[*OF l*(*2*)] **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
show *k*⊆*?I1 k* ≠ {} ∃ *a b*. *k* = {*a*..*b*} **unfolding** *l*
using *p*(*2*−*3*)[*OF l*(*2*)] *l*(*3*) **unfolding** *uv* **apply**− **prefer** *3* **apply**(*subst interval-split*) **by** *auto*
fix *k'* **assume** *k'*∈*?p1* **then guess** *l'* **unfolding** *mem-Collect-eq* **apply**−**by**(*erule*

exE,(*erule conjE*)+) **note** *l'=this*
assume *k≠k'* **thus** *interior k ∩ interior k' = {}* **unfolding** *l l'* **using** *p(5)[OF l(2) l'(2)]* **by** *auto* **}**
{ fix *k* **assume** *k∈?p2* **then guess** *l* **unfolding** *mem-Collect-eq* **apply−by**(*erule exE*,(*erule conjE*)+) **note** *l=this*
guess *u v* **using** *p(4)[OF l(2)]* **apply−by**(*erule exE*)+ **note** *uv=this*
show *k⊆?I2 k ≠ {} ∃ a b. k = {a..b}* **unfolding** *l*
using *p(2−3)[OF l(2)] l(3)* **unfolding** *uv* **apply−** **prefer** *3* **apply**(*subst interval-split*) **by** *auto*
fix *k'* **assume** *k'∈?p2* **then guess** *l'* **unfolding** *mem-Collect-eq* **apply−by**(*erule exE*,(*erule conjE*)+) **note** *l'=this*
assume *k≠k'* **thus** *interior k ∩ interior k' = {}* **unfolding** *l l'* **using** *p(5)[OF l(2) l'(2)]* **by** *auto* **}**
qed

lemma *has-integral-split*: **fixes** *f*::*real^'n ⇒ 'a*::*real-normed-vector*
assumes (*f has-integral i*) (*{a..b} ∩ {x. x$k ≤ c}*) (*f has-integral j*) (*{a..b} ∩ {x. x$k ≥ c}*)
shows (*f has-integral* (*i + j*)) (*{a..b}*)
proof(*unfold has-integral*,*rule*,*rule*) **case** *goal1* **hence** *e*:*e/2>0* **by** *auto*
guess *d1* **using** *has-integralD*[*OF assms(1)*[*unfolded interval-split*] *e*] **. note** *d1=this*[*unfolded interval-split*[*THEN sym*]]
guess *d2* **using** *has-integralD*[*OF assms(2)*[*unfolded interval-split*] *e*] **. note** *d2=this*[*unfolded interval-split*[*THEN sym*]]
let *?d = λx. if x$k = c then* (*d1 x ∩ d2 x*) *else ball x* (*abs*(*x$k − c*)) *∩ d1 x ∩ d2 x*
show *?case* **apply**(*rule-tac x=?d* **in** *exI*,*rule*) **defer apply**(*rule*,*rule*,(*erule conjE*)+)
proof− **show** *gauge ?d* **using** *d1(1) d2(1)* **unfolding** *gauge-def* **by** *auto*
fix *p* **assume** *p tagged-division-of {a..b} ?d fine p* **note** *p = this tagged-division-ofD*[*OF this(1)*]
have *lem0*:⋀*x kk.* (*x*,*kk*) ∈ *p* ⟹ ~(*kk ∩ {x. x$k ≤ c} = {}*) ⟹ *x$k ≤ c*
⋀*x kk.* (*x*,*kk*) ∈ *p* ⟹ ~(*kk ∩ {x. x$k ≥ c} = {}*) ⟹ *x$k ≥ c*
proof− **fix** *x kk* **assume** *as*:(*x*,*kk*)∈*p*
show ~(*kk ∩ {x. x$k ≤ c} = {}*) ⟹ *x$k ≤ c*
proof(*rule ccontr*) **case** *goal1*
from *this(2)*[*unfolded not-le*] **have** *kk ⊆ ball x |x $ k − c|*
using *p(2)*[*unfolded fine-def*,*rule-format*,*OF as*,*unfolded split-conv*] **by** *auto*
hence ∃ *y. y ∈ ball x |x $ k − c| ∩ {x. x $ k ≤ c}* **using** *goal1(1)* **by** *blast*
then guess *y* **.. hence** *|x $ k − y $ k| < |x $ k − c| y$k ≤ c* **apply−apply**(*rule le-less-trans*)
using *component-le-norm*[*of x − y k*,*unfolded vector-minus-component*] **by**(*auto simp add*:*dist-norm*)
thus *False* **using** *goal1(2)*[*unfolded not-le*] **by**(*auto simp add*:*field-simps*)
qed
show ~(*kk ∩ {x. x$k ≥ c} = {}*) ⟹ *x$k ≥ c*
proof(*rule ccontr*) **case** *goal1*
from *this(2)*[*unfolded not-le*] **have** *kk ⊆ ball x |x $ k − c|*
using *p(2)*[*unfolded fine-def*,*rule-format*,*OF as*,*unfolded split-conv*] **by**

auto
hence $\exists y.\ y \in ball\ x\ |x\ \$\ k - c| \cap \{x.\ x\ \$\ k \geq c\}$ **using** *goal1*(*1*) **by** *blast*
then guess *y* **.. hence** $|x\ \$\ k - y\ \$\ k| < |x\ \$\ k - c|\ y\$k \geq c$ **apply**−**apply**(*rule le-less-trans*)
using *component-le-norm*[*of x − y k*,*unfolded vector-minus-component*] **by**(*auto simp add*:*dist-norm*)
thus *False* **using** *goal1*(*2*)[*unfolded not-le*] **by**(*auto simp add*:*field-simps*)
qed
qed

have *lem1*: $\bigwedge f\ P\ Q.\ (\forall x\ k.\ (x,k) \in \{(x,f\ k) \mid x\ k.\ P\ x\ k\} \longrightarrow Q\ x\ k) \longleftrightarrow (\forall x\ k.\ P\ x\ k \longrightarrow Q\ x\ (f\ k))$ **by** *auto*
have *lem2*: $\bigwedge f\ s\ P\ f.\ finite\ s \Longrightarrow finite\ \{(x,f\ k) \mid x\ k.\ (x,k) \in s \wedge P\ x\ k\}$
proof− **case** *goal1* **thus** *?case* **apply**−**apply**(*rule finite-subset*[*of* - $(\lambda(x,k).\ (x,f\ k))\ `\ s$]) **by** *auto* **qed**
have *lem3*: $\bigwedge g$::$(real\ \hat{}\ 'n \Rightarrow bool) \Rightarrow real\ \hat{}\ 'n \Rightarrow bool.\ finite\ p \Longrightarrow$
$setsum\ (\lambda(x,k).\ content\ k *_R f\ x)\ \{(x,g\ k)\ |x\ k.\ (x,k) \in p \wedge \sim(g\ k = \{})\}$
$= setsum\ (\lambda(x,k).\ content\ k *_R f\ x)\ ((\lambda(x,k).\ (x,g\ k))\ `\ p)$
apply(*rule setsum-mono-zero-left*) **prefer** *3*
proof fix g::$(real\ \hat{}\ 'n \Rightarrow bool) \Rightarrow real\ \hat{}\ 'n \Rightarrow bool$ **and** i::$(real\hat{}'n) \times ((real\hat{}'n)\ set)$
assume $i \in (\lambda(x,\ k).\ (x,\ g\ k))\ `\ p - \{(x,\ g\ k)\ |x\ k.\ (x,\ k) \in p \wedge g\ k \neq \{}\}$
then obtain *x k* **where** *xk*:i=$(x,g\ k)\ (x,k) \in p\ (x,g\ k) \notin \{(x,\ g\ k)\ |x\ k.\ (x,\ k) \in p \wedge g\ k \neq \{}\}$ **by** *auto*
have $content\ (g\ k) = 0$ **using** *xk* **using** *content-empty* **by** *auto*
thus $(\lambda(x,\ k).\ content\ k *_R f\ x)\ i = 0$ **unfolding** *xk split-conv* **by** *auto*
qed *auto*
have *lem4*:$\bigwedge g.\ (\lambda(x,l).\ content\ (g\ l) *_R f\ x) = (\lambda(x,l).\ content\ l *_R f\ x)\ o\ (\lambda(x,l).\ (x,g\ l))$ **apply**(*rule ext*) **by** *auto*

let *?M1* = $\{(x,kk \cap \{x.\ x\$k \leq c\})\ |x\ kk.\ (x,kk) \in p \wedge kk \cap \{x.\ x\$k \leq c\} \neq \{}\}$
have *norm* $((\sum(x,\ k)\in ?M1.\ content\ k *_R f\ x) - i) < e/2$ **apply**(*rule d1*(*2*),*rule tagged-division-ofI*)
apply(*rule lem2 p*(*3*))+ **prefer** *6* **apply**(*rule fineI*)
proof− **show** $\bigcup\{k.\ \exists x.\ (x,\ k) \in ?M1\} = \{a..b\} \cap \{x.\ x\$k \leq c\}$ **unfolding** *p*(*8*)[*THEN sym*] **by** *auto*
fix *x l* **assume** *xl*:$(x,l) \in ?M1$
then guess *x' l'* **unfolding** *mem-Collect-eq* **apply**− **unfolding** *Pair-eq* **apply**((*erule exE*)+,(*erule conjE*)+) **. note** *xl'*=*this*
have $l' \subseteq d1\ x'$ **apply**(*rule order-trans*[*OF fineD*[*OF p*(*2*) *xl'*(*3*)]]) **by** *auto*
thus $l \subseteq d1\ x$ **unfolding** *xl'* **by** *auto*
show $x \in l\ l \subseteq \{a..b\} \cap \{x.\ x\ \$\ k \leq c\}$ **unfolding** *xl'* **using** *p*(*4*−*6*)[*OF xl'*(*3*)] **using** *xl'*(*4*)
using *lem0*(*1*)[*OF xl'*(*3*−*4*)] **by** *auto*
show $\exists a\ b.\ l = \{a..b\}$ **unfolding** *xl'* **using** *p*(*6*)[*OF xl'*(*3*)] **by**(*fastsimp simp add*: *interval-split*[**where** *c*=*c* **and** *k*=*k*])
fix *y r* **let** *?goal* = $interior\ l \cap interior\ r = \{}$ **assume** *yr*:$(y,r) \in ?M1$
then guess *y' r'* **unfolding** *mem-Collect-eq* **apply**− **unfolding** *Pair-eq*

apply((*erule exE*)+,(*erule conjE*)+) **.** **note** *yr'=this*
assume *as*:(*x*,*l*) ≠ (*y*,*r*) **show** *interior l* ∩ *interior r* = {}
proof(*cases l'* = *r'* ⟶ *x'* = *y'*)
case *False* **thus** *?thesis* **using** *p*(*7*)[*OF xl'*(*3*) *yr'*(*3*)] **using** *as* **unfolding** *xl' yr'* **by** *auto*
next case *True* **hence** *l'* ≠ *r'* **using** *as* **unfolding** *xl' yr'* **by** *auto*
thus *?thesis* **using** *p*(*7*)[*OF xl'*(*3*) *yr'*(*3*)] **using** *as* **unfolding** *xl' yr'* **by** *auto*
qed qed moreover

let *?M2* = {(*x*,*kk* ∩ {*x*. *x*$*k* ≥ *c*}) |*x kk*. (*x*,*kk*) ∈ *p* ∧ *kk* ∩ {*x*. *x*$*k* ≥ *c*} ≠ {}}
have *norm* ((∑(*x*, *k*)∈*?M2*. *content k* ∗*R f x*) − *j*) < *e*/*2* **apply**(*rule d2*(*2*),*rule tagged-division-ofI*)
apply(*rule lem2 p*(*3*))+ **prefer** *6* **apply**(*rule fineI*)
proof− **show** ⋃{*k*. ∃*x*. (*x*, *k*) ∈ *?M2*} = {*a*..*b*} ∩ {*x*. *x*$*k* ≥ *c*} **unfolding** *p*(*8*)[*THEN sym*] **by** *auto*
fix *x l* **assume** *xl*:(*x*,*l*)∈*?M2*
then guess *x' l'* **unfolding** *mem-Collect-eq* **apply**− **unfolding** *Pair-eq* **apply**((*erule exE*)+,(*erule conjE*)+) **.** **note** *xl'=this*
have *l'* ⊆ *d2 x'* **apply**(*rule order-trans*[*OF fineD*[*OF p*(*2*) *xl'*(*3*)]]) **by** *auto*
thus *l* ⊆ *d2 x* **unfolding** *xl'* **by** *auto*
show *x*∈*l l* ⊆ {*a*..*b*} ∩ {*x*. *x* $ *k* ≥ *c*} **unfolding** *xl'* **using** *p*(*4*−*6*)[*OF xl'*(*3*)] **using** *xl'*(*4*)
using *lem0*(*2*)[*OF xl'*(*3*−*4*)] **by** *auto*
show ∃*a b*. *l* = {*a*..*b*} **unfolding** *xl'* **using** *p*(*6*)[*OF xl'*(*3*)] **by**(*fastsimp simp add*: *interval-split*[**where** *c*=*c* **and** *k*=*k*])
fix *y r* **let** *?goal* = *interior l* ∩ *interior r* = {} **assume** *yr*:(*y*,*r*)∈*?M2*
then guess *y' r'* **unfolding** *mem-Collect-eq* **apply**− **unfolding** *Pair-eq* **apply**((*erule exE*)+,(*erule conjE*)+) **.** **note** *yr'=this*
assume *as*:(*x*,*l*) ≠ (*y*,*r*) **show** *interior l* ∩ *interior r* = {}
proof(*cases l'* = *r'* ⟶ *x'* = *y'*)
case *False* **thus** *?thesis* **using** *p*(*7*)[*OF xl'*(*3*) *yr'*(*3*)] **using** *as* **unfolding** *xl' yr'* **by** *auto*
next case *True* **hence** *l'* ≠ *r'* **using** *as* **unfolding** *xl' yr'* **by** *auto*
thus *?thesis* **using** *p*(*7*)[*OF xl'*(*3*) *yr'*(*3*)] **using** *as* **unfolding** *xl' yr'* **by** *auto*
qed qed ultimately

have *norm* (((∑(*x*, *k*)∈*?M1*. *content k* ∗*R f x*) − *i*) + ((∑(*x*, *k*)∈*?M2*. *content k* ∗*R f x*) − *j*)) < *e*/*2* + *e*/*2*
apply− **apply**(*rule norm-triangle-lt*) **by** *auto*
also { **have** ∗:⋀*x y*. *x* = (*0*::*real*) ⟹ *x* ∗*R* (*y*::'*a*) = *0* **using** *scaleR-zero-left* **by** *auto*
have ((∑(*x*, *k*)∈*?M1*. *content k* ∗*R f x*) − *i*) + ((∑(*x*, *k*)∈*?M2*. *content k* ∗*R f x*) − *j*)
= (∑(*x*, *k*)∈*?M1*. *content k* ∗*R f x*) + (∑(*x*, *k*)∈*?M2*. *content k* ∗*R f x*) − (*i* + *j*) **by** *auto*
also have ... = (∑(*x*, *ka*)∈*p*. *content* (*ka* ∩ {*x*. *x* $ *k* ≤ *c*}) ∗*R f x*) +

$(\sum(x,\ ka){\in}p.\ content\ (ka \cap \{x.\ c \le x\ \$\ k\})\ *_R\ f\ x) - (i + j)$
unfolding *lem3*[*OF p*(*3*)] **apply**(*subst setsum-reindex-nonzero*[*OF p*(*3*)])
defer apply(*subst setsum-reindex-nonzero*[*OF p*(*3*)])
defer unfolding *lem4*[*THEN sym*] **apply**(*rule refl*) **unfolding** *split-paired-all split-conv* **apply**(*rule-tac*[!] *)
proof− **case** *goal1* **thus** *?case* **apply**− **apply**(*rule tagged-division-split-left-inj* [*OF p*(*1*), *of a b aa ba*]) **by** *auto*
next case *goal2* **thus** *?case* **apply**− **apply**(*rule tagged-division-split-right-inj*[*OF p*(*1*), *of a b aa ba*]) **by** *auto*
qed also note *setsum-addf*[*THEN sym*]
also have *:$\bigwedge x.\ x{\in}p \Longrightarrow (\lambda(x,\ ka).\ content\ (ka \cap \{x.\ x\ \$\ k \le c\})\ *_R\ f\ x)\ x$ + $(\lambda(x,\ ka).\ content\ (ka \cap \{x.\ c \le x\ \$\ k\})\ *_R\ f\ x)\ x$
= $(\lambda(x,ka).\ content\ ka\ *_R\ f\ x)\ x$ **unfolding** *split-paired-all split-conv*
proof− **fix** *a b* **assume** $(a,b) \in p$ **from** *p*(*6*)[*OF this*] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv=this*
thus $content\ (b \cap \{x.\ x\ \$\ k \le c\})\ *_R\ f\ a + content\ (b \cap \{x.\ c \le x\ \$\ k\})\ *_R\ f\ a = content\ b\ *_R\ f\ a$
unfolding *scaleR-left-distrib*[*THEN sym*] **unfolding** *uv content-split*[*of u v k c*] **by** *auto*
qed note *setsum-cong2*[*OF this*]
finally have $(\sum(x,\ k){\in}\{(x,\ kk \cap \{x.\ x\ \$\ k \le c\})\ |x\ kk.\ (x,\ kk) \in p \wedge kk \cap \{x.\ x\ \$\ k \le c\} \neq \{\}\}.\ content\ k\ *_R\ f\ x) - i +$
$((\sum(x,\ k){\in}\{(x,\ kk \cap \{x.\ c \le x\ \$\ k\})\ |x\ kk.\ (x,\ kk) \in p \wedge kk \cap \{x.\ c \le x\ \$\ k\} \neq \{\}\}.\ content\ k\ *_R\ f\ x) - j) =$
$(\sum(x,\ ka){\in}p.\ content\ ka\ *_R\ f\ x) - (i + j)$ **by** *auto* **}**
finally show $norm\ ((\sum(x,\ k){\in}p.\ content\ k\ *_R\ f\ x) - (i + j)) < e$ **by** *auto*
qed qed

23.17 A sort of converse, integrability on subintervals.

lemma *tagged-division-union-interval*:
assumes *p1 tagged-division-of* $(\{a..b\} \cap \{x{::}real\hat{}'n.\ x\$k \le (c{::}real)\})$ *p2 tagged-division-of* $(\{a..b\} \cap \{x.\ x\$k \ge c\})$
shows (*p1* ∪ *p2*) *tagged-division-of* $(\{a..b\})$
proof− **have** *:$\{a..b\} = (\{a..b\} \cap \{x.\ x\$k \le c\}) \cup (\{a..b\} \cap \{x.\ x\$k \ge c\})$ **by** *auto*
show *?thesis* **apply**(*subst* *) **apply**(*rule tagged-division-union*[*OF assms*])
unfolding *interval-split interior-closed-interval*
by(*auto simp add*: *vector-less-def elim*!:*allE*[**where** *x*=*k*]) **qed**

lemma *has-integral-separate-sides*: **fixes** *f*::*real*^'*m* ⇒ '*a*::*real-normed-vector*
assumes (*f has-integral i*) $(\{a..b\})$ *e*>*0*
obtains *d* **where** *gauge d* $(\forall p1\ p2.\ p1\ tagged\text{-}division\text{-}of\ (\{a..b\} \cap \{x.\ x\$k \le c\}) \wedge d\ fine\ p1 \wedge$
$p2\ tagged\text{-}division\text{-}of\ (\{a..b\} \cap \{x.\ x\$k \ge c\}) \wedge d\ fine\ p2$
$\longrightarrow norm((setsum\ (\lambda(x,k).\ content\ k\ *_R\ f\ x)\ p1 +$
$setsum\ (\lambda(x,k).\ content\ k\ *_R\ f\ x)\ p2) - i) < e)$
proof− **guess** *d* **using** *has-integralD*[*OF assms*] **. note** *d=this*
show *?thesis* **apply**(*rule that*[*of d*]) **apply**(*rule d*) **apply**(*rule*,*rule*,*rule*,(*erule*

conjE)+)
proof− **fix** *p1 p2* **assume** *p1 tagged-division-of* $\{a..b\} \cap \{x.\ x\ \$\ k \le c\}$ *d fine p1* **note** *p1*=*tagged-division-ofD*[*OF this*(*1*)] *this*
assume *p2 tagged-division-of* $\{a..b\} \cap \{x.\ c \le x\ \$\ k\}$ *d fine p2* **note** *p2*=*tagged-division-ofD*[*OF this*(*1*)] *this*
note *tagged-division-union-interval*[*OF p1*(*7*) *p2*(*7*)] **note** *p12* = *tagged-division-ofD*[*OF this*] *this*
have *norm* $((\sum(x,\ k){\in}p1.\ content\ k *_R f\ x) + (\sum(x,\ k){\in}p2.\ content\ k *_R f\ x) - i) = norm\ ((\sum(x,\ k){\in}p1 \cup p2.\ content\ k *_R f\ x) - i)$
apply(*subst setsum-Un-zero*) **apply**(*rule p1 p2*)+ **apply**(*rule*) **unfolding** *split-paired-all split-conv*
proof− **fix** *a b* **assume** *ab*:$(a,b) \in p1 \cap p2$
have $(a,b) \in p1$ **using** *ab* **by** *auto* **from** *p1*(*4*)[*OF this*] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv* =*this*
have $b \subseteq \{x.\ x\$k = c\}$ **using** *ab p1*(*3*)[*of a b*] *p2*(*3*)[*of a b*] **by** *fastsimp*
moreover have *interior* $\{x.\ x\ \$\ k = c\} = \{\}$
proof(*rule ccontr*) **case** *goal1* **then obtain** *x* **where** *x*:$x{\in}interior\ \{x.\ x\$k = c\}$ **by** *auto*
then guess *e* **unfolding** *mem-interior* **.. note** *e*=*this*
have *x*:$x\$k = c$ **using** *x interior-subset* **by** *fastsimp*
have ∗:$\bigwedge i.\ |(x - (x + (\chi\ i.\ if\ i = k\ then\ e\ /\ 2\ else\ 0)))\ \$\ i| = (if\ i = k\ then\ e/2\ else\ 0)$ **using** *e* **by** *auto*
have $x + (\chi\ i.\ if\ i = k\ then\ e/2\ else\ 0) \in ball\ x\ e$ **unfolding** *mem-ball dist-norm*
apply(*rule le-less-trans*[*OF norm-le-l1*]) **unfolding** ∗
unfolding *setsum-delta*[*OF finite-UNIV*] **using** *e* **by** *auto*
hence $x + (\chi\ i.\ if\ i = k\ then\ e/2\ else\ 0) \in \{x.\ x\$k = c\}$ **using** *e* **by** *auto*
thus *False* **unfolding** *mem-Collect-eq* **using** *e x* **by** *auto*
qed ultimately have *content b* = *0* **unfolding** *uv content-eq-0-interior* **apply**−**apply**(*drule subset-interior*) **by** *auto*
thus $content\ b *_R f\ a = 0$ **by** *auto*
qed *auto*
also have $\ldots < e$ **by**(*rule d*(*2*) *p12 fine-union p1 p2*)+
finally show $norm\ ((\sum(x,\ k){\in}p1.\ content\ k *_R f\ x) + (\sum(x,\ k){\in}p2.\ content\ k *_R f\ x) - i) < e$ **. qed qed**

lemma *integrable-split*[*intro*]: **fixes** *f*::*real*^'*n* ⇒ '*a*::{*real-normed-vector*,*complete-space*} **assumes** *f integrable-on* $\{a..b\}$
shows *f integrable-on* $(\{a..b\} \cap \{x.\ x\$k \le c\})$ (**is** *?t1*) **and** *f integrable-on* $(\{a..b\} \cap \{x.\ x\$k \ge c\})$ (**is** *?t2*)
proof− **guess** *y* **using** *assms* **unfolding** *integrable-on-def* **.. note** *y*=*this*
def $b' \equiv (\chi\ i.\ if\ i = k\ then\ min\ (b\$k)\ c\ else\ b\$i)$::*real*^'*n*
and $a' \equiv (\chi\ i.\ if\ i = k\ then\ max\ (a\$k)\ c\ else\ a\$i)$::*real*^'*n*
show *?t1 ?t2* **unfolding** *interval-split integrable-cauchy* **unfolding** *interval-split*[*THEN sym*]
proof(*rule-tac*[!] *allI impI*)+ **fix** *e*::*real* **assume** *e*>*0* **hence** *e*/*2*>*0* **by** *auto*
from *has-integral-separate-sides*[*OF y this*,*of k c*] **guess** *d* **. note** *d*=*this*[*rule-format*]
let *?P* = $\lambda A.\ \exists d.\ gauge\ d \wedge (\forall p1\ p2.\ p1\ tagged\text{-}division\text{-}of\ \{a..b\} \cap A \wedge d\ fine\ p1 \wedge p2\ tagged\text{-}division\text{-}of\ \{a..b\} \cap A \wedge d\ fine\ p2 \longrightarrow$

norm $((\sum (x,\ k){\in}p1.\ content\ k *_R f\ x) - (\sum (x,\ k){\in}p2.\ content\ k *_R f\ x)) < e$)
show *?P* $\{x.\ x\ \$\ k \le c\}$ **apply**(*rule-tac x=d* **in** *exI*) **apply**(*rule*,*rule d*) **apply**(*rule*,*rule*,*rule*)
proof− **fix** *p1 p2* **assume** *as*:*p1 tagged-division-of* $\{a..b\} \cap \{x.\ x\ \$\ k \le c\}$ ∧ *d fine p1* ∧ *p2 tagged-division-of* $\{a..b\} \cap \{x.\ x\ \$\ k \le c\}$ ∧ *d fine p2*
show *norm* $((\sum (x,\ k){\in}p1.\ content\ k *_R f\ x) - (\sum (x,\ k){\in}p2.\ content\ k *_R f\ x)) < e$
proof− **guess** *p* **using** *fine-division-exists*[*OF d*(*1*), *of a′ b*] **. note** *p*=*this*
show *?thesis* **using** *norm-triangle-half-l*[*OF d*(*2*)[*of p1 p*] *d*(*2*)[*of p2 p*]]
using *as* **unfolding** *interval-split b′-def*[*symmetric*] *a′-def*[*symmetric*]
using *p* **using** *assms* **by**(*auto simp add*:*algebra-simps*)
qed qed
show *?P* $\{x.\ x\ \$\ k \ge c\}$ **apply**(*rule-tac x=d* **in** *exI*) **apply**(*rule*,*rule d*) **apply**(*rule*,*rule*,*rule*)
proof− **fix** *p1 p2* **assume** *as*:*p1 tagged-division-of* $\{a..b\} \cap \{x.\ x\ \$\ k \ge c\}$ ∧ *d fine p1* ∧ *p2 tagged-division-of* $\{a..b\} \cap \{x.\ x\ \$\ k \ge c\}$ ∧ *d fine p2*
show *norm* $((\sum (x,\ k){\in}p1.\ content\ k *_R f\ x) - (\sum (x,\ k){\in}p2.\ content\ k *_R f\ x)) < e$
proof− **guess** *p* **using** *fine-division-exists*[*OF d*(*1*), *of a b′*] **. note** *p*=*this*
show *?thesis* **using** *norm-triangle-half-l*[*OF d*(*2*)[*of p p1*] *d*(*2*)[*of p p2*]]
using *as* **unfolding** *interval-split b′-def*[*symmetric*] *a′-def*[*symmetric*]
using *p* **using** *assms* **by**(*auto simp add*:*algebra-simps*) **qed qed qed qed**

23.18 Generalized notion of additivity.

definition *neutral opp* = (*SOME x*. ∀ *y*. *opp x y* = *y* ∧ *opp y x* = *y*)

definition *operative* :: ('*a* ⇒ '*a* ⇒ '*a*) ⇒ (('*real*^'*n*) *set* ⇒ '*a*) ⇒ *bool* **where**
operative opp f ≡
(∀ *a b*. *content* $\{a..b\}$ = *0* ⟶ *f* $\{a..b\}$ = *neutral*(*opp*)) ∧
(∀ *a b c k*. *f*($\{a..b\}$) =
opp (*f*($\{a..b\} \cap \{x.\ x\$k \le c\}$))
(*f*($\{a..b\} \cap \{x.\ x\$k \ge c\}$))))

lemma *operativeD*[*dest*]: **assumes** *operative opp f*
shows ⋀*a b*. *content* $\{a..b\}$ = *0* ⟹ *f* $\{a..b\}$ = *neutral*(*opp*)
⋀*a b c k*. *f*($\{a..b\}$) = *opp* (*f*($\{a..b\} \cap \{x.\ x\$k \le c\}$)) (*f*($\{a..b\} \cap \{x.\ x\$k \ge c\}$))
using *assms* **unfolding** *operative-def* **by** *auto*

lemma *operative-trivial*:
operative opp f ⟹ *content*($\{a..b\}$) = *0* ⟹ *f*($\{a..b\}$) = *neutral opp*
unfolding *operative-def* **by** *auto*

lemma *property-empty-interval*:
(∀ *a b*. *content*($\{a..b\}$) = *0* ⟶ *P*($\{a..b\}$)) ⟹ *P* {}
using *content-empty* **unfolding** *empty-as-interval* **by** *auto*

lemma *operative-empty*: *operative opp f* $\Longrightarrow$ *f* {} = *neutral opp*
unfolding *operative-def* **apply**(*rule property-empty-interval*) **by** *auto*

23.19 Using additivity of lifted function to encode definedness.

lemma *forall-option*: (∀ *x*. *P x*) ⟷ *P None* ∧ (∀ *x*. *P*(*Some x*))
by (*metis option.nchotomy*)

lemma *exists-option*:
(∃ *x*. *P x*) ⟷ *P None* ∨ (∃ *x*. *P*(*Some x*))
by (*metis option.nchotomy*)

fun *lifted* **where**
lifted (*opp*::'*a*⇒'*a*⇒'*b*) (*Some x*) (*Some y*) = *Some*(*opp x y*) |
lifted opp None - = (*None*::'*b option*) |
lifted opp - *None* = *None*

lemma *lifted-simp-1*[*simp*]: *lifted opp v None* = *None*
apply(*induct v*) **by** *auto*

definition *monoidal opp* ≡ (∀ *x y*. *opp x y* = *opp y x*) ∧
(∀ *x y z*. *opp x* (*opp y z*) = *opp* (*opp x y*) *z*) ∧
(∀ *x*. *opp* (*neutral opp*) *x* = *x*)

lemma *monoidalI*: **assumes** ⋀*x y*. *opp x y* = *opp y x*
⋀*x y z*. *opp x* (*opp y z*) = *opp* (*opp x y*) *z*
⋀*x*. *opp* (*neutral opp*) *x* = *x* **shows** *monoidal opp*
unfolding *monoidal-def* **using** *assms* **by** *fastsimp*

lemma *monoidal-ac*: **assumes** *monoidal opp*
shows *opp* (*neutral opp*) *a* = *a opp a* (*neutral opp*) = *a opp a b* = *opp b a*
opp (*opp a b*) *c* = *opp a* (*opp b c*) *opp a* (*opp b c*) = *opp b* (*opp a c*)
using *assms* **unfolding** *monoidal-def* **apply**− **by** *metis*+

lemma *monoidal-simps*[*simp*]: **assumes** *monoidal opp*
shows *opp* (*neutral opp*) *a* = *a opp a* (*neutral opp*) = *a*
using *monoidal-ac*[*OF assms*] **by** *auto*

lemma *neutral-lifted*[*cong*]: **assumes** *monoidal opp*
shows *neutral* (*lifted opp*) = *Some*(*neutral opp*)
apply(*subst neutral-def*) **apply**(*rule some-equality*) **apply**(*rule*,*induct-tac y*)
prefer *3*
proof− **fix** *x* **assume** ∀ *y*. *lifted opp x y* = *y* ∧ *lifted opp y x* = *y*
thus *x* = *Some* (*neutral opp*) **apply**(*induct x*) **defer**
apply *rule* **apply**(*subst neutral-def*) **apply**(*subst eq-commute*,*rule some-equality*)
apply(*rule*,*erule-tac x*=*Some y* **in** *allE*) **defer apply**(*erule-tac x*=*Some x* **in**
allE) **by** *auto*
qed(*auto simp add*:*monoidal-ac*[*OF assms*])

lemma *monoidal-lifted*[*intro*]: **assumes** *monoidal opp* **shows** *monoidal*(*lifted opp*)
unfolding *monoidal-def forall-option neutral-lifted*[*OF assms*] **using** *monoidal-ac*[*OF assms*] **by** *auto*

definition *support opp f s* = {*x*. *x*∈*s* ∧ *f x* ≠ *neutral opp*}
definition *fold′ opp e s* ≡ (*if finite s then fold opp e s else e*)
definition *iterate opp s f* ≡ *fold′* (λ*x a*. *opp* (*f x*) *a*) (*neutral opp*) (*support opp f s*)

lemma *support-subset*[*intro*]:*support opp f s* ⊆ *s* **unfolding** *support-def* **by** *auto*
lemma *support-empty*[*simp*]:*support opp f* {} = {} **using** *support-subset*[*of opp f* {}] **by** *auto*

lemma *fun-left-comm-monoidal*[*intro*]: **assumes** *monoidal opp* **shows** *fun-left-comm opp*
unfolding *fun-left-comm-def* **using** *monoidal-ac*[*OF assms*] **by** *auto*

lemma *support-clauses*:
⋀*f g s*. *support opp f* {} = {}
⋀*f g s*. *support opp f* (*insert x s*) = (*if f*(*x*) = *neutral opp then support opp f s else insert x* (*support opp f s*))
⋀*f g s*. *support opp f* (*s* − {*x*}) = (*support opp f s*) − {*x*}
⋀*f g s*. *support opp f* (*s* ∪ *t*) = (*support opp f s*) ∪ (*support opp f t*)
⋀*f g s*. *support opp f* (*s* ∩ *t*) = (*support opp f s*) ∩ (*support opp f t*)
⋀*f g s*. *support opp f* (*s* − *t*) = (*support opp f s*) − (*support opp f t*)
⋀*f g s*. *support opp g* (*f ‘ s*) = *f ‘* (*support opp* (*g o f*) *s*)
unfolding *support-def* **by** *auto*

lemma *finite-support*[*intro*]:*finite s* ⟹ *finite* (*support opp f s*)
unfolding *support-def* **by** *auto*

lemma *iterate-empty*[*simp*]:*iterate opp* {} *f* = *neutral opp*
unfolding *iterate-def fold′-def* **by** *auto*

lemma *iterate-insert*[*simp*]: **assumes** *monoidal opp finite s*
shows *iterate opp* (*insert x s*) *f* = (*if x* ∈ *s then iterate opp s f else opp* (*f x*) (*iterate opp s f*))
proof(*cases x*∈*s*) **case** *True* **hence** ∗:*insert x s* = *s* **by** *auto*
show *?thesis* **unfolding** *iterate-def if-P*[*OF True*] ∗ **by** *auto*
next case *False* **note** *x*=*this*
note ∗ = *fun-left-comm.fun-left-comm-apply*[*OF fun-left-comm-monoidal*[*OF assms*(*1*)]]
show *?thesis* **proof**(*cases f x* = *neutral opp*)
case *True* **show** *?thesis* **unfolding** *iterate-def if-not-P*[*OF x*] *support-clauses if-P*[*OF True*]
unfolding *True monoidal-simps*[*OF assms*(*1*)] **by** *auto*
next case *False* **show** *?thesis* **unfolding** *iterate-def fold′-def if-not-P*[*OF x*] *support-clauses if-not-P*[*OF False*]
apply(*subst fun-left-comm.fold-insert*[*OF* ∗ *finite-support*])

using ‹*finite s*› **unfolding** *support-def* **using** *False x* **by** *auto* **qed qed**

lemma *iterate-some*:
assumes *monoidal opp finite s*
shows *iterate* (*lifted opp*) *s* (λ*x*. *Some*(*f x*)) = *Some* (*iterate opp s f*) **using** *assms*(*2*)
proof(*induct s*) **case** *empty* **thus** *?case* **using** *assms* **by** *auto*
next case (*insert x F*) **show** *?case* **apply**(*subst iterate-insert*) **prefer** *3* **apply**(*subst if-not-P*)
defer unfolding *insert*(*3*) *lifted.simps* **apply** *rule* **using** *assms insert* **by** *auto*
qed

23.20 Two key instances of additivity.

lemma *neutral-add*[*simp*]:
neutral op + = (*0*::-::*comm-monoid-add*) **unfolding** *neutral-def*
apply(*rule some-equality*) **defer apply**(*erule-tac x=0* **in** *allE*) **by** *auto*

lemma *operative-content*[*intro*]: *operative* (*op* +) *content*
unfolding *operative-def content-split*[*THEN sym*] *neutral-add* **by** *auto*

lemma *neutral-monoid*: *neutral* ((*op* +)::('*a*::*comm-monoid-add*) ⇒ '*a* ⇒ '*a*) = *0*
by (*rule neutral-add*)

lemma *monoidal-monoid*[*intro*]:
shows *monoidal* ((*op* +)::('*a*::*comm-monoid-add*) ⇒ '*a* ⇒ '*a*)
unfolding *monoidal-def neutral-monoid* **by**(*auto simp add*: *algebra-simps*)

lemma *operative-integral*: **fixes** *f*::*real^'n* ⇒ '*a*::*banach*
shows *operative* (*lifted*(*op* +)) (λ*i*. *if f integrable-on i then Some*(*integral i f*) *else None*)
unfolding *operative-def* **unfolding** *neutral-lifted*[*OF monoidal-monoid*] *neutral-add*
apply(*rule*,*rule*,*rule*,*rule*) **defer apply**(*rule allI*)+
proof− **fix** *a b c k* **show** (*if f integrable-on* {*a*..*b*} *then Some* (*integral* {*a*..*b*} *f*) *else None*) =
lifted op + (*if f integrable-on* {*a*..*b*} ∩ {*x*. *x* $ *k* ≤ *c*} *then Some* (*integral* ({*a*..*b*} ∩ {*x*. *x* $ *k* ≤ *c*}) *f*) *else None*)
(*if f integrable-on* {*a*..*b*} ∩ {*x*. *c* ≤ *x* $ *k*} *then Some* (*integral* ({*a*..*b*} ∩ {*x*. *c* ≤ *x* $ *k*}) *f*) *else None*)
proof(*cases f integrable-on* {*a*..*b*})
case *True* **show** *?thesis* **unfolding** *if-P*[*OF True*]
unfolding *if-P*[*OF integrable-split*(*1*)[*OF True*]] *if-P*[*OF integrable-split*(*2*)[*OF True*]]
unfolding *lifted.simps option.inject* **apply**(*rule integral-unique*) **apply**(*rule has-integral-split*)
apply(*rule-tac*[!] *integrable-integral integrable-split*)+ **using** *True* **by** *assumption*+
next case *False* **have** (¬ (*f integrable-on* {*a*..*b*} ∩ {*x*. *x* $ *k* ≤ *c*})) ∨ (¬ (*f integrable-on* {*a*..*b*} ∩ {*x*. *c* ≤ *x* $ *k*}))

proof(*rule ccontr*) **case** *goal1* **hence** *f integrable-on* {*a*..*b*} **apply**− **unfolding** *integrable-on-def*
apply(*rule-tac x*=*integral* ({*a*..*b*} ∩ {*x*. *x* $ *k* ≤ *c*}) *f* + *integral* ({*a*..*b*} ∩ {*x*. *x* $ *k* ≥ *c*}) *f* **in** *exI*)
apply(*rule has-integral-split*) **apply**(*rule-tac*[!] *integrable-integral*) **by** *auto*
thus *False* **using** *False* **by** *auto*
qed thus *?thesis* **using** *False* **by** *auto*
qed next
fix *a b* **assume** *as*:*content* {*a*..*b*::*real*^'*n*} = *0*
thus (*if f integrable-on* {*a*..*b*} *then Some* (*integral* {*a*..*b*} *f*) *else None*) = *Some 0*
unfolding *if-P*[*OF integrable-on-null*[*OF as*]] **using** *has-integral-null-eq*[*OF as*] **by** *auto* **qed**

23.21 Points of division of a partition.

definition *division-points* (*k*::(*real*^'*n*) *set*) *d* =
{(*j*,*x*). (*interval-lowerbound k*)$*j* < *x* ∧ *x* < (*interval-upperbound k*)$*j* ∧
(∃ *i*∈*d*. (*interval-lowerbound i*)$*j* = *x* ∨ (*interval-upperbound i*)$*j* = *x*)}

lemma *division-points-finite*: **assumes** *d division-of i*
shows *finite* (*division-points i d*)
proof− **note** *assm* = *division-ofD*[*OF assms*]
let *?M* = λ*j*. {(*j*,*x*)|*x*. (*interval-lowerbound i*)$*j* < *x* ∧ *x* < (*interval-upperbound i*)$*j* ∧
(∃ *i*∈*d*. (*interval-lowerbound i*)$*j* = *x* ∨ (*interval-upperbound i*)$*j* = *x*)}
have ∗:*division-points i d* = ⋃(*?M* ' *UNIV*)
unfolding *division-points-def* **by** *auto*
show *?thesis* **unfolding** ∗ **using** *assm* **by** *auto* **qed**

lemma *division-points-subset*:
assumes *d division-of* {*a*..*b*} ∀ *i*. *a*$*i* < *b*$*i* *a*$*k* < *c* *c* < *b*$*k*
shows *division-points* ({*a*..*b*} ∩ {*x*. *x*$*k* ≤ *c*}) {*l* ∩ {*x*. *x*$*k* ≤ *c*} | *l* . *l* ∈ *d* ∧ ~(*l* ∩ {*x*. *x*$*k* ≤ *c*} = {})}
⊆ *division-points* ({*a*..*b*}) *d* (**is** *?t1*) **and**
division-points ({*a*..*b*} ∩ {*x*. *x*$*k* ≥ *c*}) {*l* ∩ {*x*. *x*$*k* ≥ *c*} | *l* . *l* ∈ *d* ∧ ~(*l* ∩ {*x*. *x*$*k* ≥ *c*} = {})}
⊆ *division-points* ({*a*..*b*}) *d* (**is** *?t2*)
proof− **note** *assm* = *division-ofD*[*OF assms*(*1*)]
have ∗:∀ *i*. *a*$*i* ≤ *b*$*i* ∀ *i*. *a*$*i* ≤ (χ *i*. *if i* = *k then min* (*b* $ *k*) *c else b* $ *i*) $ *i*
∀ *i*. (χ *i*. *if i* = *k then max* (*a* $ *k*) *c else a* $ *i*) $ *i* ≤ *b*$*i* *min* (*b* $ *k*) *c* = *c* *max* (*a* $ *k*) *c* = *c*
using *assms* **using** *less-imp-le* **by** *auto*
show *?t1* **unfolding** *division-points-def interval-split*[*of a b*]
unfolding *interval-bounds*[*OF* ∗(*1*)] *interval-bounds*[*OF* ∗(*2*)] *interval-bounds*[*OF* ∗(*3*)] *Cart-lambda-beta* **unfolding** ∗
unfolding *subset-eq* **apply**(*rule*) **unfolding** *mem-Collect-eq split-beta* **apply**(*erule bexE conjE*)+ **unfolding** *mem-Collect-eq* **apply**(*erule exE conjE*)+
proof− **fix** *i l x* **assume** *as*:*a* $ *fst x* < *snd x snd x* < (*if fst x* = *k then c else*

b $ fst x)
interval-lowerbound i $ fst x = snd x ∨ interval-upperbound i $ fst x = snd x
i = l ∩ {x. x $ k ≤ c} l ∈ d l ∩ {x. x $ k ≤ c} ≠ {}
from *assm*(*4*)[*OF this*(*5*)] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *l*=*this*
have ∗:∀ *i. u $ i ≤* (*χ i. if i = k then min* (*v $ k*) *c else v $ i*) *$ i* **using** *as*(*6*)
unfolding *l interval-split interval-ne-empty as* **.**
have ∗∗:∀ *i. u$i ≤ v$i* **using** *l* **using** *as*(*6*) **unfolding** *interval-ne-empty*[*THEN sym*] **by** *auto*
show *a $ fst x < snd x ∧ snd x < b $ fst x ∧* (∃ *i*∈*d. interval-lowerbound i $ fst x = snd x ∨ interval-upperbound i $ fst x = snd x*)
using *as*(*1*−*3*,*5*) **unfolding** *l interval-split interval-ne-empty as interval-bounds*[*OF* ∗] *Cart-lambda-beta* **apply**−
apply(*rule*,*assumption*,*rule*) **defer apply**(*rule-tac x*={*u..v*} **in** *bexI*) **unfolding** *interval-bounds*[*OF* ∗∗]
apply(*case-tac*[!] *fst x = k*) **using** *assms* **by** *auto*
qed
show *?t2* **unfolding** *division-points-def interval-split*[*of a b*]
unfolding *interval-bounds*[*OF* ∗(*1*)] *interval-bounds*[*OF* ∗(*2*)] *interval-bounds*[*OF* ∗(*3*)] *Cart-lambda-beta* **unfolding** ∗
unfolding *subset-eq* **apply**(*rule*) **unfolding** *mem-Collect-eq split-beta* **apply**(*erule bexE conjE*)+ **unfolding** *mem-Collect-eq* **apply**(*erule exE conjE*)+
proof− **fix** *i l x* **assume** *as*:(*if fst x = k then c else a $ fst x*) *< snd x snd x < b $ fst x interval-lowerbound i $ fst x = snd x ∨ interval-upperbound i $ fst x = snd x*
i = l ∩ {x. c ≤ x $ k} l ∈ d l ∩ {x. c ≤ x $ k} ≠ {}
from *assm*(*4*)[*OF this*(*5*)] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *l*=*this*
have ∗:∀ *i.* (*χ i. if i = k then max* (*u $ k*) *c else u $ i*) *$ i ≤ v $ i* **using** *as*(*6*) **unfolding** *l interval-split interval-ne-empty as* **.**
have ∗∗:∀ *i. u$i ≤ v$i* **using** *l* **using** *as*(*6*) **unfolding** *interval-ne-empty*[*THEN sym*] **by** *auto*
show *a $ fst x < snd x ∧ snd x < b $ fst x ∧* (∃ *i*∈*d. interval-lowerbound i $ fst x = snd x ∨ interval-upperbound i $ fst x = snd x*)
using *as*(*1*−*3*,*5*) **unfolding** *l interval-split interval-ne-empty as interval-bounds*[*OF* ∗] *Cart-lambda-beta* **apply**−
apply *rule* **defer apply**(*rule*,*assumption*) **apply**(*rule-tac x*={*u..v*} **in** *bexI*) **unfolding** *interval-bounds*[*OF* ∗∗]
apply(*case-tac*[!] *fst x = k*) **using** *assms* **by** *auto* **qed qed**

lemma *division-points-psubset*:
assumes *d division-of* {*a..b*} ∀ *i. a$i < b$i a$k < c c < b$k*
l ∈ d interval-lowerbound l$k = c ∨ interval-upperbound l$k = c
shows *division-points* ({*a..b*} ∩ {*x. x$k ≤ c*}) {*l ∩* {*x. x$k ≤ c*} | *l. l*∈*d ∧ l ∩* {*x. x$k ≤ c*} ≠ {}} ⊂ *division-points* ({*a..b*}) *d* (**is** *?D1* ⊂ *?D*)
division-points ({*a..b*} ∩ {*x. x$k ≥ c*}) {*l ∩* {*x. x$k ≥ c*} | *l. l*∈*d ∧ l ∩* {*x. x$k ≥ c*} ≠ {}} ⊂ *division-points* ({*a..b*}) *d* (**is** *?D2* ⊂ *?D*)
proof− **have** *ab*:∀ *i. a$i ≤ b$i* **using** *assms*(*2*) **by**(*auto intro*!:*less-imp-le*)
guess *u v* **using** *division-ofD*(*4*)[*OF assms*(*1*,*5*)] **apply**−**by**(*erule exE*)+ **note** *l*=*this*
have *uv*:∀ *i. u$i ≤ v$i* ∀ *i. a$i ≤ u$i ∧ v$i ≤ b$i* **using** *division-ofD*(*2*,*2*,*3*)[*OF*

assms(*1*,*5*)] **unfolding** *l interval-ne-empty*
unfolding *subset-eq* **apply**− **defer apply**(*erule-tac x*=*u* **in** *ballE*, *erule-tac x*=*v* **in** *ballE*) **unfolding** *mem-interval* **by** *auto*
have ∗:*interval-upperbound* ({*a*..*b*} ∩ {*x*. *x* $ *k* ≤ *interval-upperbound l* $ *k*}) $ *k* = *interval-upperbound l* $ *k*
interval-upperbound ({*a*..*b*} ∩ {*x*. *x* $ *k* ≤ *interval-lowerbound l* $ *k*}) $ *k* = *interval-lowerbound l* $ *k*
unfolding *interval-split* **apply**(*subst interval-bounds*) **prefer** *3* **apply**(*subst interval-bounds*)
unfolding *l interval-bounds*[*OF uv*(*1*)] **using** *uv*[*rule-format*,*of k*] *ab* **by** *auto*
have ∃*x*. *x* ∈ *?D* − *?D1* **using** *assms*(*2*−) **apply**−**apply**(*erule disjE*)
apply(*rule-tac x*=(*k*,(*interval-lowerbound l*)$*k*) **in** *exI*) **defer**
apply(*rule-tac x*=(*k*,(*interval-upperbound l*)$*k*) **in** *exI*)
unfolding *division-points-def* **unfolding** *interval-bounds*[*OF ab*]
apply *auto* **unfolding** ∗ **by** *auto*
thus *?D1* ⊂ *?D* **apply**−**apply**(*rule*,*rule division-points-subset*[*OF assms*(*1*−*4*)])
by *auto*

have ∗:*interval-lowerbound* ({*a*..*b*} ∩ {*x*. *x* $ *k* ≥ *interval-lowerbound l* $ *k*}) $ *k* = *interval-lowerbound l* $ *k*
interval-lowerbound ({*a*..*b*} ∩ {*x*. *x* $ *k* ≥ *interval-upperbound l* $ *k*}) $ *k* = *interval-upperbound l* $ *k*
unfolding *interval-split* **apply**(*subst interval-bounds*) **prefer** *3* **apply**(*subst interval-bounds*)
unfolding *l interval-bounds*[*OF uv*(*1*)] **using** *uv*[*rule-format*,*of k*] *ab* **by** *auto*
have ∃*x*. *x* ∈ *?D* − *?D2* **using** *assms*(*2*−) **apply**−**apply**(*erule disjE*)
apply(*rule-tac x*=(*k*,(*interval-lowerbound l*)$*k*) **in** *exI*) **defer**
apply(*rule-tac x*=(*k*,(*interval-upperbound l*)$*k*) **in** *exI*)
unfolding *division-points-def* **unfolding** *interval-bounds*[*OF ab*]
apply *auto* **unfolding** ∗ **by** *auto*
thus *?D2* ⊂ *?D* **apply**−**apply**(*rule*,*rule division-points-subset*[*OF assms*(*1*−*4*)])
by *auto* **qed**

23.22 Preservation by divisions and tagged divisions.

lemma *support-support*[*simp*]:*support opp f* (*support opp f s*) = *support opp f s*
unfolding *support-def* **by** *auto*

lemma *iterate-support*[*simp*]: *iterate opp* (*support opp f s*) *f* = *iterate opp s f*
unfolding *iterate-def support-support* **by** *auto*

lemma *iterate-expand-cases*:
iterate opp s f = (*if finite*(*support opp f s*) *then iterate opp* (*support opp f s*) *f else neutral opp*)
apply(*cases*) **apply**(*subst if-P*,*assumption*) **unfolding** *iterate-def support-support fold′-def* **by** *auto*

lemma *iterate-image*: **assumes** *monoidal opp inj-on f s*
shows *iterate opp* (*f* ‘ *s*) *g* = *iterate opp s* (*g* ∘ *f*)

proof− **have** ∗:⋀*s*. *finite s* ⟹ ∀ *x*∈*s*. ∀ *y*∈*s*. *f x* = *f y* ⟶ *x* = *y* ⟹
iterate opp (*f* ‘ *s*) *g* = *iterate opp s* (*g* ∘ *f*)
proof− **case** *goal1* **show** *?case* **using** *goal1*
proof(*induct s*) **case** *empty* **thus** *?case* **using** *assms*(*1*) **by** *auto*
next case (*insert x s*) **show** *?case* **unfolding** *iterate-insert*[*OF assms*(*1*) *insert*(*1*)]
unfolding *if-not-P*[*OF insert*(*2*)] **apply**(*subst insert*(*3*)[*THEN sym*])
unfolding *image-insert* **defer apply**(*subst iterate-insert*[*OF assms*(*1*)])
apply(*rule finite-imageI insert*)+ **apply**(*subst if-not-P*)
unfolding *image-iff o-def* **using** *insert*(*2*,*4*) **by** *auto*
qed qed
show *?thesis*
apply(*cases finite* (*support opp g* (*f* ‘ *s*)))
apply(*subst* (*1*) *iterate-support*[*THEN sym*],*subst* (*2*) *iterate-support*[*THEN sym*])
unfolding *support-clauses* **apply**(*rule* ∗)**apply**(*rule finite-imageD*,*assumption*) **unfolding** *inj-on-def*[*symmetric*]
apply(*rule subset-inj-on*[*OF assms*(*2*) *support-subset*])+
apply(*subst iterate-expand-cases*) **unfolding** *support-clauses* **apply**(*simp only*: *if-False*)
apply(*subst iterate-expand-cases*) **apply**(*subst if-not-P*) **by** *auto* **qed**

lemma *iterate-nonzero-image-lemma*:
assumes *monoidal opp finite s g*(*a*) = *neutral opp*
∀ *x*∈*s*. ∀ *y*∈*s*. *f x* = *f y* ∧ *x* ≠ *y* ⟶ *g*(*f x*) = *neutral opp*
shows *iterate opp* {*f x* | *x*. *x* ∈ *s* ∧ *f x* ≠ *a*} *g* = *iterate opp s* (*g* ∘ *f*)
proof− **have** ∗:{*f x* |*x*. *x* ∈ *s* ∧ ~(*f x* = *a*)} = *f* ‘ {*x*. *x* ∈ *s* ∧ ~(*f x* = *a*)} **by** *auto*
have ∗∗:*support opp* (*g* ∘ *f*) {*x* ∈ *s*. *f x* ≠ *a*} = *support opp* (*g* ∘ *f*) *s*
unfolding *support-def* **using** *assms*(*3*) **by** *auto*
show *?thesis* **unfolding** ∗
apply(*subst iterate-support*[*THEN sym*]) **unfolding** *support-clauses*
apply(*subst iterate-image*[*OF assms*(*1*)]) **defer**
apply(*subst*(*2*) *iterate-support*[*THEN sym*]) **apply**(*subst* ∗∗)
unfolding *inj-on-def* **using** *assms*(*3*,*4*) **unfolding** *support-def* **by** *auto* **qed**

lemma *iterate-eq-neutral*:
assumes *monoidal opp* ∀ *x* ∈ *s*. (*f*(*x*) = *neutral opp*)
shows (*iterate opp s f* = *neutral opp*)
proof− **have** ∗:*support opp f s* = {} **unfolding** *support-def* **using** *assms*(*2*) **by** *auto*
show *?thesis* **apply**(*subst iterate-support*[*THEN sym*])
unfolding ∗ **using** *assms*(*1*) **by** *auto* **qed**

lemma *iterate-op*: **assumes** *monoidal opp finite s*
shows *iterate opp s* (λ*x*. *opp* (*f x*) (*g x*)) = *opp* (*iterate opp s f*) (*iterate opp s g*) **using** *assms*(*2*)

proof(*induct s*) **case** *empty* **thus** *?case* **unfolding** *iterate-insert*[*OF assms*(*1*)]
using *assms*(*1*) **by** *auto*
next case (*insert x F*) **show** *?case* **unfolding** *iterate-insert*[*OF assms*(*1*) *in-sert*(*1*)] *if-not-P*[*OF insert*(*2*)] *insert*(*3*)
unfolding *monoidal-ac*[*OF assms*(*1*)] **by**(*rule refl*) **qed**

lemma *iterate-eq*: **assumes** *monoidal opp* ⋀*x*. *x* ∈ *s* ⟹ *f x* = *g x*
shows *iterate opp s f* = *iterate opp s g*
proof− **have** ∗:*support opp g s* = *support opp f s*
unfolding *support-def* **using** *assms*(*2*) **by** *auto*
show *?thesis*
proof(*cases finite* (*support opp f s*))
case *False* **thus** *?thesis* **apply**(*subst iterate-expand-cases*,*subst*(*2*) *iterate-expand-cases*)
unfolding ∗ **by** *auto*
next def *su* ≡ *support opp f s*
case *True* **note** *support-subset*[*of opp f s*]
thus *?thesis* **apply**− **apply**(*subst iterate-support*[*THEN sym*],*subst*(*2*) *iterate-support*[*THEN sym*]) **unfolding** ∗ **using** *True*
unfolding *su-def*[*symmetric*]
proof(*induct su*) **case** *empty* **show** *?case* **by** *auto*
next case (*insert x s*) **show** *?case* **unfolding** *iterate-insert*[*OF assms*(*1*) *insert*(*1*)]
unfolding *if-not-P*[*OF insert*(*2*)] **apply**(*subst insert*(*3*))
defer apply(*subst assms*(*2*)[*of x*]) **using** *insert* **by** *auto* **qed qed qed**

lemma *nonempty-witness*: **assumes** *s* ≠ {} **obtains** *x* **where** *x* ∈ *s* **using** *assms*
by *auto*

lemma *operative-division*: **fixes** *f*::(*real*^′*n*) *set* ⇒ ′*a*
assumes *monoidal opp operative opp f d division-of* {*a*..*b*}
shows *iterate opp d f* = *f* {*a*..*b*}
proof− **def** *C* ≡ *card* (*division-points* {*a*..*b*} *d*) **thus** *?thesis* **using** *assms*
proof(*induct C arbitrary*:*a b d rule*:*full-nat-induct*)
case *goal1*
{ presume ∗:*content* {*a*..*b*} ≠ *0* ⟹ *?case*
thus *?case* **apply**−**apply**(*cases*) **defer apply** *assumption*
proof− **assume** *as*:*content* {*a*..*b*} = *0*
show *?case* **unfolding** *operativeD*(*1*)[*OF assms*(*2*) *as*] **apply**(*rule iterate-eq-neutral*[*OF goal1*(*2*)])
proof fix *x* **assume** *x*:*x*∈*d*
then guess *u v* **apply**(*drule-tac division-ofD*(*4*)[*OF goal1*(*4*)]) **by**(*erule exE*)+
thus *f x* = *neutral opp* **using** *division-of-content-0*[*OF as goal1*(*4*)]
using *operativeD*(*1*)[*OF assms*(*2*)] *x* **by** *auto*
qed qed }
assume *content* {*a*..*b*} ≠ *0* **note** *ab* = *this*[*unfolded content-lt-nz*[*THEN sym*] *content-pos-lt-eq*]
hence *ab*′:∀ *i*. *a*$*i* ≤ *b*$*i* **by** (*auto intro*!: *less-imp-le*) **show** *?case*
proof(*cases division-points* {*a*..*b*} *d* = {})

case *True* **have** *d'*:$\forall\, i{\in}d.\ \exists\, u\ v.\ i = \{u..v\} \wedge$
$(\forall\, j.\ u\$j = a\$j \wedge v\$j = a\$j \vee u\$j = b\$j \wedge v\$j = b\$j \vee u\$j = a\$j \wedge v\$j = b\$j)$
unfolding *forall-in-division*[*OF goal1*(*4*)] **apply**(*rule*,*rule*,*rule*)
apply(*rule-tac x*=*a* **in** *exI*,*rule-tac x*=*b* **in** *exI*) **apply**(*rule*,*rule refl*) **apply**(*rule*)
proof− **fix** *u v j* **assume** *as*:$\{u..v\} \in d$ **note** *division-ofD*(*3*)[*OF goal1*(*4*) *this*]
hence *uv*:$\forall\, i.\ u\$i \le v\$i\ u\$j \le v\$j$ **unfolding** *interval-ne-empty* **by** *auto*
have ∗:$\bigwedge p\ r\ Q.\ p \vee r \vee (\forall\, x{\in}d.\ Q\ x) \Longrightarrow p \vee r \vee (Q\ \{u..v\})$ **using** *as* **by** *auto*
have $(j,\ u\$j) \notin$ *division-points* $\{a..b\}\ d$
$(j,\ v\$j) \notin$ *division-points* $\{a..b\}\ d$ **using** *True* **by** *auto*
note *this*[*unfolded de-Morgan-conj division-points-def mem-Collect-eq split-conv interval-bounds*[*OF ab'*] *bex-simps*]
note ∗[*OF this*(*1*)] ∗[*OF this*(*2*)] **note** *this*[*unfolded interval-bounds*[*OF uv*(*1*)]]
moreover have $a\$j \le u\$j\ v\$j \le b\$j$ **using** *division-ofD*(*2*,*2*,*3*)[*OF goal1*(*4*) *as*]
unfolding *subset-eq* **apply**− **apply**(*erule-tac x*=*u* **in** *ballE*,*erule-tac*[*3*] *x*=*v* **in** *ballE*)
unfolding *interval-ne-empty mem-interval* **by** *auto*
ultimately show $u\$j = a\$j \wedge v\$j = a\$j \vee u\$j = b\$j \wedge v\$j = b\$j \vee u\$j = a\$j \wedge v\$j = b\$j$
unfolding *not-less de-Morgan-disj* **using** *ab*[*rule-format*,*of j*] *uv*(*2*) **by** *auto*
qed have $(1/2) *_R (a+b) \in \{a..b\}$ **unfolding** *mem-interval* **using** *ab* **by**(*auto intro*!:*less-imp-le*)
note *this*[*unfolded division-ofD*(*6*)[*OF goal1*(*4*),*THEN sym*] *Union-iff*]
then guess *i* **.. note** *i*=*this* **guess** *u v* **using** *d'*[*rule-format*,*OF i*(*1*)] **apply**−**by**(*erule exE conjE*)+ **note** *uv*=*this*
have $\{a..b\} \in d$
proof− **{ presume** $i = \{a..b\}$ **thus** *?thesis* **using** *i* **by** *auto* **}**
{ presume $u = a\ v = b$ **thus** $i = \{a..b\}$ **using** *uv* **by** *auto* **}**
show $u = a\ v = b$ **unfolding** *Cart-eq*
proof(*rule-tac*[!] *allI*) **fix** *j* **note** *i*(*2*)[*unfolded uv mem-interval*,*rule-format*,*of j*]
thus $u\ \$\ j = a\ \$\ j\ v\ \$\ j = b\ \$\ j$ **using** *uv*(*2*)[*rule-format*,*of j*] **by** *auto*
qed qed
hence ∗:$d = insert\ \{a..b\}\ (d - \{\{a..b\}\})$ **by** *auto*
have *iterate opp* $(d - \{\{a..b\}\})\ f = neutral\ opp$ **apply**(*rule iterate-eq-neutral*[*OF goal1*(*2*)])
proof fix *x* **assume** *x*:$x \in d - \{\{a..b\}\}$ **hence** $x{\in}d$ **by** *auto* **note** *d'*[*rule-format*,*OF this*]
then guess *u v* **apply**−**by**(*erule exE conjE*)+ **note** *uv*=*this*
have $u{\neq}a \vee v{\neq}b$ **using** *x*[*unfolded uv*] **by** *auto*
then obtain *j* **where** $u\$j \neq a\$j \vee v\$j \neq b\$j$ **unfolding** *Cart-eq* **by** *auto*
hence $u\$j = v\$j$ **using** *uv*(*2*)[*rule-format*,*of j*] **by** *auto*
hence *content* $\{u..v\} = 0$ **unfolding** *content-eq-0* **apply**(*rule-tac x*=*j* **in**

exI) **by** *auto*
thus $f\,x = $ *neutral opp* **unfolding** *uv*(*1*) **by**(*rule operativeD*(*1*)[*OF goal1*(*3*)])
qed thus *iterate opp d f* = *f* {*a*..*b*} **apply**−**apply**(*subst* ∗)
apply(*subst iterate-insert*[*OF goal1*(*2*)]) **using** *goal1*(*2*,*4*) **by** *auto*
next case *False* **hence** ∃ *x*. *x*∈*division-points* {*a*..*b*} *d* **by** *auto*
then guess *k c* **unfolding** *split-paired-Ex* **apply**− **unfolding** *division-points-def mem-Collect-eq split-conv*
by(*erule exE conjE*)+ **note** *kc*=*this*[*unfolded interval-bounds*[*OF ab'*]]
from *this*(*3*) **guess** *j* **..** **note** *j*=*this*
def *d1* ≡ {*l* ∩ {*x*. *x*$*k* ≤ *c*} | *l*. *l* ∈ *d* ∧ *l* ∩ {*x*. *x*$*k* ≤ *c*} ≠ {}}
def *d2* ≡ {*l* ∩ {*x*. *x*$*k* ≥ *c*} | *l*. *l* ∈ *d* ∧ *l* ∩ {*x*. *x*$*k* ≥ *c*} ≠ {}}
def *cb* ≡ (χ *i*. *if i* = *k then c else b*$*i*) **and** *ca* ≡ (χ *i*. *if i* = *k then c else a*$*i*)
note *division-points-psubset*[*OF goal1*(*4*) *ab kc*(*1*−*2*) *j*]
note *psubset-card-mono*[*OF* - *this*(*1*)] *psubset-card-mono*[*OF* - *this*(*2*)]
hence ∗:(*iterate opp d1 f*) = *f* ({*a*..*b*} ∩ {*x*. *x*$*k* ≤ *c*}) (*iterate opp d2 f*) = *f* ({*a*..*b*} ∩ {*x*. *x*$*k* ≥ *c*})
apply− **unfolding** *interval-split* **apply**(*rule-tac*[!] *goal1*(*1*)[*rule-format*])
using *division-split*[*OF goal1*(*4*), **where** *k*=*k* **and** *c*=*c*]
unfolding *interval-split d1-def*[*symmetric*] *d2-def*[*symmetric*] **unfolding** *goal1*(*2*) *Suc-le-mono*
using *goal1*(*2*−*3*) **using** *division-points-finite*[*OF goal1*(*4*)] **by** *auto*
have *f* {*a*..*b*} = *opp* (*iterate opp d1 f*) (*iterate opp d2 f*) (**is** - = *?prev*)
unfolding ∗ **apply**(*rule operativeD*(*2*)) **using** *goal1*(*3*) **.**
also have *iterate opp d1 f* = *iterate opp d* (λ*l*. *f*(*l* ∩ {*x*. *x*$*k* ≤ *c*}))
unfolding *d1-def* **apply**(*rule iterate-nonzero-image-lemma*[*unfolded o-def*])
unfolding *empty-as-interval* **apply**(*rule goal1 division-of-finite operativeD*[*OF goal1*(*3*)])+
unfolding *empty-as-interval*[*THEN sym*] **apply**(*rule content-empty*)
proof(*rule*,*rule*,*rule*,*erule conjE*) **fix** *l y* **assume** *as*:*l* ∈ *d y* ∈ *d l* ∩ {*x*. *x* $ *k* ≤ *c*} = *y* ∩ {*x*. *x* $ *k* ≤ *c*} *l* ≠ *y*
from *division-ofD*(*4*)[*OF goal1*(*4*) *this*(*1*)] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *l*=*this*
show *f* (*l* ∩ {*x*. *x* $ *k* ≤ *c*}) = *neutral opp* **unfolding** *l interval-split*
apply(*rule operativeD*(*1*) *goal1*)+ **unfolding** *interval-split*[*THEN sym*] **apply**(*rule division-split-left-inj*)
apply(*rule goal1*) **unfolding** *l*[*THEN sym*] **apply**(*rule as*(*1*),*rule as*(*2*)) **by**(*rule as*)+
qed also have *iterate opp d2 f* = *iterate opp d* (λ*l*. *f*(*l* ∩ {*x*. *x*$*k* ≥ *c*}))
unfolding *d2-def* **apply**(*rule iterate-nonzero-image-lemma*[*unfolded o-def*])
unfolding *empty-as-interval* **apply**(*rule goal1 division-of-finite operativeD*[*OF goal1*(*3*)])+
unfolding *empty-as-interval*[*THEN sym*] **apply**(*rule content-empty*)
proof(*rule*,*rule*,*rule*,*erule conjE*) **fix** *l y* **assume** *as*:*l* ∈ *d y* ∈ *d l* ∩ {*x*. *c* ≤ *x* $ *k*} = *y* ∩ {*x*. *c* ≤ *x* $ *k*} *l* ≠ *y*
from *division-ofD*(*4*)[*OF goal1*(*4*) *this*(*1*)] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *l*=*this*
show *f* (*l* ∩ {*x*. *x* $ *k* ≥ *c*}) = *neutral opp* **unfolding** *l interval-split*
apply(*rule operativeD*(*1*) *goal1*)+ **unfolding** *interval-split*[*THEN sym*]

apply(*rule division-split-right-inj*)
apply(*rule goal1*) **unfolding** *l*[*THEN sym*] **apply**(*rule as*(*1*),*rule as*(*2*)) **by**(*rule as*)+
qed also have ∗:∀ *x*∈*d*. *f x* = *opp* (*f* (*x* ∩ {*x*. *x* $ *k* ≤ *c*})) (*f* (*x* ∩ {*x*. *c* ≤ *x* $ *k*}))
unfolding *forall-in-division*[*OF goal1*(*4*)] **apply**(*rule*,*rule*,*rule*,*rule operativeD*(*2*)) **using** *goal1*(*3*) **.**
have *opp* (*iterate opp d* (λ*l*. *f* (*l* ∩ {*x*. *x* $ *k* ≤ *c*}))) (*iterate opp d* (λ*l*. *f* (*l* ∩ {*x*. *c* ≤ *x* $ *k*})))
= *iterate opp d f* **apply**(*subst*(*3*) *iterate-eq*[*OF* - ∗[*rule-format*]]) **prefer** *3*
apply(*rule iterate-op*[*THEN sym*]) **using** *goal1* **by** *auto*
finally show *?thesis* **by** *auto*
qed qed qed

lemma *iterate-image-nonzero*: **assumes** *monoidal opp*
finite s ∀ *x*∈*s*. ∀ *y*∈*s*. ~(*x* = *y*) ∧ *f x* = *f y* ⟶ *g*(*f x*) = *neutral opp*
shows *iterate opp* (*f* ‘ *s*) *g* = *iterate opp s* (*g* ∘ *f*) **using** *assms*
proof(*induct rule*:*finite-subset-induct*[*OF assms*(*2*) *subset-refl*])
case *goal1* **show** *?case* **using** *assms*(*1*) **by** *auto*
next case *goal2* **have** ∗:⋀*x y*. *y* = *neutral opp* ⟹ *x* = *opp y x* **using** *assms*(*1*) **by** *auto*
show *?case* **unfolding** *image-insert* **apply**(*subst iterate-insert*[*OF assms*(*1*)])
apply(*rule finite-imageI goal2*)+
apply(*cases f a* ∈ *f* ‘ *F*) **unfolding** *if-P if-not-P* **apply**(*subst goal2*(*4*)[*OF assms*(*1*) *goal2*(*1*)]) **defer**
apply(*subst iterate-insert*[*OF assms*(*1*) *goal2*(*1*)]) **defer**
apply(*subst iterate-insert*[*OF assms*(*1*) *goal2*(*1*)])
unfolding *if-not-P*[*OF goal2*(*3*)] **defer unfolding** *image-iff* **defer apply**(*erule bexE*)
apply(*rule* ∗) **unfolding** *o-def* **apply**(*rule-tac y*=*x* **in** *goal2*(*7*)[*rule-format*])
using *goal2* **unfolding** *o-def* **by** *auto* **qed**

lemma *operative-tagged-division*: **assumes** *monoidal opp operative opp f d tagged-division-of* {*a*..*b*}
shows *iterate*(*opp*) *d* (λ(*x*,*l*). *f l*) = *f* {*a*..*b*}
proof− **have** ∗:(λ(*x*,*l*). *f l*) = (*f o snd*) **unfolding** *o-def* **by**(*rule*,*auto*) **note** *assm* = *tagged-division-ofD*[*OF assms*(*3*)]
have *iterate*(*opp*) *d* (λ(*x*,*l*). *f l*) = *iterate opp* (*snd* ‘ *d*) *f* **unfolding** ∗
apply(*rule iterate-image-nonzero*[*THEN sym*,*OF assms*(*1*)]) **apply**(*rule tagged-division-of-finite assms*)+
unfolding *Ball-def split-paired-All snd-conv* **apply**(*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*erule conjE*)
proof− **fix** *a b aa ba* **assume** *as*:(*a*, *b*) ∈ *d* (*aa*, *ba*) ∈ *d* (*a*, *b*) ≠ (*aa*, *ba*) *b* = *ba*
guess *u v* **using** *assm*(*4*)[*OF as*(*1*)] **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
show *f b* = *neutral opp* **unfolding** *uv* **apply**(*rule operativeD*(*1*)[*OF assms*(*2*)])
unfolding *content-eq-0-interior* **using** *tagged-division-ofD*(*5*)[*OF assms*(*3*) *as*(*1*−*3*)]
unfolding *as*(*4*)[*THEN sym*] *uv* **by** *auto*

qed also have ... = *f* {*a*..*b*}
using *operative-division*[*OF assms*(*1*−*2*) *division-of-tagged-division*[*OF assms*(*3*)]]
.
finally show *?thesis* **. qed**

23.23 Additivity of content.

lemma *setsum-iterate*:**assumes** *finite s* **shows** *setsum f s* = *iterate op* + *s f*
proof− **have** ∗:*setsum f s* = *setsum f* (*support op* + *f s*)
apply(*rule setsum-mono-zero-right*)
unfolding *support-def neutral-monoid* **using** *assms* **by** *auto*
thus *?thesis* **unfolding** ∗ *setsum-def iterate-def fold-image-def fold'-def*
unfolding *neutral-monoid* **. qed**

lemma *additive-content-division*: **assumes** *d division-of* {*a*..*b*}
shows *setsum content d* = *content*({*a*..*b*})
unfolding *operative-division*[*OF monoidal-monoid operative-content assms*,*THEN sym*]
apply(*subst setsum-iterate*) **using** *assms* **by** *auto*

lemma *additive-content-tagged-division*:
assumes *d tagged-division-of* {*a*..*b*}
shows *setsum* (λ(*x*,*l*). *content l*) *d* = *content*({*a*..*b*})
unfolding *operative-tagged-division*[*OF monoidal-monoid operative-content assms*,*THEN sym*]
apply(*subst setsum-iterate*) **using** *assms* **by** *auto*

23.24 Finally, the integral of a constant

lemma *has-integral-const*[*intro*]:
((λ*x*. *c*) *has-integral* (*content*({*a*..*b*::*real*^'*n*}) ∗*R* *c*)) ({*a*..*b*})
unfolding *has-integral* **apply**(*rule*,*rule*,*rule-tac x*=λ*x*. *ball x 1* **in** *exI*)
apply(*rule*,*rule gauge-trivial*)**apply**(*rule*,*rule*,*erule conjE*)
unfolding *split-def* **apply**(*subst scaleR-left.setsum*[*THEN sym*, *unfolded o-def*])
defer apply(*subst additive-content-tagged-division*[*unfolded split-def*]) **apply** *assumption* **by** *auto*

23.25 Bounds on the norm of Riemann sums and the integral itself.

lemma *dsum-bound*: **assumes** *p division-of* {*a*..*b*} *norm*(*c*) ≤ *e*
shows *norm*(*setsum* (λ*l*. *content l* ∗*R* *c*) *p*) ≤ *e* ∗ *content*({*a*..*b*}) (**is** *?l* ≤ *?r*)
apply(*rule order-trans*,*rule setsum-norm*) **defer unfolding** *norm-scaleR setsum-left-distrib*[*THEN sym*]
apply(*rule order-trans*[*OF mult-left-mono*],*rule assms*,*rule setsum-abs-ge-zero*)
apply(*subst mult-commute*) **apply**(*rule mult-left-mono*)
apply(*rule order-trans*[*of - setsum content p*]) **apply**(*rule eq-refl*,*rule setsum-cong2*)
apply(*subst abs-of-nonneg*) **unfolding** *additive-content-division*[*OF assms*(*1*)]
proof− **from** *order-trans*[*OF norm-ge-zero*[*of c*] *assms*(*2*)] **show** *0* ≤ *e* **.**

fix *x* **assume** *x∈p* **from** *division-ofD(4)[OF assms(1) this]* **guess** *u v* **apply−by**(*erule exE*)+
thus *0 ≤ content x* **using** *content-pos-le* **by** *auto*
qed(*insert assms,auto*)

lemma *rsum-bound*: **assumes** *p tagged-division-of* {*a..b*} ∀*x∈*{*a..b*}. *norm*(*f x*) ≤ *e*
shows *norm*(*setsum* (λ(*x,k*). *content k* ∗$_R$ *f x*) *p*) ≤ *e* ∗ *content*({*a..b*})
proof(*cases* {*a..b*} = {}) **case** *True*
show *?thesis* **using** *assms(1)* **unfolding** *True tagged-division-of-trivial* **by** *auto*
next case *False* **show** *?thesis*
apply(*rule order-trans,rule setsum-norm*) **defer unfolding** *split-def norm-scaleR*
apply(*rule order-trans*[*OF setsum-mono*]) **apply**(*rule mult-left-mono*[*OF - abs-ge-zero, of - e*]) **defer**
unfolding *setsum-left-distrib*[*THEN sym*] **apply**(*subst mult-commute*) **apply**(*rule mult-left-mono*)
apply(*rule order-trans*[*of - setsum* (*content* ∘ *snd*) *p*]) **apply**(*rule eq-refl,rule setsum-cong2*)
apply(*subst o-def, rule abs-of-nonneg*)
proof− **show** *setsum* (*content* ∘ *snd*) *p* ≤ *content* {*a..b*} **apply**(*rule eq-refl*)
unfolding *additive-content-tagged-division*[*OF assms(1),THEN sym*] *split-def*
by *auto*
guess *w* **using** *nonempty-witness*[*OF False*] **.**
thus *e≥0* **apply−apply**(*rule order-trans*) **defer apply**(*rule assms(2)*[*rule-format*],*assumption*)
by *auto*
fix *xk* **assume** ∗:*xk∈p* **guess** *x k* **using** *surj-pair*[*of xk*] **apply−by**(*erule exE*)+
note *xk* = *this* ∗[*unfolded this*]
from *tagged-division-ofD(4)*[*OF assms(1) xk(2)*] **guess** *u v* **apply−by**(*erule exE*)+ **note** *uv=this*
show *0≤ content* (*snd xk*) **unfolding** *xk snd-conv uv* **by**(*rule content-pos-le*)
show *norm* (*f* (*fst xk*)) ≤ *e* **unfolding** *xk fst-conv* **using** *tagged-division-ofD(2,3)*[*OF assms(1) xk(2)*] *assms(2)* **by** *auto*
qed(*insert assms,auto*) **qed**

lemma *rsum-diff-bound*:
assumes *p tagged-division-of* {*a..b*} ∀*x∈*{*a..b*}. *norm*(*f x* − *g x*) ≤ *e*
shows *norm*(*setsum* (λ(*x,k*). *content k* ∗$_R$ *f x*) *p* − *setsum* (λ(*x,k*). *content k* ∗$_R$ *g x*) *p*) ≤ *e* ∗ *content*({*a..b*})
apply(*rule order-trans*[*OF - rsum-bound*[*OF assms*]]) **apply**(*rule eq-refl*) **apply**(*rule arg-cong*[**where** *f=norm*])
unfolding *setsum-subtractf*[*THEN sym*] **apply**(*rule setsum-cong2*) **unfolding** *scaleR.diff-right* **by** *auto*

lemma *has-integral-bound*: **fixes** *f*::*real^'n* ⇒ *'a*::*real-normed-vector*
assumes *0* ≤ *B* (*f has-integral i*) ({*a..b*}) ∀*x∈*{*a..b*}. *norm*(*f x*) ≤ *B*
shows *norm i* ≤ *B* ∗ *content* {*a..b*}
proof− **let** *?P* = *content* {*a..b*} > *0* **{ presume** *?P* ⟹ *?thesis*
thus *?thesis* **proof**(*cases ?P*) **case** *False*
hence ∗:*content* {*a..b*} = *0* **using** *content-lt-nz* **by** *auto*

hence $**{:}i = 0$ **using** *assms(2)* **apply**(*subst has-integral-null-eq[THEN sym]*) **by** *auto*
show *?thesis* **unfolding** $* ** $ **using** *assms(1)* **by** *auto*
qed *auto* **}** **assume** *ab:?P*
{ presume $\neg$ *?thesis* $\Longrightarrow$ *False* **thus** *?thesis* **by** *auto* **}**
assume $\neg$ *?thesis* **hence** $*{:}$*norm i − B * content {a..b} > 0* **by** *auto*
from *assms(2)*[*unfolded has-integral,rule-format,OF* $*$] **guess** *d* **apply−by**(*erule exE conjE*)+ **note** *d=this*[*rule-format*]
from *fine-division-exists*[*OF this(1), of a b*] **guess** *p* **. note** *p=this*
have $*{:}\bigwedge s\ B.$ *norm s ≤ B* $\Longrightarrow$ $\neg$ *(norm (s − i) < norm i − B)*
proof− **case** *goal1* **thus** *?case* **unfolding** *not-less*
using *norm-triangle-sub*[*of i s*] **unfolding** *norm-minus-commute* **by** *auto*
qed show *False* **using** *d(2)*[*OF conjI*[*OF p*]] $*$[*OF rsum-bound*[*OF p(1) assms(3)*]] **by** *auto* **qed**

23.26 Similar theorems about relationship among components.

lemma *rsum-component-le*: **fixes** *f*::*real^'n* $\Rightarrow$ *real^'m*
assumes *p tagged-division-of* {*a..b*} $\forall x{\in}${*a..b*}. *(f x)$i ≤ (g x)$i*
shows *(setsum* (λ(*x*,*k*). *content k* $*_R$ *f x) p)$i ≤ (setsum* (λ(*x*,*k*). *content k* $*_R$ *g x) p)$i*
unfolding *setsum-component* **apply**(*rule setsum-mono*)
apply(*rule mp*) **defer apply** *assumption* **unfolding** *split-paired-all* **apply** *rule* **unfolding** *split-conv*
proof− **fix** *a b* **assume** *ab*:(*a*,*b*) ∈ *p* **note** *assm = tagged-division-ofD(2−4)*[*OF assms(1) ab*]
from *this(3)* **guess** *u v* **apply−by**(*erule exE*)+ **note** *b=this*
show *(content b* $*_R$ *f a)* $ *i ≤ (content b* $*_R$ *g a)* $ *i* **unfolding** *b*
unfolding *Cart-nth.scaleR real-scaleR-def* **apply**(*rule mult-left-mono*)
defer apply(*rule content-pos-le*,*rule assms(2)*[*rule-format*]) **using** *assm* **by** *auto* **qed**

lemma *has-integral-component-le*: **fixes** *f*::*real^'n* $\Rightarrow$ *real^'m*
assumes *(f has-integral i) s (g has-integral j) s* $\forall x{\in}s.$ *(f x)$k ≤ (g x)$k*
shows *i$k ≤ j$k*
proof− **have** *lem*:$\bigwedge a\ b\ g\ i\ j.\ \bigwedge f$::*real^'n* $\Rightarrow$ *real^'m. (f has-integral i) ({a..b})* $\Longrightarrow$
(g has-integral j) ({a..b}) $\Longrightarrow$ $\forall x{\in}${*a..b*}. *(f x)$k ≤ (g x)$k* $\Longrightarrow$ *i$k ≤ j$k*
proof(*rule ccontr*) **case** *goal1* **hence** $*{:}$*0 < (i$k − j$k) / 3* **by** *auto*
guess *d1* **using** *goal1(1)*[*unfolded has-integral,rule-format,OF* $*$] **apply−by**(*erule exE conjE*)+ **note** *d1=this*[*rule-format*]
guess *d2* **using** *goal1(2)*[*unfolded has-integral,rule-format,OF* $*$] **apply−by**(*erule exE conjE*)+ **note** *d2=this*[*rule-format*]
guess *p* **using** *fine-division-exists*[*OF gauge-inter*[*OF d1(1) d2(1)*], *of a b*] **unfolding** *fine-inter* **.**
note *p = this(1) conjunctD2*[*OF this(2)*] **note** *le-less-trans*[*OF component-le-norm*, *of - - k*]
note *this*[*OF d1(2)*[*OF conjI*[*OF p(1,2)*]]] *this*[*OF d2(2)*[*OF conjI*[*OF p(1,3)*]]]

thus *False* **unfolding** *Cart-nth.diff* **using** *rsum-component-le*[*OF p*(*1*) *goal1*(*3*)] **by** *smt*
qed let *?P* = ∃ *a b*. *s* = {*a*..*b*}
{ presume ¬ *?P* ⟹ *?thesis* **thus** *?thesis* **proof**(*cases ?P*)
case *True* **then guess** *a b* **apply**−**by**(*erule exE*)+ **note** *s*=*this*
show *?thesis* **apply**(*rule lem*) **using** *assms*[*unfolded s*] **by** *auto*
qed *auto* **} assume** *as*:¬ *?P*
{ presume ¬ *?thesis* ⟹ *False* **thus** *?thesis* **by** *auto* **}**
assume ¬ *i*$*k* ≤ *j*$*k* **hence** *ij*:(*i*$*k* − *j*$*k*) / *3* > *0* **by** *auto*
note *has-integral-altD*[*OF - as this*] **from** *this*[*OF assms*(*1*)] *this*[*OF assms*(*2*)] **guess** *B1 B2* **. note** *B*=*this*[*rule-format*]
have *bounded* (*ball 0 B1* ∪ *ball* (*0*::*real*^'*n*) *B2*) **unfolding** *bounded-Un* **by**(*rule conjI bounded-ball*)+
from *bounded-subset-closed-interval*[*OF this*] **guess** *a b* **apply**− **by**(*erule exE*)+
note *ab* = *conjunctD2*[*OF this*[*unfolded Un-subset-iff*]]
guess *w1* **using** *B*(*2*)[*OF ab*(*1*)] **.. note** *w1*=*conjunctD2*[*OF this*]
guess *w2* **using** *B*(*4*)[*OF ab*(*2*)] **.. note** *w2*=*conjunctD2*[*OF this*]
have ∗:⋀*w1 w2 j i*::*real* .|*w1* − *i*| < (*i* − *j*) / *3* ⟹ |*w2* − *j*| < (*i* − *j*) / *3* ⟹ *w1* ≤ *w2* ⟹ *False* **by** *smt*
note *le-less-trans*[*OF component-le-norm*[*of - k*]] **note** *this*[*OF w1*(*2*)] *this*[*OF w2*(*2*)] **moreover**
have *w1*$*k* ≤ *w2*$*k* **apply**(*rule lem*[*OF w1*(*1*) *w2*(*1*)]) **using** *assms* **by** *auto*
ultimately
show *False* **unfolding** *Cart-nth.diff* **by**(*rule* ∗) **qed**

lemma *integral-component-le*: **fixes** *f*::*real*^'*n* ⇒ *real*^'*m*
assumes *f integrable-on s g integrable-on s* ∀ *x*∈*s*. (*f x*)$*k* ≤ (*g x*)$*k*
shows (*integral s f*)$*k* ≤ (*integral s g*)$*k*
apply(*rule has-integral-component-le*) **using** *integrable-integral assms* **by** *auto*

lemma *has-integral-dest-vec1-le*: **fixes** *f*::*real*^'*n* ⇒ *real*^*1*
assumes (*f has-integral i*) *s* (*g has-integral j*) *s* ∀ *x*∈*s*. *f x* ≤ *g x*
shows *dest-vec1 i* ≤ *dest-vec1 j* **apply**(*rule has-integral-component-le*[*OF assms*(*1*−*2*)])
using *assms*(*3*) **unfolding** *vector-le-def* **by** *auto*

lemma *integral-dest-vec1-le*: **fixes** *f*::*real*^'*n* ⇒ *real*^*1*
assumes *f integrable-on s g integrable-on s* ∀ *x*∈*s*. *f x* ≤ *g x*
shows *dest-vec1*(*integral s f*) ≤ *dest-vec1*(*integral s g*)
apply(*rule has-integral-dest-vec1-le*) **apply**(*rule-tac*[*1*−*2*] *integrable-integral*) **using** *assms* **by** *auto*

lemma *has-integral-component-nonneg*: **fixes** *f*::*real*^'*n* ⇒ *real*^'*m*
assumes (*f has-integral i*) *s* ∀ *x*∈*s*. *0* ≤ (*f x*)$*k* **shows** *0* ≤ *i*$*k*
using *has-integral-component-le*[*OF has-integral-0 assms*(*1*)] **using** *assms*(*2*) **by** *auto*

lemma *integral-component-nonneg*: **fixes** *f*::*real*^'*n* ⇒ *real*^'*m*
assumes *f integrable-on s* ∀ *x*∈*s*. *0* ≤ (*f x*)$*k* **shows** *0* ≤ (*integral s f*)$*k*
apply(*rule has-integral-component-nonneg*) **using** *assms* **by** *auto*

lemma *has-integral-dest-vec1-nonneg*: **fixes** *f*::*real^'n* ⇒ *real^1*
assumes (*f has-integral i*) *s* ∀*x*∈*s*. *0* ≤ *f x* **shows** *0* ≤ *i*
using *has-integral-component-nonneg*[*OF assms*(*1*), *of 1*]
using *assms*(*2*) **unfolding** *vector-le-def* **by** *auto*

lemma *integral-dest-vec1-nonneg*: **fixes** *f*::*real^'n* ⇒ *real^1*
assumes *f integrable-on s* ∀*x*∈*s*. *0* ≤ *f x* **shows** *0* ≤ *integral s f*
apply(*rule has-integral-dest-vec1-nonneg*) **using** *assms* **by** *auto*

lemma *has-integral-component-neg*: **fixes** *f*::*real^'n* ⇒ *real^'m*
assumes (*f has-integral i*) *s* ∀*x*∈*s*. (*f x*)$*k* ≤ *0* **shows** *i*$*k* ≤ *0*
using *has-integral-component-le*[*OF assms*(*1*) *has-integral-0*] *assms*(*2*) **by** *auto*

lemma *has-integral-dest-vec1-neg*: **fixes** *f*::*real^'n* ⇒ *real^1*
assumes (*f has-integral i*) *s* ∀*x*∈*s*. *f x* ≤ *0* **shows** *i* ≤ *0*
using *has-integral-component-neg*[*OF assms*(*1*),*of 1*] **using** *assms*(*2*) **by** *auto*

lemma *has-integral-component-lbound*:
assumes (*f has-integral i*) {*a*..*b*} ∀*x*∈{*a*..*b*}. *B* ≤ *f*(*x*)$*k* **shows** *B* ∗ *content* {*a*..*b*} ≤ *i*$*k*
using *has-integral-component-le*[*OF has-integral-const assms*(*1*),*of* (χ *i*. *B*) *k*] *assms*(*2*)
unfolding *Cart-lambda-beta vector-scaleR-component* **by**(*auto simp add*:*field-simps*)

lemma *has-integral-component-ubound*:
assumes (*f has-integral i*) {*a*..*b*} ∀*x*∈{*a*..*b*}. *f x*$*k* ≤ *B*
shows *i*$*k* ≤ *B* ∗ *content*({*a*..*b*})
using *has-integral-component-le*[*OF assms*(*1*) *has-integral-const*, *of k vec B*]
unfolding *vec-component Cart-nth.scaleR* **using** *assms*(*2*) **by**(*auto simp add*:*field-simps*)

lemma *integral-component-lbound*:
assumes *f integrable-on* {*a*..*b*} ∀*x*∈{*a*..*b*}. *B* ≤ *f*(*x*)$*k*
shows *B* ∗ *content*({*a*..*b*}) ≤ (*integral*({*a*..*b*}) *f*)$*k*
apply(*rule has-integral-component-lbound*) **using** *assms* **unfolding** *has-integral-integral* **by** *auto*

lemma *integral-component-ubound*:
assumes *f integrable-on* {*a*..*b*} ∀*x*∈{*a*..*b*}. *f*(*x*)$*k* ≤ *B*
shows (*integral*({*a*..*b*}) *f*)$*k* ≤ *B* ∗ *content*({*a*..*b*})
apply(*rule has-integral-component-ubound*) **using** *assms* **unfolding** *has-integral-integral* **by** *auto*

23.27 Uniform limit of integrable functions is integrable.

lemma *real-arch-invD*:
0 < (*e*::*real*) ⟹ (∃*n*::*nat*. *n* ≠ *0* ∧ *0* < *inverse* (*real n*) ∧ *inverse* (*real n*) < *e*)
by(*subst*(*asm*) *real-arch-inv*)

lemma *integrable-uniform-limit*: **fixes** f::*real^'n* $\Rightarrow$ *'a*::*banach*
assumes $\forall\, e{>}0.\ \exists\, g.\ (\forall\, x{\in}\{a..b\}.\ norm(f\ x - g\ x) \leq e) \wedge g$ *integrable-on* $\{a..b\}$
shows f *integrable-on* $\{a..b\}$
proof− **{ presume** $*$:*content* $\{a..b\} > 0 \Longrightarrow$ *?thesis*
show *?thesis* **apply** *cases* **apply**(*rule* $*$,*assumption*)
unfolding *content-lt-nz integrable-on-def* **using** *has-integral-null* **by** *auto* **}**
assume *as*:*content* $\{a..b\} > 0$
have $*$:$\bigwedge P.\ \forall\, e{>}(0$::*real*$).\ P\ e \Longrightarrow \forall\, n$::*nat*$.\ P\ (inverse\ (real\ n{+}1))$ **by** *auto*
from *choice*[*OF* $*$[*OF assms*]] **guess** g **.. note** *g*=*conjunctD2*[*OF this*[*rule-format*],*rule-format*]
from *choice*[*OF allI*[*OF g*(*2*)[*unfolded integrable-on-def*], *of* $\lambda x.\ x$]] **guess** i **..**
note *i*=*this*[*rule-format*]

have *Cauchy i* **unfolding** *Cauchy-def*
proof(*rule*,*rule*) **fix** *e*::*real* **assume** *e*>*0*
hence *e / 4 / content* $\{a..b\} > 0$ **using** *as* **by**(*auto simp add*:*field-simps*)
then guess *M* **apply**−**apply**(*subst*(*asm*) *real-arch-inv*) **by**(*erule exE conjE*)+
note *M*=*this*
show $\exists\, M.\ \forall\, m{\geq}M.\ \forall\, n{\geq}M.\ dist\ (i\ m)\ (i\ n) < e$ **apply**(*rule-tac x*=*M* **in**
exI,*rule*,*rule*,*rule*,*rule*)
proof− **case** *goal1* **have** *e*/*4*>*0* **using** ‹*e*>*0*› **by** *auto* **note** $* = i$[*unfolded*
has-integral,*rule-format*,*OF this*]
from $*$[*of m*] **guess** *gm* **apply**−**by**(*erule conjE exE*)+ **note** *gm*=*this*[*rule-format*]
from $*$[*of n*] **guess** *gn* **apply**−**by**(*erule conjE exE*)+ **note** *gn*=*this*[*rule-format*]
from *fine-division-exists*[*OF gauge-inter*[*OF gm*(*1*) *gn*(*1*)], *of a b*] **guess** *p* **.**
note *p*=*this*
have *lem2*:$\bigwedge$*s1 s2 i1 i2*. *norm*(*s2* − *s1*) ≤ *e*/*2* $\Longrightarrow$ *norm*(*s1* − *i1*) < *e* / *4*
$\Longrightarrow$ *norm*(*s2* − *i2*) < *e* / *4* $\Longrightarrow$*norm*(*i1* − *i2*) < *e*
proof− **case** *goal1* **have** *norm* (*i1* − *i2*) ≤ *norm* (*i1* − *s1*) + *norm* (*s1* −
s2) + *norm* (*s2* − *i2*)
using *norm-triangle-ineq*[*of i1* − *s1 s1* − *i2*]
using *norm-triangle-ineq*[*of s1* − *s2 s2* − *i2*] **by**(*auto simp add*:*algebra-simps*)
also have ... < *e* **using** *goal1* **unfolding** *norm-minus-commute* **by**(*auto*
simp add:*algebra-simps*)
finally show *?case* **.**
qed
show *?case* **unfolding** *dist-norm* **apply**(*rule lem2*) **defer**
apply(*rule gm*(*2*)[*OF conjI*[*OF p*(*1*)]],*rule-tac*[*2*] *gn*(*2*)[*OF conjI*[*OF p*(*1*)]])
using *conjunctD2*[*OF p*(*2*)[*unfolded fine-inter*]] **apply**− **apply** *assumption*+
apply(*rule order-trans*)
apply(*rule rsum-diff-bound*[*OF p*(*1*), **where** *e*=*2 / real M*])
proof show *2 / real M* $*$ *content* $\{a..b\} \leq e$ / *2* **unfolding** *divide-inverse*
using *M as* **by**(*auto simp add*:*field-simps*)
fix *x* **assume** *x*:$x \in \{a..b\}$
have *norm* (*f x* − *g n x*) + *norm* (*f x* − *g m x*) ≤ *inverse* (*real n* + *1*) +
inverse (*real m* + *1*)
using *g*(*1*)[*OF x*, *of n*] *g*(*1*)[*OF x*, *of m*] **by** *auto*
also have ... ≤ *inverse* (*real M*) + *inverse* (*real M*) **apply**(*rule add-mono*)
apply(*rule-tac*[!] *le-imp-inverse-le*) **using** *goal1 M* **by** *auto*

also have ... = *2 / real M* **unfolding** *divide-inverse* **by** *auto*
finally show *norm (g n x − g m x) ≤ 2 / real M*
using *norm-triangle-le*[*of g n x − f x f x − g m x 2 / real M*]
by(*auto simp add*:*algebra-simps simp add*:*norm-minus-commute*)
qed qed qed
from *this*[*unfolded convergent-eq-cauchy*[*THEN sym*]] **guess** *s* **.. note** *s=this*

show *?thesis* **unfolding** *integrable-on-def* **apply**(*rule-tac x=s* **in** *exI*) **unfolding** *has-integral*
proof(*rule*,*rule*)
case *goal1* **hence** ∗:*e/3 > 0* **by** *auto*
from *s*[*unfolded Lim-sequentially*,*rule-format*,*OF this*] **guess** *N1* **.. note** *N1=this*
from *goal1 as* **have** *e / 3 / content* {*a..b*} *> 0* **by**(*auto simp add*:*field-simps*)
from *real-arch-invD*[*OF this*] **guess** *N2* **apply−by**(*erule exE conjE*)+ **note** *N2=this*
from *i*[*of N1 + N2*,*unfolded has-integral*,*rule-format*,*OF* ∗] **guess** *g′* **.. note** *g′=conjunctD2*[*OF this*,*rule-format*]
have *lem*:⋀*sf sg i. norm*(*sf − sg*) ≤ *e / 3* ⟹ *norm*(*i − s*) < *e / 3* ⟹ *norm*(*sg − i*) < *e / 3* ⟹ *norm*(*sf − s*) < *e*
proof− **case** *goal1* **have** *norm* (*sf − s*) ≤ *norm* (*sf − sg*) + *norm* (*sg − i*) + *norm* (*i − s*)
using *norm-triangle-ineq*[*of sf − sg sg − s*]
using *norm-triangle-ineq*[*of sg − i i − s*] **by**(*auto simp add*:*algebra-simps*)
also have ... < *e* **using** *goal1* **unfolding** *norm-minus-commute* **by**(*auto simp add*:*algebra-simps*)
finally show *?case* **.**
qed
show *?case* **apply**(*rule-tac x=g′* **in** *exI*) **apply**(*rule*,*rule g′*)
proof(*rule*,*rule*) **fix** *p* **assume** *p*:*p tagged-division-of* {*a..b*} ∧ *g′ fine p* **note** ∗ = *g′*(*2*)[*OF this*]
show *norm* ((∑ (*x*, *k*)∈*p. content k* ∗$_R$ *f x*) − *s*) < *e* **apply−apply**(*rule lem*[*OF - - ∗*])
apply(*rule order-trans*,*rule rsum-diff-bound*[*OF p*[*THEN conjunct1*]]) **apply**(*rule*,*rule g*,*assumption*)
proof− **have** *content* {*a..b*} < *e / 3* ∗ (*real N2*)
using *N2* **unfolding** *inverse-eq-divide* **using** *as* **by**(*auto simp add*:*field-simps*)
hence *content* {*a..b*} < *e / 3* ∗ (*real* (*N1 + N2*) + *1*)
apply−apply(*rule less-le-trans*,*assumption*) **using** ‹*e>0*› **by** *auto*
thus *inverse* (*real* (*N1 + N2*) + *1*) ∗ *content* {*a..b*} ≤ *e / 3*
unfolding *inverse-eq-divide* **by**(*auto simp add*:*field-simps*)
show *norm* (*i* (*N1 + N2*) − *s*) < *e / 3* **by**(*rule N1*[*rule-format*,*unfolded dist-norm*],*auto*)
qed qed qed qed

23.28 Negligible sets.

definition *indicator s* ≡ (λ*x. if x* ∈ *s then 1 else* (*0*::*real*))

lemma *dest-vec1-indicator*:

indicator s x = (if x ∈ s then 1 else 0) **unfolding** *indicator-def* **by** *auto*

lemma *indicator-pos-le*[*intro*]: *0 ≤ (indicator s x)* **unfolding** *indicator-def* **by** *auto*

lemma *indicator-le-1*[*intro*]: *(indicator s x) ≤ 1* **unfolding** *indicator-def* **by** *auto*

lemma *dest-vec1-indicator-abs-le-1*: *abs(indicator s x) ≤ 1*
unfolding *indicator-def* **by** *auto*

definition *negligible (s::(real^'n) set) ≡ (∀ a b. ((indicator s) has-integral 0) {a..b})*

lemma *indicator-simps*[*simp*]:*x∈s ⟹ indicator s x = 1 x∉s ⟹ indicator s x = 0*
unfolding *indicator-def* **by** *auto*

23.29 Negligibility of hyperplane.

lemma *vsum-nonzero-image-lemma*:
assumes *finite s g(a) = 0*
∀ x∈s. ∀ y∈s. f x = f y ∧ x ≠ y ⟶ g(f x) = 0
shows *setsum g {f x |x. x ∈ s ∧ f x ≠ a} = setsum (g o f) s*
unfolding *setsum-iterate*[*OF assms(1)*] **apply**(*subst setsum-iterate*) **defer**
apply(*rule iterate-nonzero-image-lemma*) **apply**(*rule assms monoidal-monoid*)+
unfolding *assms* **using** *neutral-add* **unfolding** *neutral-add* **using** *assms* **by** *auto*

lemma *interval-doublesplit*: **shows** *{a..b} ∩ {x . abs(x$k − c) ≤ (e::real)} =*
{(χ i. if i = k then max (a$k) (c − e) else a$i) .. (χ i. if i = k then min (b$k) (c + e) else b$i)}
proof− **have** ∗:⋀*x c e::real. abs(x − c) ≤ e ⟷ x ≥ c − e ∧ x ≤ c + e* **by** *auto*
have ∗∗:⋀*s P Q. s ∩ {x. P x ∧ Q x} = (s ∩ {x. Q x}) ∩ {x. P x}* **by** *blast*
show *?thesis* **unfolding** ∗ ∗∗ *interval-split* **by**(*rule refl*) **qed**

lemma *division-doublesplit*: **assumes** *p division-of {a..b::real^'n}*
shows *{l ∩ {x. abs(x$k − c) ≤ e} |l. l ∈ p ∧ l ∩ {x. abs(x$k − c) ≤ e} ≠ {}} division-of ({a..b} ∩ {x. abs(x$k − c) ≤ e})*
proof− **have** ∗:⋀*x c. abs(x − c) ≤ e ⟷ x ≥ c − e ∧ x ≤ c + e* **by** *auto*
have ∗∗:⋀*p q p' q'. p division-of q ⟹ p = p' ⟹ q = q' ⟹ p' division-of q'* **by** *auto*
note *division-split(1)*[*OF assms*, **where** *c=c+e* **and** *k=k*,*unfolded interval-split*]
note *division-split(2)*[*OF this*, **where** *c=c−e* **and** *k=k*]
thus *?thesis* **apply**(*rule* ∗∗) **unfolding** *interval-doublesplit* **unfolding** ∗ **unfolding** *interval-split interval-doublesplit*
apply(*rule set-ext*) **unfolding** *mem-Collect-eq* **apply** *rule* **apply**(*erule conjE exE*)+ **apply**(*rule-tac x=la* **in** *exI*) **defer**
apply(*erule conjE exE*)+ **apply**(*rule-tac x=l ∩ {x. c + e ≥ x $ k}* **in** *exI*) **apply** *rule* **defer apply** *rule*
apply(*rule-tac x=l* **in** *exI*) **by** *blast*+ **qed**

lemma *content-doublesplit*: **assumes** *0 < e*
obtains *d* **where** *0 < d content*({*a..b*} ∩ {*x*. *abs*(*x*$*k* − *c*) ≤ *d*}) < *e*
proof(*cases content* {*a..b*} = *0*)
case *True* **show** *?thesis* **apply**(*rule that*[*of 1*]) **defer unfolding** *interval-doublesplit*
apply(*rule le-less-trans*[*OF content-subset*]) **defer apply**(*subst True*)
unfolding *interval-doublesplit*[*THEN sym*] **using** *assms* **by** *auto*
next case *False* **def** *d* ≡ *e* / *3* / *setprod* (λ*i*. *b*$*i* − *a*$*i*) (*UNIV* − {*k*})
note *False*[*unfolded content-eq-0 not-ex not-le*, *rule-format*]
hence *prod0*:*0* < *setprod* (λ*i*. *b*$*i* − *a*$*i*) (*UNIV* − {*k*}) **apply**−**apply**(*rule setprod-pos*) **by** *smt*
hence *d* > *0* **unfolding** *d-def* **using** *assms* **by**(*auto simp add*:*field-simps*) **thus** *?thesis*
proof(*rule that*[*of d*]) **have** ∗:*UNIV* = *insert k* (*UNIV* − {*k*}) **by** *auto*
have ∗∗:{*a..b*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*} ≠ {} ⟹
(∏ *i*∈*UNIV* − {*k*}. *interval-upperbound* ({*a..b*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) $ *i*
− *interval-lowerbound* ({*a..b*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) $ *i*)
= (∏ *i*∈*UNIV* − {*k*}. *b*$*i* − *a*$*i*) **apply**(*rule setprod-cong*,*rule refl*)
unfolding *interval-doublesplit interval-eq-empty not-ex not-less* **unfolding** *interval-bounds* **by** *auto*
show *content* ({*a..b*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) < *e* **apply**(*cases*) **unfolding** *content-def* **apply**(*subst if-P*,*assumption*,*rule assms*)
unfolding *if-not-P* **apply**(*subst* ∗) **apply**(*subst setprod-insert*) **unfolding** ∗∗
unfolding *interval-doublesplit interval-eq-empty not-ex not-less* **unfolding** *interval-bounds* **unfolding** *Cart-lambda-beta if-P*[*OF refl*]
proof− **have** (*min* (*b* $ *k*) (*c* + *d*) − *max* (*a* $ *k*) (*c* − *d*)) ≤ *2* ∗ *d* **by** *auto*
also have ... < *e* / (∏ *i*∈*UNIV* − {*k*}. *b* $ *i* − *a* $ *i*) **unfolding** *d-def* **using** *assms prod0* **by**(*auto simp add*:*field-simps*)
finally show (*min* (*b* $ *k*) (*c* + *d*) − *max* (*a* $ *k*) (*c* − *d*)) ∗ (∏ *i*∈*UNIV* − {*k*}. *b* $ *i* − *a* $ *i*) < *e*
unfolding *pos-less-divide-eq*[*OF prod0*] **. qed** *auto* **qed qed**

lemma *negligible-standard-hyperplane*[*intro*]: *negligible* {*x*. *x*$*k* = (*c*::*real*)}
unfolding *negligible-def has-integral* **apply**(*rule*,*rule*,*rule*,*rule*)
proof− **case** *goal1* **from** *content-doublesplit*[*OF this*,*of a b k c*] **guess** *d* **. note** *d*=*this* **let** *?i* = *indicator* {*x*. *x*$*k* = *c*}
show *?case* **apply**(*rule-tac x*=λ*x*. *ball x d* **in** *exI*) **apply**(*rule*,*rule gauge-ball*,*rule d*)
proof(*rule*,*rule*) **fix** *p* **assume** *p*:*p tagged-division-of* {*a..b*} ∧ (λ*x*. *ball x d*) *fine p*
have ∗:(∑ (*x*, *ka*)∈*p*. *content ka* ∗*R* *?i x*) = (∑ (*x*, *ka*)∈*p*. *content* (*ka* ∩ {*x*. *abs*(*x*$*k* − *c*) ≤ *d*}) ∗*R* *?i x*)
apply(*rule setsum-cong2*) **unfolding** *split-paired-all real-scaleR-def mult-cancel-right split-conv*
apply(*cases*,*rule disjI1*,*assumption*,*rule disjI2*)
proof− **fix** *x l* **assume** *as*:(*x*,*l*)∈*p* *?i x* ≠ *0* **hence** *xk*:*x*$*k* = *c* **unfolding** *indicator-def* **apply**−**by**(*rule ccontr*,*auto*)
show *content l* = *content* (*l* ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) **apply**(*rule arg-cong*[**where** *f*=*content*])

apply(*rule set-ext*,*rule*,*rule*) **unfolding** *mem-Collect-eq*
proof− **fix** *y* **assume** *y*:*y*∈*l* **note** *p*[*THEN conjunct2*,*unfolded fine-def*,*rule-format*,*OF as*(*1*),*unfolded split-conv*]
note *this*[*unfolded subset-eq mem-ball dist-norm*,*rule-format*,*OF y*] **note** *le-less-trans*[*OF component-le-norm*[*of - k*] *this*]
thus |*y* $ *k* − *c*| ≤ *d* **unfolding** *Cart-nth.diff xk* **by** *auto*
qed *auto* **qed**
note *p*′= *tagged-division-ofD*[*OF p*[*THEN conjunct1*]] **and** *p*′′=*division-of-tagged-division*[*OF p*[*THEN conjunct1*]]
show *norm* ((∑ (*x*, *ka*)∈*p*. *content ka* ∗*R* *?i x*) − *0*) < *e* **unfolding** *diff-0-right* ∗ **unfolding** *real-scaleR-def real-norm-def*
apply(*subst abs-of-nonneg*) **apply**(*rule setsum-nonneg*,*rule*) **unfolding** *split-paired-all split-conv*
apply(*rule mult-nonneg-nonneg*) **apply**(*drule p*′(*4*)) **apply**(*erule exE*)+ **apply**(*rule-tac b*=*b* **in** *back-subst*)
prefer *2* **apply**(*subst*(*asm*) *eq-commute*) **apply** *assumption*
apply(*subst interval-doublesplit*) **apply**(*rule content-pos-le*) **apply**(*rule indicator-pos-le*)
proof− **have** (∑ (*x*, *ka*)∈*p*. *content* (*ka* ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) ∗ *?i x*) ≤ (∑ (*x*, *ka*)∈*p*. *content* (*ka* ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}))
apply(*rule setsum-mono*) **unfolding** *split-paired-all split-conv*
apply(*rule mult-right-le-one-le*) **apply**(*drule p*′(*4*)) **by**(*auto simp add*:*interval-doublesplit intro*!:*content-pos-le*)
also have ... < *e* **apply**(*subst setsum-over-tagged-division-lemma*[*OF p*[*THEN conjunct1*]])
proof− **case** *goal1* **have** *content* ({*u*..*v*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) ≤ *content* {*u*..*v*}
unfolding *interval-doublesplit* **apply**(*rule content-subset*) **unfolding** *interval-doublesplit*[*THEN sym*] **by** *auto*
thus *?case* **unfolding** *goal1* **unfolding** *interval-doublesplit* **using** *content-pos-le* **by** *smt*
next have ∗:*setsum content* {*l* ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*} |*l*. *l* ∈ *snd* ‘ *p* ∧ *l* ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*} ≠ {}} ≥ *0*
apply(*rule setsum-nonneg*,*rule*) **unfolding** *mem-Collect-eq image-iff* **apply**(*erule exE bexE conjE*)+ **unfolding** *split-paired-all*
proof− **fix** *x l a b* **assume** *as*:*x* = *l* ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*} (*a*, *b*) ∈ *p l* = *snd* (*a*, *b*)
guess *u v* **using** *p*′(*4*)[*OF as*(*2*)] **apply**−**by**(*erule exE*)+ **note** ∗ = *this*
show *content x* ≥ *0* **unfolding** *as snd-conv* ∗ *interval-doublesplit* **by**(*rule content-pos-le*)
qed have ∗∗:*norm* (*1*::*real*) ≤ *1* **by** *auto* **note** *division-doublesplit*[*OF p*′′,*unfolded interval-doublesplit*]
note *dsum-bound*[*OF this* ∗∗,*unfolded interval-doublesplit*[*THEN sym*]]
note *this*[*unfolded real-scaleR-def real-norm-def mult-1-right mult-1*, *of k c d*] **note** *le-less-trans*[*OF this d*(*2*)]
from *this*[*unfolded abs-of-nonneg*[*OF* ∗]] **show** (∑ *ka*∈*snd* ‘ *p*. *content* (*ka* ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*})) < *e*
apply(*subst vsum-nonzero-image-lemma*[*of snd* ‘ *p content* {}, *unfolded o-def*,*THEN sym*])
apply(*rule finite-imageI p*′ *content-empty*)+ **unfolding** *forall-in-division*[*OF*

p'']
proof(*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*erule conjE*) **fix** *m n u v*
assume *as*:{*m..n*} ∈ *snd* ' *p* {*u..v*} ∈ *snd* ' *p* {*m..n*} ≠ {*u..v*} {*m..n*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*} = {*u..v*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}
have ({*m..n*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) ∩ ({*u..v*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) ⊆ {*m..n*} ∩ {*u..v*} **by** *blast*
note *subset-interior*[*OF this*, *unfolded division-ofD*(*5*)[*OF p'' as*(*1*−*3*)] *interior-inter*[*of* {*m..n*}]]
hence *interior* ({*m..n*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) = {} **unfolding** *as Int-absorb* **by** *auto*
thus *content* ({*m..n*} ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) = *0* **unfolding** *interval-doublesplit content-eq-0-interior*[*THEN sym*] **.**
qed qed
finally show (∑ (*x*, *ka*)∈*p*. *content* (*ka* ∩ {*x*. |*x* $ *k* − *c*| ≤ *d*}) ∗ *?i x*) < *e*
.
qed qed qed

23.30 A technical lemma about "refinement" of division.

lemma *tagged-division-finer*: **fixes** *p*::((*real*^'*n*) × ((*real*^'*n*) *set*)) *set*
assumes *p tagged-division-of* {*a..b*} *gauge d*
obtains *q* **where** *q tagged-division-of* {*a..b*} *d fine q* ∀(*x*,*k*) ∈ *p*. *k* ⊆ *d*(*x*) ⟶ (*x*,*k*) ∈ *q*
proof−
let *?P* = λ*p*. *p tagged-partial-division-of* {*a..b*} ⟶ *gauge d* ⟶
(∃ *q*. *q tagged-division-of* (⋃{*k*. ∃ *x*. (*x*,*k*) ∈ *p*}) ∧ *d fine q* ∧
(∀ (*x*,*k*) ∈ *p*. *k* ⊆ *d*(*x*) ⟶ (*x*,*k*) ∈ *q*))
{ have ∗:*finite p p tagged-partial-division-of* {*a..b*} **using** *assms*(*1*) **unfolding** *tagged-division-of-def* **by** *auto*
presume ⋀*p*. *finite p* ⟹ *?P p* **from** *this*[*rule-format*,*OF* ∗ *assms*(*2*)] **guess** *q* **.. note** *q*=*this*
thus *?thesis* **apply**−**apply**(*rule that*[*of q*]) **unfolding** *tagged-division-ofD*[*OF assms*(*1*)] **by** *auto*
} **fix** *p*::((*real*^'*n*) × ((*real*^'*n*) *set*)) *set* **assume** *as*:*finite p*
show *?P p* **apply**(*rule*,*rule*) **using** *as* **proof**(*induct p*)
case *empty* **show** *?case* **apply**(*rule-tac x*={} **in** *exI*) **unfolding** *fine-def* **by** *auto*
next case (*insert xk p*) **guess** *x k* **using** *surj-pair*[*of xk*] **apply**− **by**(*erule exE*)+ **note** *xk*=*this*
note *tagged-partial-division-subset*[*OF insert*(*4*) *subset-insertI*]
from *insert*(*3*)[*OF this insert*(*5*)] **guess** *q1* **.. note** *q1* = *conjunctD3*[*OF this*]
have ∗:⋃{*l*. ∃ *y*. (*y*,*l*) ∈ *insert xk p*} = *k* ∪ ⋃{*l*. ∃ *y*. (*y*,*l*) ∈ *p*} **unfolding** *xk* **by** *auto*
note *p* = *tagged-partial-division-ofD*[*OF insert*(*4*)]
from *p*(*4*)[*unfolded xk*, *OF insertI1*] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*

have *finite* {*k*. ∃ *x*. (*x*, *k*) ∈ *p*}
apply(*rule finite-subset*[*of* - *snd* ' *p*],*rule*) **unfolding** *subset-eq image-iff*

mem-Collect-eq
apply(*erule exE*,*rule-tac x*=(*xa*,*x*) **in** *bexI*) **using** *p* **by** *auto*
hence *int*:*interior* {*u*..*v*} ∩ *interior* (⋃{*k*. ∃ *x*. (*x*, *k*) ∈ *p*}) = {}
apply(*rule inter-interior-unions-intervals*) **apply**(*rule open-interior*) **apply**(*rule-tac*[!] *ballI*)
unfolding *mem-Collect-eq* **apply**(*erule-tac*[!] *exE*) **apply**(*drule p*(*4*)[*OF insertI2*],*assumption*)
apply(*rule p*(*5*)) **unfolding** *uv xk* **apply**(*rule insertI1*,*rule insertI2*) **apply** *assumption*
using *insert*(*2*) **unfolding** *uv xk* **by** *auto*

show *?case* **proof**(*cases* {*u*..*v*} ⊆ *d x*)
case *True* **thus** *?thesis* **apply**(*rule-tac x*={(*x*,{*u*..*v*})} ∪ *q1* **in** *exI*) **apply** *rule*
unfolding ∗ *uv* **apply**(*rule tagged-division-union*,*rule tagged-division-of-self*)
apply(*rule p*[*unfolded xk uv*] *insertI1*)+ **apply**(*rule q1*,*rule int*)
apply(*rule*,*rule fine-union*,*subst fine-def*) **defer apply**(*rule q1*)
unfolding *Ball-def split-paired-All split-conv* **apply**(*rule*,*rule*,*rule*,*rule*)
apply(*erule insertE*) **defer apply**(*rule UnI2*) **apply**(*drule q1*(*3*)[*rule-format*]) **unfolding** *xk uv* **by** *auto*
next case *False* **from** *fine-division-exists*[*OF assms*(*2*), *of u v*] **guess** *q2* **. note** *q2*=*this*
show *?thesis* **apply**(*rule-tac x*=*q2* ∪ *q1* **in** *exI*)
apply *rule* **unfolding** ∗ *uv* **apply**(*rule tagged-division-union q2 q1 int fine-union*)+
unfolding *Ball-def split-paired-All split-conv* **apply** *rule* **apply**(*rule fine-union*)
apply(*rule q1 q2*)+ **apply**(*rule*,*rule*,*rule*,*rule*) **apply**(*erule insertE*)
apply(*rule UnI2*) **defer apply**(*drule q1*(*3*)[*rule-format*])**using** *False* **unfolding** *xk uv* **by** *auto*
qed qed qed

23.31 Hence the main theorem about negligible sets.

lemma *finite-product-dependent*: **assumes** *finite s* ⋀*x*. *x*∈*s*⟹ *finite* (*t x*)
shows *finite* {(*i*, *j*) |*i j*. *i* ∈ *s* ∧ *j* ∈ *t i*} **using** *assms*
proof(*induct*) **case** (*insert x s*)
have ∗:{(*i*, *j*) |*i j*. *i* ∈ *insert x s* ∧ *j* ∈ *t i*} = (λ*y*. (*x*,*y*)) ‘ (*t x*) ∪ {(*i*, *j*) |*i j*. *i* ∈ *s* ∧ *j* ∈ *t i*} **by** *auto*
show *?case* **unfolding** ∗ **apply**(*rule finite-UnI*) **using** *insert* **by** *auto* **qed** *auto*

lemma *sum-sum-product*: **assumes** *finite s* ∀ *i*∈*s*. *finite* (*t i*)
shows *setsum* (λ*i*. *setsum* (*x i*) (*t i*)::*real*) *s* = *setsum* (λ(*i*,*j*). *x i j*) {(*i*,*j*) | *i j*. *i* ∈ *s* ∧ *j* ∈ *t i*} **using** *assms*
proof(*induct*) **case** (*insert a s*)
have ∗:{(*i*, *j*) |*i j*. *i* ∈ *insert a s* ∧ *j* ∈ *t i*} = (λ*y*. (*a*,*y*)) ‘ (*t a*) ∪ {(*i*, *j*) |*i j*. *i* ∈ *s* ∧ *j* ∈ *t i*} **by** *auto*
show *?case* **unfolding** ∗ **apply**(*subst setsum-Un-disjoint*) **unfolding** *setsum-insert*[*OF insert*(*1*−*2*)]
prefer *4* **apply**(*subst insert*(*3*)) **unfolding** *add-right-cancel*

proof− **show** *setsum* (*x a*) (*t a*) = (∑ (*xa*, *y*)∈*Pair a* ‘ *t a*. *x xa y*) **apply**(*subst setsum-reindex*) **unfolding** *inj-on-def* **by** *auto*
show *finite* {(*i*, *j*) |*i j*. *i* ∈ *s* ∧ *j* ∈ *t i*} **apply**(*rule finite-product-dependent*) **using** *insert* **by** *auto*
qed(*insert insert*, *auto*) **qed** *auto*

lemma *has-integral-negligible*: **fixes** *f*::*real^'n* ⇒ *'a*::*real-normed-vector*
assumes *negligible s* ∀ *x*∈(*t* − *s*). *f x* = *0*
shows (*f has-integral 0*) *t*
proof− **presume** *P*:⋀*f*::*real^'n* ⇒ *'a*. ⋀*a b*. (∀ *x*. ~(*x* ∈ *s*) ⟶ *f x* = *0*) ⟹ (*f has-integral 0*) ({*a..b*})
let *?f* = (λ*x*. *if x* ∈ *t then f x else 0*)
show *?thesis* **apply**(*rule-tac f*=*?f* **in** *has-integral-eq*) **apply**(*rule*) **unfolding** *if-P* **apply**(*rule refl*)
apply(*subst has-integral-alt*) **apply**(*cases*,*subst if-P*,*assumption*) **unfolding** *if-not-P*
proof− **assume** ∃ *a b*. *t* = {*a..b*} **then guess** *a b* **apply**−**by**(*erule exE*)+ **note** *t* = *this*
show (*?f has-integral 0*) *t* **unfolding** *t* **apply**(*rule P*) **using** *assms*(*2*) **unfolding** *t* **by** *auto*
next show ∀ *e*>*0*. ∃ *B*>*0*. ∀ *a b*. *ball 0 B* ⊆ {*a..b*} ⟶ (∃ *z*. ((λ*x*. *if x* ∈ *t then ?f x else 0*) *has-integral z*) {*a..b*} ∧ *norm* (*z* − *0*) < *e*)
apply(*safe*,*rule-tac x*=*1* **in** *exI*,*rule*) **apply**(*rule zero-less-one*,*safe*) **apply**(*rule-tac x*=*0* **in** *exI*)
apply(*rule*,*rule P*) **using** *assms*(*2*) **by** *auto*
qed
next fix *f*::*real^'n* ⇒ *'a* **and** *a b*::*real^'n* **assume** *assm*:∀ *x*. *x* ∉ *s* ⟶ *f x* = *0*
show (*f has-integral 0*) {*a..b*} **unfolding** *has-integral*
proof(*safe*) **case** *goal1*
hence ⋀*n*. *e* / *2* / ((*real n*+*1*) ∗ (*2* ^ *n*)) > *0*
apply−**apply**(*rule divide-pos-pos*) **defer apply**(*rule mult-pos-pos*) **by**(*auto simp add*:*field-simps*)
note *assms*(*1*)[*unfolded negligible-def has-integral*,*rule-format*,*OF this*,*of a b*] **note** *allI*[*OF this*,*of* λ*x*. *x*]
from *choice*[*OF this*] **guess** *d* **.. note** *d*=*conjunctD2*[*OF this*[*rule-format*]]
show *?case* **apply**(*rule-tac x*=λ*x*. *d* (*nat* ⌊*norm* (*f x*)⌋) *x* **in** *exI*)
proof *safe* **show** *gauge* (λ*x*. *d* (*nat* ⌊*norm* (*f x*)⌋) *x*) **using** *d*(*1*) **unfolding** *gauge-def* **by** *auto*
fix *p* **assume** *as*:*p tagged-division-of* {*a..b*} (λ*x*. *d* (*nat* ⌊*norm* (*f x*)⌋) *x*) *fine p*
let *?goal* = *norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗*R* *f x*) − *0*) < *e*
{ presume *p*≠{} ⟹ *?goal* **thus** *?goal* **apply**(*cases p*={}) **using** *goal1* **by** *auto* **}**
assume *as'*:*p* ≠ {} **from** *real-arch-simple*[*of Sup*((λ(*x*,*k*). *norm*(*f x*)) ‘ *p*)] **guess** *N* **..**
hence *N*:∀ *x*∈(λ(*x*, *k*). *norm* (*f x*)) ‘ *p*. *x* ≤ *real N* **apply**(*subst*(*asm*) *Sup-finite-le-iff*) **using** *as as'* **by** *auto*
have ∀ *i*. ∃ *q*. *q tagged-division-of* {*a..b*} ∧ (*d i*) *fine q* ∧ (∀ (*x*, *k*)∈*p*. *k* ⊆ (*d i*) *x* ⟶ (*x*, *k*) ∈ *q*)

apply(*rule*,*rule tagged-division-finer*[*OF as*(*1*) *d*(*1*)]) **by** *auto*
from *choice*[*OF this*] **guess** *q* **.. note** *q*=*conjunctD3*[*OF this*[*rule-format*]]
have ∗:⋀*i*. (∑(*x*, *k*)∈*q i*. *content k* ∗$_R$ *indicator s x*) ≥ *0* **apply**(*rule setsum-nonneg*,*safe*)
unfolding *real-scaleR-def* **apply**(*rule mult-nonneg-nonneg*) **apply**(*drule tagged-division-ofD*(*4*)[*OF q*(*1*)]) **by** *auto*
have ∗∗:⋀*f g s t*. *finite s* ⟹ *finite t* ⟹ (∀(*x*,*y*) ∈ *t*. (*0*::*real*) ≤ *g*(*x*,*y*)) ⟹ (∀*y*∈*s*. ∃*x*. (*x*,*y*) ∈ *t* ∧ *f*(*y*) ≤ *g*(*x*,*y*)) ⟹ *setsum f s* ≤ *setsum g t*
proof− **case** *goal1* **thus** *?case* **apply**−**apply**(*rule setsum-le-included*[*of s t g snd f*]) **prefer** *4*
apply *safe* **apply**(*erule-tac x*=*x* **in** *ballE*) **apply**(*erule exE*) **apply**(*rule-tac x*=(*xa*,*x*) **in** *bexI*) **by** *auto* **qed**
have *norm* ((∑(*x*, *k*)∈*p*. *content k* ∗$_R$ *f x*) − *0*) ≤ *setsum* (λ*i*. (*real i* + *1*) ∗
norm(*setsum* (λ(*x*,*k*). *content k* ∗$_R$ *indicator s x*) (*q i*))) {*0*..*N*+*1*}
unfolding *real-norm-def setsum-right-distrib abs-of-nonneg*[*OF* ∗] *diff-0-right*
apply(*rule order-trans*,*rule setsum-norm*) **defer apply**(*subst sum-sum-product*) **prefer** *3*
proof(*rule* ∗∗,*safe*) **show** *finite* {(*i*, *j*) |*i j*. *i* ∈ {*0*..*N* + *1*} ∧ *j* ∈ *q i*} **apply**(*rule finite-product-dependent*) **using** *q* **by** *auto*
fix *i a b* **assume** *as*′′:(*a*,*b*) ∈ *q i* **show** *0* ≤ (*real i* + *1*) ∗ (*content b* ∗$_R$ *indicator s a*)
unfolding *real-scaleR-def* **apply**(*rule mult-nonneg-nonneg*) **defer apply**(*rule mult-nonneg-nonneg*)
using *tagged-division-ofD*(*4*)[*OF q*(*1*) *as*′′] **by** *auto*
next fix *i*::*nat* **show** *finite* (*q i*) **using** *q* **by** *auto*
next fix *x k* **assume** *xk*:(*x*,*k*) ∈ *p* **def** *n* ≡ *nat* ⌊*norm* (*f x*)⌋
have ∗:*norm* (*f x*) ∈ (λ(*x*, *k*). *norm* (*f x*)) ‘ *p* **using** *xk* **by** *auto*
have *nfx*:*real n* ≤ *norm*(*f x*) *norm*(*f x*) ≤ *real n* + *1* **unfolding** *n-def* **by** *auto*
hence *n* ∈ {*0*..*N* + *1*} **using** *N*[*rule-format*,*OF* ∗] **by** *auto*
moreover note *as*(*2*)[*unfolded fine-def*,*rule-format*,*OF xk*,*unfolded split-conv*]
note *q*(*3*)[*rule-format*,*OF xk*,*unfolded split-conv*,*rule-format*,*OF this*] **note** *this*[*unfolded n-def*[*symmetric*]]
moreover have *norm* (*content k* ∗$_R$ *f x*) ≤ (*real n* + *1*) ∗ (*content k* ∗ *indicator s x*)
proof(*cases x*∈*s*) **case** *False* **thus** *?thesis* **using** *assm* **by** *auto*
next case *True* **have** ∗:*content k* ≥ *0* **using** *tagged-division-ofD*(*4*)[*OF as*(*1*) *xk*] **by** *auto*
moreover have *content k* ∗ *norm* (*f x*) ≤ *content k* ∗ (*real n* + *1*) **apply**(*rule mult-mono*) **using** *nfx* ∗ **by** *auto*
ultimately show *?thesis* **unfolding** *abs-mult* **using** *nfx True* **by**(*auto simp add*:*field-simps*)
qed ultimately show ∃*y*. (*y*, *x*, *k*) ∈ {(*i*, *j*) |*i j*. *i* ∈ {*0*..*N* + *1*} ∧ *j* ∈ *q i*} ∧ *norm* (*content k* ∗$_R$ *f x*) ≤ (*real y* + *1*) ∗ (*content k* ∗$_R$ *indicator s x*)
apply(*rule-tac x*=*n* **in** *exI*,*safe*) **apply**(*rule-tac x*=*n* **in** *exI*,*rule-tac x*=(*x*,*k*) **in** *exI*,*safe*) **by** *auto*
qed(*insert as*, *auto*)
also have ... ≤ *setsum* (λ*i*. *e* / *2* / *2* ^ *i*) {*0*..*N*+*1*} **apply**(*rule setsum-mono*)

proof− **case** *goal1* **thus** *?case* **apply**(*subst mult-commute*, *subst pos-le-divide-eq*[*THEN sym*])
using *d*(*2*)[*rule-format*,*of q i i*] **using** *q*[*rule-format*] **by**(*auto simp add*:*field-simps*)
qed also have ... < *e* ∗ *inverse 2* ∗ *2* **unfolding** *divide-inverse setsum-right-distrib*[*THEN sym*]
apply(*rule mult-strict-left-mono*) **unfolding** *power-inverse atLeastLessThanSuc-atLeastAtMost*[*THEN sym*]
apply(*subst sumr-geometric*) **using** *goal1* **by** *auto*
finally show *?goal* **by** *auto* **qed qed qed**

lemma *has-integral-spike*: **fixes** *f*::*real^'n* ⇒ *'a*::*real-normed-vector*
assumes *negligible s* (∀ *x*∈(*t* − *s*). *g x* = *f x*) (*f has-integral y*) *t*
shows (*g has-integral y*) *t*
proof− **{ fix** *a b*::*real^'n* **and** *f g* ::*real^'n* ⇒ *'a* **and** *y*::*'a*
assume *as*:∀ *x* ∈ {*a*..*b*} − *s*. *g x* = *f x* (*f has-integral y*) {*a*..*b*}
have ((λ*x*. *f x* + (*g x* − *f x*)) *has-integral* (*y* + *0*)) {*a*..*b*} **apply**(*rule has-integral-add*[*OF as*(*2*)])
apply(*rule has-integral-negligible*[*OF assms*(*1*)]) **using** *as* **by** *auto*
hence (*g has-integral y*) {*a*..*b*} **by** *auto* **}** **note** ∗ = *this*
show *?thesis* **apply**(*subst has-integral-alt*) **using** *assms*(*2*−) **apply**−**apply**(*rule cond-cases*,*safe*)
apply(*rule* ∗, *assumption*+) **apply**(*subst*(*asm*) *has-integral-alt*) **unfolding** *if-not-P*
apply(*erule-tac x*=*e* **in** *allE*,*safe*,*rule-tac x*=*B* **in** *exI*,*safe*) **apply**(*erule-tac x*=*a* **in** *allE*,*erule-tac x*=*b* **in** *allE*,*safe*)
apply(*rule-tac x*=*z* **in** *exI*,*safe*) **apply**(*rule* ∗[**where** *fa2*=λ*x*. *if x*∈*t then f x else 0*]) **by** *auto* **qed**

lemma *has-integral-spike-eq*:
assumes *negligible s* ∀ *x*∈(*t* − *s*). *g x* = *f x*
shows ((*f has-integral y*) *t* ⟷ (*g has-integral y*) *t*)
apply *rule* **apply**(*rule-tac*[!] *has-integral-spike*[*OF assms*(*1*)]) **using** *assms*(*2*) **by** *auto*

lemma *integrable-spike*: **assumes** *negligible s* ∀ *x*∈(*t* − *s*). *g x* = *f x f integrable-on t*
shows *g integrable-on t*
using *assms* **unfolding** *integrable-on-def* **apply**−**apply**(*erule exE*)
apply(*rule*,*rule has-integral-spike*) **by** *fastsimp*+

lemma *integral-spike*: **assumes** *negligible s* ∀ *x*∈(*t* − *s*). *g x* = *f x*
shows *integral t f* = *integral t g*
unfolding *integral-def* **using** *has-integral-spike-eq*[*OF assms*] **by** *auto*

23.32 Some other trivialities about negligible sets.

lemma *negligible-subset*[*intro*]: **assumes** *negligible s t* $\subseteq$ *s* **shows** *negligible t* **unfolding** *negligible-def*
proof(*safe*) **case** *goal1* **show** *?case* **using** *assms*(*1*)[*unfolded negligible-def*,*rule-format*,*of a b*]
apply−**apply**(*rule has-integral-spike*[*OF assms*(*1*)]) **defer apply** *assumption*
using *assms*(*2*) **unfolding** *indicator-def* **by** *auto* **qed**

lemma *negligible-diff*[*intro?*]: **assumes** *negligible s* **shows** *negligible*(*s* − *t*) **using** *assms* **by** *auto*

lemma *negligible-inter*: **assumes** *negligible s* ∨ *negligible t* **shows** *negligible*(*s* ∩ *t*) **using** *assms* **by** *auto*

lemma *negligible-union*: **assumes** *negligible s negligible t* **shows** *negligible* (*s* ∪ *t*) **unfolding** *negligible-def*
proof *safe* **case** *goal1* **note** *assm* = *assms*[*unfolded negligible-def*,*rule-format*,*of a b*]
thus *?case* **apply**(*subst has-integral-spike-eq*[*OF assms*(*2*)])
defer apply *assumption* **unfolding** *indicator-def* **by** *auto* **qed**

lemma *negligible-union-eq*[*simp*]: *negligible* (*s* ∪ *t*) ⟷ (*negligible s* ∧ *negligible t*)
using *negligible-union* **by** *auto*

lemma *negligible-sing*[*intro*]: *negligible* {*a*::*real*^*'n*}
proof− **guess** *x* **using** *UNIV-witness*[**where** *'a*=*'n*] **..**
show *?thesis* **using** *negligible-standard-hyperplane*[*of x a*$*x*] **by** *auto* **qed**

lemma *negligible-insert*[*simp*]: *negligible*(*insert a s*) ⟷ *negligible s*
apply(*subst insert-is-Un*) **unfolding** *negligible-union-eq* **by** *auto*

lemma *negligible-empty*[*intro*]: *negligible* {} **by** *auto*

lemma *negligible-finite*[*intro*]: **assumes** *finite s* **shows** *negligible s*
using *assms* **apply**(*induct s*) **by** *auto*

lemma *negligible-unions*[*intro*]: **assumes** *finite s* ∀ *t*∈*s*. *negligible t* **shows** *negligible*(⋃ *s*)
using *assms* **by**(*induct*,*auto*)

lemma *negligible*: *negligible s* ⟷ (∀ *t*::(*real*^*'n*) *set*. (*indicator s has-integral 0*) *t*)
apply *safe* **defer apply**(*subst negligible-def*)
proof− **fix** *t*::(*real*^*'n*) *set* **assume** *as*:*negligible s*
have ∗:(λ*x*. *if x* ∈ *s* ∩ *t then 1 else 0*) = (λ*x*. *if x*∈*t then if x*∈*s then 1 else 0 else 0*) **by**(*rule ext*,*auto*)
show (*indicator s has-integral 0*) *t* **apply**(*subst has-integral-alt*)
apply(*cases*,*subst if-P*,*assumption*) **unfolding** *if-not-P* **apply**(*safe*,*rule as*[*unfolded*

negligible-def,*rule-format*])
apply(*rule-tac x=1* **in** *exI*) **apply**(*safe*,*rule zero-less-one*) **apply**(*rule-tac x=0* **in** *exI*)
using *negligible-subset*[*OF as*,*of s* ∩ *t*] **unfolding** *negligible-def indicator-def* **unfolding** ∗ **by** *auto* **qed** *auto*

23.33 Finite case of the spike theorem is quite commonly needed.

lemma *has-integral-spike-finite*: **assumes** *finite s* ∀ *x*∈*t*−*s*. *g x* = *f x* (*f has-integral y*) *t* **shows** (*g has-integral y*) *t*
apply(*rule has-integral-spike*) **using** *assms* **by** *auto*

lemma *has-integral-spike-finite-eq*: **assumes** *finite s* ∀ *x*∈*t*−*s*. *g x* = *f x*
shows ((*f has-integral y*) *t* ⟷ (*g has-integral y*) *t*)
apply *rule* **apply**(*rule-tac*[!] *has-integral-spike-finite*) **using** *assms* **by** *auto*

lemma *integrable-spike-finite*:
assumes *finite s* ∀ *x*∈*t*−*s*. *g x* = *f x f integrable-on t* **shows** *g integrable-on t*
using *assms* **unfolding** *integrable-on-def* **apply** *safe* **apply**(*rule-tac x=y* **in** *exI*)
apply(*rule has-integral-spike-finite*) **by** *auto*

23.34 In particular, the boundary of an interval is negligible.

lemma *negligible-frontier-interval*: *negligible*({*a*..*b*} − {*a*<..<*b*})
proof− **let** *?A* = ⋃((λ*k*. {*x*. *x*$*k* = *a*$*k*} ∪ {*x*. *x*$*k* = *b*$*k*}) ‘ *UNIV*)
have {*a*..*b*} − {*a*<..<*b*} ⊆ *?A* **apply** *rule* **unfolding** *Diff-iff mem-interval not-all*
apply(*erule conjE exE*)+ **apply**(*rule-tac X*={*x*. *x* $ *xa* = *a* $ *xa*} ∪ {*x*. *x* $ *xa* = *b* $ *xa*} **in** *UnionI*)
apply(*erule-tac*[!] *x*=*xa* **in** *allE*) **by** *auto*
thus *?thesis* **apply**−**apply**(*rule negligible-subset*[*of ?A*]) **apply**(*rule negligible-unions*[*OF finite-imageI*]) **by** *auto* **qed**

lemma *has-integral-spike-interior*:
assumes ∀ *x*∈{*a*<..<*b*}. *g x* = *f x* (*f has-integral y*) ({*a*..*b*}) **shows** (*g has-integral y*) ({*a*..*b*})
apply(*rule has-integral-spike*[*OF negligible-frontier-interval - assms*(*2*)]) **using** *assms*(*1*) **by** *auto*

lemma *has-integral-spike-interior-eq*:
assumes ∀ *x*∈{*a*<..<*b*}. *g x* = *f x* **shows** ((*f has-integral y*) ({*a*..*b*}) ⟷ (*g has-integral y*) ({*a*..*b*}))
apply *rule* **apply**(*rule-tac*[!] *has-integral-spike-interior*) **using** *assms* **by** *auto*

lemma *integrable-spike-interior*: **assumes** ∀ *x*∈{*a*<..<*b*}. *g x* = *f x f integrable-on* {*a*..*b*} **shows** *g integrable-on* {*a*..*b*}
using *assms* **unfolding** *integrable-on-def* **using** *has-integral-spike-interior*[*OF*

assms(1)] **by** *auto*

23.35 Integrability of continuous functions.

lemma *neutral-and*[*simp*]: *neutral op* $\wedge$ $=$ *True*
unfolding *neutral-def* **apply**(*rule some-equality*) **by** *auto*

lemma *monoidal-and*[*intro*]: *monoidal op* $\wedge$ **unfolding** *monoidal-def* **by** *auto*

lemma *iterate-and*[*simp*]: **assumes** *finite s* **shows** (*iterate op* $\wedge$) *s p* $\longleftrightarrow$ ($\forall$ *x*$\in$*s*. *p x*) **using** *assms*
apply *induct* **unfolding** *iterate-insert*[*OF monoidal-and*] **by** *auto*

lemma *operative-division-and*: **assumes** *operative op* $\wedge$ *P d division-of* {*a*..*b*}
shows ($\forall$ *i*$\in$*d*. *P i*) $\longleftrightarrow$ *P* {*a*..*b*}
using *operative-division*[*OF monoidal-and assms*] *division-of-finite*[*OF assms(2)*]
by *auto*

lemma *operative-approximable*: **assumes** *0* $\leq$ *e* **fixes** *f*::*real^'n* $\Rightarrow$ *'a*::*banach*
shows *operative op* $\wedge$ (λ*i*. $\exists$*g*. ($\forall$ *x*$\in$*i*. *norm* (*f x* $-$ *g* (*x*::*real^'n*)) $\leq$ *e*) $\wedge$ *g integrable-on i*) **unfolding** *operative-def neutral-and*
proof *safe* **fix** *a b*::*real^'n* **{ assume** *content* {*a*..*b*} $=$ *0*
thus $\exists$*g*. ($\forall$ *x*$\in${*a*..*b*}. *norm* (*f x* $-$ *g x*) $\leq$ *e*) $\wedge$ *g integrable-on* {*a*..*b*}
apply(*rule-tac x=f* **in** *exI*) **using** *assms* **by**(*auto intro*!:*integrable-on-null*) **}**
{ fix *c k g* **assume** *as*:$\forall$ *x*$\in${*a*..*b*}. *norm* (*f x* $-$ *g x*) $\leq$ *e g integrable-on* {*a*..*b*}
show $\exists$*g*. ($\forall$ *x*$\in${*a*..*b*} $\cap$ {*x*. *x* $ *k* $\leq$ *c*}. *norm* (*f x* $-$ *g x*) $\leq$ *e*) $\wedge$ *g integrable-on* {*a*..*b*} $\cap$ {*x*. *x* $ *k* $\leq$ *c*}
$\exists$*g*. ($\forall$ *x*$\in${*a*..*b*} $\cap$ {*x*. *c* $\leq$ *x* $ *k*}. *norm* (*f x* $-$ *g x*) $\leq$ *e*) $\wedge$ *g integrable-on* {*a*..*b*} $\cap$ {*x*. *c* $\leq$ *x* $ *k*}
apply(*rule-tac*[!] *x=g* **in** *exI*) **using** *as(1) integrable-split*[*OF as(2)*] **by** *auto*
}
fix *c k g1 g2* **assume** *as*:$\forall$ *x*$\in${*a*..*b*} $\cap$ {*x*. *x* $ *k* $\leq$ *c*}. *norm* (*f x* $-$ *g1 x*) $\leq$ *e g1 integrable-on* {*a*..*b*} $\cap$ {*x*. *x* $ *k* $\leq$ *c*}
$\forall$ *x*$\in${*a*..*b*} $\cap$ {*x*. *c* $\leq$ *x* $ *k*}. *norm* (*f x* $-$ *g2 x*) $\leq$ *e g2 integrable-on* {*a*..*b*} $\cap$ {*x*. *c* $\leq$ *x* $ *k*}
let *?g* $=$ λ*x*. *if x*$*k* $=$ *c then f x else if x*$*k* $\leq$ *c then g1 x else g2 x*
show $\exists$*g*. ($\forall$ *x*$\in${*a*..*b*}. *norm* (*f x* $-$ *g x*) $\leq$ *e*) $\wedge$ *g integrable-on* {*a*..*b*} **apply**(*rule-tac x=?g* **in** *exI*)
proof *safe* **case** *goal1* **thus** *?case* **apply**$-$ **apply**(*cases x*$*k*=*c*, *case-tac x*$*k* $<$ *c*) **using** *as assms* **by** *auto*
next case *goal2* **presume** *?g integrable-on* {*a*..*b*} $\cap$ {*x*. *x* $ *k* $\leq$ *c*} *?g integrable-on* {*a*..*b*} $\cap$ {*x*. *x* $ *k* $\geq$ *c*}
then guess *h1 h2* **unfolding** *integrable-on-def* **by** *auto* **from** *has-integral-split*[*OF this*]
show *?case* **unfolding** *integrable-on-def* **by** *auto*
next show *?g integrable-on* {*a*..*b*} $\cap$ {*x*. *x* $ *k* $\leq$ *c*} *?g integrable-on* {*a*..*b*} $\cap$ {*x*. *x* $ *k* $\geq$ *c*}
apply(*rule-tac*[!] *integrable-spike*[*OF negligible-standard-hyperplane*[*of k c*]])
using *as*(*2*,*4*) **by** *auto* **qed qed**

lemma *approximable-on-division*: **fixes** *f*::*real^'n ⇒ 'a*::*banach*
assumes *0 ≤ e d division-of {a..b} ∀ i∈d. ∃ g. (∀ x∈i. norm (f x − g x) ≤ e) ∧ g integrable-on i*
obtains *g* **where** *∀ x∈{a..b}. norm (f x − g x) ≤ e g integrable-on {a..b}*
proof− **note** *∗ = operative-division*[*OF monoidal-and operative-approximable*[*OF assms*(*1*)] *assms*(*2*)]
note *this*[*unfolded iterate-and*[*OF division-of-finite*[*OF assms*(*2*)]]] **from** *assms*(*3*)[*unfolded this*[*of f*]]
guess *g* **.. thus** *thesis* **apply**−**apply**(*rule that*[*of g*]) **by** *auto* **qed**

lemma *integrable-continuous*: **fixes** *f*::*real^'n ⇒ 'a*::*banach*
assumes *continuous-on {a..b} f* **shows** *f integrable-on {a..b}*
proof(*rule integrable-uniform-limit*,*safe*) **fix** *e*::*real* **assume** *e*:*0 < e*
from *compact-uniformly-continuous*[*OF assms compact-interval*,*unfolded uniformly-continuous-on-def*,*rule-fo e*] **guess** *d* **..**
note *d*=*conjunctD2*[*OF this*,*rule-format*]
from *fine-division-exists*[*OF gauge-ball*[*OF d*(*1*)], *of a b*] **guess** *p* **. note** *p*=*this*
note *p' = tagged-division-ofD*[*OF p*(*1*)]
have *∗*:*∀ i∈snd ‘ p. ∃ g. (∀ x∈i. norm (f x − g x) ≤ e) ∧ g integrable-on i*
proof(*safe*,*unfold snd-conv*) **fix** *x l* **assume** *as*:(*x*,*l*) *∈ p*
from *p'*(*4*)[*OF this*] **guess** *a b* **apply**−**by**(*erule exE*)+ **note** *l*=*this*
show *∃ g. (∀ x∈l. norm (f x − g x) ≤ e) ∧ g integrable-on l* **apply**(*rule-tac x*=*λy. f x* **in** *exI*)
proof *safe* **show** *(λy. f x) integrable-on l* **unfolding** *integrable-on-def l* **by**(*rule*,*rule has-integral-const*)
fix *y* **assume** *y*:*y∈l* **note** *fineD*[*OF p*(*2*) *as*,*unfolded subset-eq*,*rule-format*,*OF this*]
note *d*(*2*)[*OF - - this*[*unfolded mem-ball*]]
thus *norm (f y − f x) ≤ e* **using** *y p'*(*2−3*)[*OF as*] **unfolding** *dist-norm l norm-minus-commute* **by** *fastsimp* **qed qed**
from *e* **have** *0 ≤ e* **by** *auto* **from** *approximable-on-division*[*OF this division-of-tagged-division*[*OF p*(*1*)] *∗*] **guess** *g* **.**
thus *∃ g. (∀ x∈{a..b}. norm (f x − g x) ≤ e) ∧ g integrable-on {a..b}* **by** *auto*
qed

23.36 Specialization of additivity to one dimension.

lemma *operative-1-lt*: **assumes** *monoidal opp*
shows *operative opp f ⟷ ((∀ a b. b ≤ a ⟶ f {a..b::real^1} = neutral opp) ∧ (∀ a b c. a < c ∧ c < b ⟶ opp (f{a..c})(f{c..b}) = f {a..b}))*
unfolding *operative-def content-eq-0-1 forall-1 vector-le-def vector-less-def*
proof *safe* **fix** *a b c*::*real^1* **assume** *as*:*∀ a b c. f {a..b} = opp (f ({a..b} ∩ {x. x $ 1 ≤ c})) (f ({a..b} ∩ {x. c ≤ x $ 1})) a $ 1 < c $ 1 c $ 1 < b $ 1*
from *this*(*2−*) **have** *{a..b} ∩ {x. x $ 1 ≤ c $ 1} = {a..c} {a..b} ∩ {x. x $ 1 ≥ c $ 1} = {c..b}* **by** *auto*
thus *opp (f {a..c}) (f {c..b}) = f {a..b}* **unfolding** *as*(*1*)[*rule-format*,*of a b c*$*1*] **by** *auto*
next fix *a b*::*real^1* **and** *c*::*real*

assume *as*:$\forall\ a\ b.\ b\ \$\ 1 \leq a\ \$\ 1 \longrightarrow f\ \{a..b\} = \mathit{neutral}\ \mathit{opp}\ \forall\ a\ b\ c.\ a\ \$\ 1 < c\ \$\ 1 \wedge c\ \$\ 1 < b\ \$\ 1 \longrightarrow \mathit{opp}\ (f\ \{a..c\})\ (f\ \{c..b\}) = f\ \{a..b\}$
show $f\ \{a..b\} = \mathit{opp}\ (f\ (\{a..b\} \cap \{x.\ x\ \$\ 1 \leq c\}))\ (f\ (\{a..b\} \cap \{x.\ c \leq x\ \$\ 1\}))$
proof(*cases* $c \in \{a\$1\ ..\ b\$1\}$)
case *False* **hence** $c<a\$1 \vee c>b\$1$ **by** *auto*
thus *?thesis* **apply**−**apply**(*erule disjE*)
proof− **assume** $c<a\$1$ **hence** $*:\{a..b\} \cap \{x.\ x\ \$\ 1 \leq c\} = \{1..0\}\ \ \{a..b\} \cap \{x.\ c \leq x\ \$\ 1\} = \{a..b\}$ **by** *auto*
show *?thesis* **unfolding** $*$ **apply**(*subst as*(*1*)[*rule-format*,*of 0 1*]) **using** *assms* **by** *auto*
next **assume** $b\$1<c$ **hence** $*:\{a..b\} \cap \{x.\ x\ \$\ 1 \leq c\} = \{a..b\}\ \ \{a..b\} \cap \{x.\ c \leq x\ \$\ 1\} = \{1..0\}$ **by** *auto*
show *?thesis* **unfolding** $*$ **apply**(*subst as*(*1*)[*rule-format*,*of 0 1*]) **using** *assms* **by** *auto*
qed
next case *True* **hence** $*:\mathit{min}\ (b\ \$\ 1)\ c = c\ \mathit{max}\ (a\ \$\ 1)\ c = c$ **by** *auto*
show *?thesis* **unfolding** *interval-split num1-eq-iff if-True* $*$ *vec-def*[*THEN sym*]
proof(*cases* $c = a\$1 \vee c = b\$1$)
case *False* **thus** $f\ \{a..b\} = \mathit{opp}\ (f\ \{a..\mathit{vec1}\ c\})\ (f\ \{\mathit{vec1}\ c..b\})$
apply−**apply**(*subst as*(*2*)[*rule-format*]) **using** *True* **by** *auto*
next case *True* **thus** $f\ \{a..b\} = \mathit{opp}\ (f\ \{a..\mathit{vec1}\ c\})\ (f\ \{\mathit{vec1}\ c..b\})$ **apply**−
proof(*erule disjE*) **assume** $c=a\$1$ **hence** $*:a = \mathit{vec1}\ c$ **unfolding** *Cart-eq* **by** *auto*
hence $f\ \{a..\mathit{vec1}\ c\} = \mathit{neutral}\ \mathit{opp}$ **apply**−**apply**(*rule as*(*1*)[*rule-format*]) **by** *auto*
thus *?thesis* **using** *assms* **unfolding** $*$ **by** *auto*
next assume $c=b\$1$ **hence** $*:b = \mathit{vec1}\ c$ **unfolding** *Cart-eq* **by** *auto*
hence $f\ \{\mathit{vec1}\ c..b\} = \mathit{neutral}\ \mathit{opp}$ **apply**−**apply**(*rule as*(*1*)[*rule-format*]) **by** *auto*
thus *?thesis* **using** *assms* **unfolding** $*$ **by** *auto* **qed qed qed qed**

lemma *operative-1-le*: **assumes** *monoidal opp*
shows *operative opp f* $\longleftrightarrow$ $((\forall\ a\ b.\ b \leq a \longrightarrow f\ \{a..b::\mathit{real\hat{}1}\} = \mathit{neutral}\ \mathit{opp}) \wedge (\forall\ a\ b\ c.\ a \leq c \wedge c \leq b \longrightarrow \mathit{opp}\ (f\{a..c\})(f\{c..b\}) = f\ \{a..b\}))$
unfolding *operative-1-lt*[*OF assms*]
proof *safe* **fix** *a b c*::*real^1* **assume** *as*:$\forall\ a\ b\ c.\ a \leq c \wedge c \leq b \longrightarrow \mathit{opp}\ (f\ \{a..c\})\ (f\ \{c..b\}) = f\ \{a..b\}\ a < c\ c < b$
show $\mathit{opp}\ (f\ \{a..c\})\ (f\ \{c..b\}) = f\ \{a..b\}$ **apply**(*rule as*(*1*)[*rule-format*]) **using** *as*(*2*−) **unfolding** *vector-le-def vector-less-def* **by** *auto*
next fix *a b c* ::*real^1*
assume $\forall\ a\ b.\ b \leq a \longrightarrow f\ \{a..b\} = \mathit{neutral}\ \mathit{opp}\ \forall\ a\ b\ c.\ a < c \wedge c < b \longrightarrow \mathit{opp}\ (f\ \{a..c\})\ (f\ \{c..b\}) = f\ \{a..b\}\ a \leq c\ c \leq b$
note *as* = *this*[*rule-format*]
show $\mathit{opp}\ (f\ \{a..c\})\ (f\ \{c..b\}) = f\ \{a..b\}$
proof(*cases* $c = a \vee c = b$)
case *False* **thus** *?thesis* **apply**−**apply**(*subst as*(*2*)) **using** *as*(*3*−) **unfolding** *vector-le-def vector-less-def Cart-eq* **by**(*auto simp del*:*dest-vec1-eq*)
next case *True* **thus** *?thesis* **apply**−

proof(*erule disjE*) **assume** ∗:*c*=*a* **hence** *f* {*a*..*c*} = *neutral opp* **apply**−**apply**(*rule as*(*1*)[*rule-format*]) **by** *auto*
thus *?thesis* **using** *assms* **unfolding** ∗ **by** *auto*
next **assume** ∗:*c*=*b* **hence** *f* {*c*..*b*} = *neutral opp* **apply**−**apply**(*rule as*(*1*)[*rule-format*]) **by** *auto*
thus *?thesis* **using** *assms* **unfolding** ∗ **by** *auto* **qed qed qed**

23.37 Special case of additivity we need for the FCT.

lemma *interval-bound-sing*[*simp*]: *interval-upperbound* {*a*} = *a interval-lowerbound* {*a*} = *a*
unfolding *interval-upperbound-def interval-lowerbound-def* **unfolding** *Cart-eq* **by** *auto*

lemma *additive-tagged-division-1*: **fixes** *f*::*real^1* ⇒ *'a*::*real-normed-vector*
assumes *dest-vec1 a* ≤ *dest-vec1 b p tagged-division-of* {*a*..*b*}
shows *setsum* (λ(*x*,*k*). *f*(*interval-upperbound k*) − *f*(*interval-lowerbound k*)) *p* = *f b* − *f a*
proof− **let** *?f* = (λ*k*::(*real^1*) *set*. *if k* = {} *then 0 else f*(*interval-upperbound k*) − *f*(*interval-lowerbound k*))
have ∗:*operative op* + *?f* **unfolding** *operative-1-lt*[*OF monoidal-monoid*] *interval-eq-empty-1* **by**(*auto simp add*:*not-less vector-less-def*)
have ∗∗:{*a*..*b*} ≠ {} **using** *assms*(*1*) **by** *auto* **note** *operative-tagged-division*[*OF monoidal-monoid* ∗ *assms*(*2*)]
note ∗ = *this*[*unfolded if-not-P*[*OF* ∗∗] *interval-bound-1*[*OF assms*(*1*)],*THEN sym*]
show *?thesis* **unfolding** ∗ **apply**(*subst setsum-iterate*[*THEN sym*]) **defer**
apply(*rule setsum-cong2*) **unfolding** *split-paired-all split-conv* **using** *assms*(*2*) **by** *auto* **qed**

23.38 A useful lemma allowing us to factor out the content size.

lemma *has-integral-factor-content*:
(*f has-integral i*) {*a*..*b*} ⟷ (∀ *e*>*0*. ∃ *d*. *gauge d* ∧ (∀ *p*. *p tagged-division-of* {*a*..*b*} ∧ *d fine p*
⟶ *norm* (*setsum* (λ(*x*,*k*). *content k* ∗*R f x*) *p* − *i*) ≤ *e* ∗ *content* {*a*..*b*}))
proof(*cases content* {*a*..*b*} = *0*)
case *True* **show** *?thesis* **unfolding** *has-integral-null-eq*[*OF True*] **apply** *safe*
apply(*rule*,*rule*,*rule gauge-trivial*,*safe*) **unfolding** *setsum-content-null*[*OF True*] *True* **defer**
apply(*erule-tac x*=*1* **in** *allE*,*safe*) **defer apply**(*rule fine-division-exists*[*of - a b*],*assumption*)
apply(*erule-tac x*=*p* **in** *allE*) **unfolding** *setsum-content-null*[*OF True*] **by** *auto*
next case *False* **note** *F* = *this*[*unfolded content-lt-nz*[*THEN sym*]]
let *?P* = λ*e opp*. ∃ *d*. *gauge d* ∧ (∀ *p*. *p tagged-division-of* {*a*..*b*} ∧ *d fine p* ⟶ *opp* (*norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗*R f x*) − *i*)) *e*)
show *?thesis* **apply**(*subst has-integral*)
proof *safe* **fix** *e*::*real* **assume** *e*:*e*>*0*

{ assume $\forall\, e{>}0.$ *?P e op* $<$ **thus** *?P* $(e * content\ \{a..b\})$ *op* $\leq$ **apply**(*erule-tac x=e * content {a..b}* **in** *allE*)
apply(*erule impE*) **defer apply**(*erule exE,rule-tac x=d* **in** *exI*)
using *F e* **by**(*auto simp add:field-simps intro:mult-pos-pos*) **}**
{ assume $\forall\, e{>}0.$ *?P* $(e * content\ \{a..b\})$ *op* $\leq$ **thus** *?P e op* $<$ **apply**(*erule-tac x=e / 2 / content {a..b}* **in** *allE*)
apply(*erule impE*) **defer apply**(*erule exE,rule-tac x=d* **in** *exI*)
using *F e* **by**(*auto simp add:field-simps intro:mult-pos-pos*) **} qed qed**

23.39 Fundamental theorem of calculus.

lemma *fundamental-theorem-of-calculus*: **fixes** *f*::*real^1* $\Rightarrow$ *'a*::*banach*
assumes $a \leq b$ $\forall x{\in}\{a..b\}.$ $((f\ o\ vec1)$ *has-vector-derivative f'*$(vec1\ x))$ $(at\ x\ within\ \{a..b\})$
shows $(f'$ *has-integral* $(f(vec1\ b) - f(vec1\ a)))$ $(\{vec1\ a..vec1\ b\})$
unfolding *has-integral-factor-content*
proof *safe* **fix** *e*::*real* **assume** *e*:*e>0* **have** *ab*:*dest-vec1* $(vec1\ a) \leq$ *dest-vec1* $(vec1\ b)$ **using** *assms(1)* **by** *auto*
note *assm* = *assms(2)*[*unfolded has-vector-derivative-def has-derivative-within-alt*]
have $*{:}\bigwedge P\ Q.\ \forall x{\in}\{a..b\}.\ P\ x \wedge (\forall\, e{>}0.\ \exists\, d{>}0.\ Q\ x\ e\ d) \Longrightarrow \forall x.\ \exists (d{::}real){>}0.\ x{\in}\{a..b\} \longrightarrow Q\ x\ e\ d$ **using** *e* **by** *blast*
note *this*[*OF assm*,*unfolded gauge-existence-lemma*] **from** *choice*[*OF this*,*unfolded Ball-def*[*symmetric*]]
guess *d* **.. note** *d*=*conjunctD2*[*OF this*[*rule-format*],*rule-format*]
show $\exists\, d.$ *gauge d* $\wedge$ $(\forall\, p.$ *p tagged-division-of* $\{vec1\ a..vec1\ b\} \wedge$ *d fine p* $\longrightarrow$
$norm\ ((\sum (x,\ k){\in}p.\ content\ k *_R f'\ x) - (f\ (vec1\ b) - f\ (vec1\ a))) \leq e * content\ \{vec1\ a..vec1\ b\})$
apply(*rule-tac x*=$\lambda x.$ *ball x* $(d\ (dest\text{-}vec1\ x))$ **in** *exI*,*safe*)
apply(*rule gauge-ball-dependent*,*rule*,*rule d(1)*)
proof− **fix** *p* **assume** *as*:*p tagged-division-of* $\{vec1\ a..vec1\ b\}$ $(\lambda x.\ ball\ x\ (d\ (dest\text{-}vec1\ x)))$ *fine p*
show $norm\ ((\sum (x,\ k){\in}p.\ content\ k *_R f'\ x) - (f\ (vec1\ b) - f\ (vec1\ a))) \leq e * content\ \{vec1\ a..vec1\ b\}$
unfolding *content-1*[*OF ab*] *additive-tagged-division-1*[*OF ab as(1)*,*of f*,*THEN sym*]
unfolding *vector-minus-component*[*THEN sym*] *additive-tagged-division-1*[*OF ab as(1)*,*of* $\lambda x.\ x$,*THEN sym*]
apply(*subst dest-vec1-setsum*) **unfolding** *setsum-right-distrib* **defer unfolding** *setsum-subtractf*[*THEN sym*]
proof(*rule setsum-norm-le*,*safe*) **fix** *x k* **assume** (*x*,*k*)∈*p*
note *xk* = *tagged-division-ofD(2−4)*[*OF as(1) this*] **from** *this(3)* **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *k*=*this*
have $*{:}$*dest-vec1 u* $\leq$ *dest-vec1 v* **using** *xk* **unfolding** *k* **by** *auto*
have *ball*:$\forall$ *xa*∈*k*. *xa* ∈ *ball x* (*d* (*dest-vec1 x*)) **using** *as(2)*[*unfolded fine-def*,*rule-format*,*OF* ‹(*x*,*k*)∈*p*›,*unfolded split-conv subset-eq*] **.**
have $norm\ ((v\$1 - u\$1) *_R f'\ x - (f\ v - f\ u)) \leq norm\ (f\ u - f\ x - (u\$1 - x\$1) *_R f'\ x) + norm\ (f\ v - f\ x - (v\$1 - x\$1) *_R f'\ x)$
apply(*rule order-trans*[*OF - norm-triangle-ineq4*]) **apply**(*rule eq-refl*) **apply**(*rule arg-cong*[**where** *f*=*norm*])

unfolding *scaleR.diff-left* **by**(*auto simp add*:*algebra-simps*)
also have ... ≤ *e* ∗ *norm* (*dest-vec1 u* − *dest-vec1 x*) + *e* ∗ *norm* (*dest-vec1 v* − *dest-vec1 x*)
apply(*rule add-mono*) **apply**(*rule d*(*2*)[*of x*$*1 u*$*1*,*unfolded o-def vec1-dest-vec1*])
prefer *4*
apply(*rule d*(*2*)[*of x*$*1 v*$*1*,*unfolded o-def vec1-dest-vec1*])
using *ball*[*rule-format*,*of u*] *ball*[*rule-format*,*of v*]
using *xk*(*1*−*2*) **unfolding** *k subset-eq* **by**(*auto simp add*:*dist-norm norm-real*)
also have ... ≤ *e* ∗ *dest-vec1* (*interval-upperbound k* − *interval-lowerbound k*)
unfolding *k interval-bound-1*[*OF* ∗] **using** *xk*(*1*) **unfolding** *k* **by**(*auto simp add*:*dist-norm norm-real field-simps*)
finally show *norm* (*content k* ∗$_R$ *f* ′ *x* − (*f* (*interval-upperbound k*) − *f* (*interval-lowerbound k*))) ≤
e ∗ *dest-vec1* (*interval-upperbound k* − *interval-lowerbound k*) **unfolding** *k interval-bound-1*[*OF* ∗] *content-1*[*OF* ∗] **.**
qed(*insert as*, *auto*) **qed qed**

23.40 Attempt a systematic general set of "offset" results for components.

lemma *gauge-modify*:
assumes (∀ *s*. *open s* ⟶ *open* {*x*. *f*(*x*) ∈ *s*}) *gauge d*
shows *gauge* (λ*x y*. *d* (*f x*) (*f y*))
using *assms* **unfolding** *gauge-def* **apply** *safe* **defer apply**(*erule-tac x*=*f x* **in** *allE*)
apply(*erule-tac x*=*d* (*f x*) **in** *allE*) **unfolding** *mem-def Collect-def* **by** *auto*

23.41 Only need trivial subintervals if the interval itself is trivial.

lemma *division-of-nontrivial*: **fixes** *s*::(*real*^′*n*) *set set*
assumes *s division-of* {*a*..*b*} *content*({*a*..*b*}) ≠ *0*
shows {*k*. *k* ∈ *s* ∧ *content k* ≠ *0*} *division-of* {*a*..*b*} **using** *assms*(*1*) **apply**−**proof**(*induct card s arbitrary*:*s rule*:*nat-less-induct*)
fix *s*::(*real*^′*n*) *set set* **assume** *assm*:*s division-of* {*a*..*b*}
∀ *m*<*card s*. ∀ *x*. *m* = *card x* ⟶ *x division-of* {*a*..*b*} ⟶ {*k* ∈ *x*. *content k* ≠ *0*} *division-of* {*a*..*b*}
note *s* = *division-ofD*[*OF assm*(*1*)] **let** *?thesis* = {*k* ∈ *s*. *content k* ≠ *0*} *division-of* {*a*..*b*}
{ presume ∗:{*k* ∈ *s*. *content k* ≠ *0*} ≠ *s* ⟹ *?thesis*
show *?thesis* **apply** *cases* **defer apply**(*rule* ∗,*assumption*) **using** *assm*(*1*) **by** *auto* **}**
assume *noteq*:{*k* ∈ *s*. *content k* ≠ *0*} ≠ *s*
then obtain *k* **where** *k*:*k*∈*s content k* = *0* **by** *auto*
from *s*(*4*)[*OF k*(*1*)] **guess** *c d* **apply**−**by**(*erule exE*)+ **note** *k*=*k this*
from *k* **have** *card s* > *0* **unfolding** *card-gt-0-iff* **using** *assm*(*1*) **by** *auto*
hence *card*:*card* (*s* − {*k*}) < *card s* **using** *assm*(*1*) *k*(*1*) **apply**(*subst card-Diff-singleton-if*) **by** *auto*

have ∗:*closed* (⋃(*s* − {*k*})) **apply**(*rule closed-Union*) **defer apply** *rule* **apply**(*drule DiffD1*,*drule s*(*4*))
apply *safe* **apply**(*rule closed-interval*) **using** *assm*(*1*) **by** *auto*
have *k* ⊆ ⋃(*s* − {*k*}) **apply** *safe* **apply**(*rule* ∗[*unfolded closed-limpt*,*rule-format*]) **unfolding** *islimpt-approachable*
proof *safe* **fix** *x* **and** *e*::*real* **assume** *as*:*x*∈*k* *e*>*0*
from *k*(*2*)[*unfolded k content-eq-0*] **guess** *i* **..**
hence *i*:*c*$*i* = *d*$*i* **using** *s*(*3*)[*OF k*(*1*),*unfolded k*] **unfolding** *interval-ne-empty* **by** *smt*
hence *xi*:*x*$*i* = *d*$*i* **using** *as* **unfolding** *k mem-interval* **by** *smt*
def *y* ≡ (χ *j*. *if j* = *i then if c*$*i* ≤ (*a*$*i* + *b*$*i*) / *2 then c*$*i* + *min e* (*b*$*i* − *c*$*i*) / *2 else c*$*i* − *min e* (*c*$*i* − *a*$*i*) / *2 else x*$*j*)
show ∃*x*′∈⋃(*s* − {*k*}). *x*′ ≠ *x* ∧ *dist x*′ *x* < *e* **apply**(*rule-tac x*=*y* **in** *bexI*)
proof have *d* ∈ {*c*..*d*} **using** *s*(*3*)[*OF k*(*1*)] **unfolding** *k interval-eq-empty mem-interval* **by**(*fastsimp simp add*: *not-less*)
hence *d* ∈ {*a*..*b*} **using** *s*(*2*)[*OF k*(*1*)] **unfolding** *k* **by** *auto* **note** *di* = *this*[*unfolded mem-interval*,*THEN spec*[**where** *x*=*i*]]
hence *xyi*:*y*$*i* ≠ *x*$*i* **unfolding** *y-def* **unfolding** *i xi Cart-lambda-beta if-P*[*OF refl*]
apply(*cases*) **apply**(*subst if-P*,*assumption*) **unfolding** *if-not-P not-le* **using** *as*(*2*) **using** *assms*(*2*)[*unfolded content-eq-0*] **by** *smt*+
thus *y* ≠ *x* **unfolding** *Cart-eq* **by** *auto*
have ∗:*UNIV* = *insert i* (*UNIV* − {*i*}) **by** *auto*
have *norm* (*y* − *x*) < *e* + *setsum* (λ*i*. *0*) (*UNIV*::′*n set*) **apply**(*rule le-less-trans*[*OF norm-le-l1*])
apply(*subst* ∗,*subst setsum-insert*) **prefer** *3* **apply**(*rule add-less-le-mono*)
proof− **show** |(*y* − *x*) $ *i*| < *e* **unfolding** *y-def Cart-lambda-beta vector-minus-component if-P*[*OF refl*]
apply(*cases*) **apply**(*subst if-P*,*assumption*) **unfolding** *if-not-P* **unfolding** *i xi* **using** *di as*(*2*) **by** *auto*
show (∑ *i*∈*UNIV* − {*i*}. |(*y* − *x*) $ *i*|) ≤ (∑ *i*∈*UNIV*. *0*) **unfolding** *y-def* **by** *auto*
qed *auto* **thus** *dist y x* < *e* **unfolding** *dist-norm* **by** *auto*
have *y*∉*k* **unfolding** *k mem-interval* **apply** *rule* **apply**(*erule-tac x*=*i* **in** *allE*) **using** *xyi* **unfolding** *k i xi* **by** *auto*
moreover have *y* ∈ ⋃*s* **unfolding** *s mem-interval*
proof note *simps* = *y-def Cart-lambda-beta if-not-P*
fix *j*::′*n* **show** *a* $ *j* ≤ *y* $ *j* ∧ *y* $ *j* ≤ *b* $ *j*
proof(*cases j* = *i*) **case** *False* **have** *x* ∈ {*a*..*b*} **using** *s*(*2*)[*OF k*(*1*)] *as*(*1*) **by** *auto*
thus *?thesis* **unfolding** *simps if-not-P*[*OF False*] **unfolding** *mem-interval* **by** *auto*
next case *True* **note** *T* = *this* **show** *?thesis*
proof(*cases c* $ *i* ≤ (*a* $ *i* + *b* $ *i*) / *2*)
case *True* **show** *?thesis* **unfolding** *simps if-P*[*OF T*] *if-P*[*OF True*] **unfolding** *i*
using *True as*(*2*) *di* **apply**−**apply** *rule* **unfolding** *T* **by** (*auto simp add*:*field-simps*)
next case *False* **thus** *?thesis* **unfolding** *simps if-P*[*OF T*] *if-not-P*[*OF*

False] **unfolding** *i*
using *True as(2) di* **apply−apply** *rule* **unfolding** *T* **by** (*auto simp add*:*field-simps*)
qed qed qed
ultimately show $y \in \bigcup(s - \{k\})$ **by** *auto*
qed qed hence $\bigcup(s - \{k\}) = \{a..b\}$ **unfolding** *s(6)*[*THEN sym*] **by** *auto*
hence $\{ka \in s - \{k\}.\ content\ ka \neq 0\}$ *division-of* $\{a..b\}$ **apply−apply**(*rule assm(2)*[*rule-format*,*OF card refl*])
apply(*rule division-ofI*) **defer apply**(*rule-tac*[*1−4*] *s*) **using** *assm(1)* **by** *auto*
moreover have $\{ka \in s - \{k\}.\ content\ ka \neq 0\} = \{k \in s.\ content\ k \neq 0\}$ **using** *k* **by** *auto* **ultimately show** *?thesis* **by** *auto* **qed**

23.42 Integrabibility on subintervals.

lemma *operative-integrable*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach* **shows**
operative op ∧ (λ*i*. *f integrable-on i*)
unfolding *operative-def neutral-and* **apply** *safe* **apply**(*subst integrable-on-def*)
unfolding *has-integral-null-eq* **apply**(*rule*,*rule refl*) **apply**(*rule*,*assumption*)+
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-split*)

lemma *integrable-subinterval*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *f integrable-on* $\{a..b\}$ $\{c..d\} \subseteq \{a..b\}$ **shows** *f integrable-on* $\{c..d\}$
apply(*cases* $\{c..d\} = \{\}$) **defer apply**(*rule partial-division-extend-1*[*OF assms(2)*],*assumption*)
using *operative-division-and*[*OF operative-integrable*,*THEN sym*,*of - - - f*] *assms(1)*
by *auto*

23.43 Combining adjacent intervals in 1 dimension.

lemma *has-integral-combine*: **assumes** (*a*::*real^1*) ≤ *c c* ≤ *b*
(*f has-integral i*) $\{a..c\}$ (*f has-integral* (*j*::*'a*::*banach*)) $\{c..b\}$
shows (*f has-integral* (*i* + *j*)) $\{a..b\}$
proof− **note** *operative-integral*[*of f*, *unfolded operative-1-le*[*OF monoidal-lifted*[*OF monoidal-monoid*]]]
note *conjunctD2*[*OF this*,*rule-format*] **note** ∗ = *this(2)*[*OF conjI*[*OF assms(1−2)*],*unfolded if-P*[*OF assms(3)*]]
hence *f integrable-on* $\{a..b\}$ **apply−** **apply**(*rule ccontr*) **apply**(*subst*(*asm*) *if-P*)
defer
apply(*subst*(*asm*) *if-P*) **using** *assms(3−)* **by** *auto*
with ∗ **show** *?thesis* **apply−apply**(*subst*(*asm*) *if-P*) **defer apply**(*subst*(*asm*) *if-P*) **defer apply**(*subst*(*asm*) *if-P*)
unfolding *lifted.simps* **using** *assms(3−)* **by**(*auto simp add*: *integrable-on-def integral-unique*) **qed**

lemma *integral-combine*: **fixes** *f*::*real^1* ⇒ *'a*::*banach*
assumes *a* ≤ *c c* ≤ *b f integrable-on* ($\{a..b\}$)
shows *integral* $\{a..c\}$ *f* + *integral* $\{c..b\}$ *f* = *integral*($\{a..b\}$) *f*
apply(*rule integral-unique*[*THEN sym*]) **apply**(*rule has-integral-combine*[*OF assms(1−2)*])
apply(*rule-tac*[!] *integrable-integral integrable-subinterval*[*OF assms(3)*])+ **using** *assms(1−2)* **by** *auto*

lemma *integrable-combine*: **fixes** *f*::*real^1* ⇒ *'a*::*banach*
assumes *a* ≤ *c c* ≤ *b f integrable-on* {*a*..*c*} *f integrable-on* {*c*..*b*}
shows *f integrable-on* {*a*..*b*} **using** *assms* **unfolding** *integrable-on-def* **by**(*fastsimp intro*!:*has-integral-combine*)

23.44 Reduce integrability to "local" integrability.

lemma *integrable-on-little-subintervals*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes ∀ *x*∈{*a*..*b*}. ∃ *d*>*0*. ∀ *u v*. *x* ∈ {*u*..*v*} ∧ {*u*..*v*} ⊆ *ball x d* ∧ {*u*..*v*} ⊆ {*a*..*b*} ⟶ *f integrable-on* {*u*..*v*}
shows *f integrable-on* {*a*..*b*}
proof− **have** ∀ *x*. ∃ *d*. *x*∈{*a*..*b*} ⟶ *d*>*0* ∧ (∀ *u v*. *x* ∈ {*u*..*v*} ∧ {*u*..*v*} ⊆ *ball x d* ∧ {*u*..*v*} ⊆ {*a*..*b*} ⟶ *f integrable-on* {*u*..*v*})
using *assms* **by** *auto* **note** *this*[*unfolded gauge-existence-lemma*] **from** *choice*[*OF this*] **guess** *d* **.. note** *d*=*this*[*rule-format*]
guess *p* **apply**(*rule fine-division-exists*[*OF gauge-ball-dependent*,*of d a b*]) **using** *d* **by** *auto* **note** *p*=*this*(*1*−*2*)
note *division-of-tagged-division*[*OF this*(*1*)] **note** ∗ = *operative-division-and*[*OF operative-integrable*,*OF this*,*THEN sym*,*of f*]
show *?thesis* **unfolding** ∗ **apply** *safe* **unfolding** *snd-conv*
proof− **fix** *x k* **assume** (*x*,*k*) ∈ *p* **note** *tagged-division-ofD*(*2*−*4*)[*OF p*(*1*) *this*] *fineD*[*OF p*(*2*) *this*]
thus *f integrable-on k* **apply** *safe* **apply**(*rule d*[*THEN conjunct2*,*rule-format*,*of x*]) **by** *auto* **qed qed**

23.45 Second FCT or existence of antiderivative.

lemma *integrable-const*[*intro*]:(λ*x*. *c*) *integrable-on* {*a*..*b*}
unfolding *integrable-on-def* **by**(*rule*,*rule has-integral-const*)

lemma *integral-has-vector-derivative*: **fixes** *f*::*real* ⇒ *'a*::*banach*
assumes *continuous-on* {*a*..*b*} *f x* ∈ {*a*..*b*}
shows ((λ*u*. *integral* {*vec a*..*vec u*} (*f o dest-vec1*)) *has-vector-derivative f*(*x*)) (*at x within* {*a*..*b*})
unfolding *has-vector-derivative-def has-derivative-within-alt*
apply *safe* **apply**(*rule scaleR.bounded-linear-left*)
proof− **fix** *e*::*real* **assume** *e*:*e*>*0*
note *compact-uniformly-continuous*[*OF assms*(*1*) *compact-real-interval*,*unfolded uniformly-continuous-on-def*]
from *this*[*rule-format*,*OF e*] **guess** *d* **apply**−**by**(*erule conjE exE*)+ **note** *d*=*this*[*rule-format*]
let *?I* = λ*a b*. *integral* {*vec1 a*..*vec1 b*} (*f* ∘ *dest-vec1*)
show ∃ *d*>*0*. ∀ *y*∈{*a*..*b*}. *norm* (*y* − *x*) < *d* ⟶ *norm* (*?I a y* − *?I a x* − (*y* − *x*) ∗*R f x*) ≤ *e* ∗ *norm* (*y* − *x*)
proof(*rule*,*rule*,*rule d*,*safe*) **case** *goal1* **show** *?case* **proof**(*cases y* < *x*)
case *False* **have** *f* ∘ *dest-vec1 integrable-on* {*vec1 a*..*vec1 y*} **apply**(*rule integrable-subinterval*,*rule integrable-continuous*)
apply(*rule continuous-on-o-dest-vec1 assms*)+ **unfolding** *not-less* **using** *assms*(*2*) *goal1* **by** *auto*
hence ∗:*?I a y* − *?I a x* = *?I x y* **unfolding** *algebra-simps* **apply**(*subst eq-commute*) **apply**(*rule integral-combine*)

using *False* **unfolding** *not-less* **using** *assms(2) goal1* **by** *auto*
have $**$:*norm* $(y - x)$ = *content* $\{vec1\ x..vec1\ y\}$ **apply**(*subst content-1*) **using** *False* **unfolding** *not-less* **by** *auto*
show *?thesis* **unfolding** $**$ **apply**(*rule has-integral-bound*[**where** f=$(\lambda u.\ f\ u - f\ x)$ *o dest-vec1*]) **unfolding** $*$ **unfolding** *o-def*
defer apply(*rule has-integral-sub*) **apply**(*rule integrable-integral*)
apply(*rule integrable-subinterval*,*rule integrable-continuous*) **apply**(*rule continuous-on-o-dest-vec1*[*unfolded o-def*] *assms*)+
proof− **show** $\{vec1\ x..vec1\ y\} \subseteq \{vec1\ a..vec1\ b\}$ **using** *goal1 assms(2)* **by** *auto*
have $*$:$y - x = norm(y - x)$ **using** *False* **by** *auto*
show $((\lambda xa.\ f\ x)$ *has-integral* $(y - x) *_R f\ x)\ \{vec1\ x..vec1\ y\}$ **apply**(*subst* $*$) **unfolding** $**$ **by** *auto*
show $\forall xa{\in}\{vec1\ x..vec1\ y\}.$ *norm* $(f\ (dest\text{-}vec1\ xa) - f\ x) \leq e$ **apply** *safe* **apply**(*rule less-imp-le*)
apply(*rule d(2)*[*unfolded dist-norm*]) **using** *assms(2)* **using** *goal1* **by** *auto*
qed(*insert e*,*auto*)
next case *True* **have** $f \circ$ *dest-vec1 integrable-on* $\{vec1\ a..vec1\ x\}$ **apply**(*rule integrable-subinterval*,*rule integrable-continuous*)
apply(*rule continuous-on-o-dest-vec1 assms*)+ **unfolding** *not-less* **using** *assms(2) goal1* **by** *auto*
hence $*$:*?I a x − ?I a y = ?I y x* **unfolding** *algebra-simps* **apply**(*subst eq-commute*) **apply**(*rule integral-combine*)
using *True* **using** *assms(2) goal1* **by** *auto*
have $**$:*norm* $(y - x)$ = *content* $\{vec1\ y..vec1\ x\}$ **apply**(*subst content-1*) **using** *True* **unfolding** *not-less* **by** *auto*
have $***$:$\bigwedge fy\ fx\ c{::}'a.\ fx - fy - (y - x) *_R c = -(fy - fx - (x - y) *_R c)$ **unfolding** *scaleR-left.diff* **by** *auto*
show *?thesis* **apply**(*subst* $***$) **unfolding** *norm-minus-cancel* $**$
apply(*rule has-integral-bound*[**where** f=$(\lambda u.\ f\ u - f\ x)$ *o dest-vec1*]) **unfolding** $*$ **unfolding** *o-def*
defer apply(*rule has-integral-sub*) **apply**(*subst minus-minus*[*THEN sym*]) **unfolding** *minus-minus*
apply(*rule integrable-integral*) **apply**(*rule integrable-subinterval*,*rule integrable-continuous*)
apply(*rule continuous-on-o-dest-vec1*[*unfolded o-def*] *assms*)+
proof− **show** $\{vec1\ y..vec1\ x\} \subseteq \{vec1\ a..vec1\ b\}$ **using** *goal1 assms(2)* **by** *auto*
have $*$:$x - y = norm(y - x)$ **using** *True* **by** *auto*
show $((\lambda xa.\ f\ x)$ *has-integral* $(x - y) *_R f\ x)\ \{vec1\ y..vec1\ x\}$ **apply**(*subst* $*$) **unfolding** $**$ **by** *auto*
show $\forall xa{\in}\{vec1\ y..vec1\ x\}.$ *norm* $(f\ (dest\text{-}vec1\ xa) - f\ x) \leq e$ **apply** *safe* **apply**(*rule less-imp-le*)
apply(*rule d(2)*[*unfolded dist-norm*]) **using** *assms(2)* **using** *goal1* **by** *auto*
qed(*insert e*,*auto*) **qed qed qed**

lemma *integral-has-vector-derivative'*: **fixes** f::*real^1* ⇒ $'a$::*banach*
assumes *continuous-on* $\{a..b\}\ f\ x \in \{a..b\}$
shows $((\lambda u.$ (*integral* $\{a..vec\ u\}\ f))$ *has-vector-derivative* $f\ x)$ (*at* $(x\$1)$ *within* $\{a\$1..b\$1\}$)

using *integral-has-vector-derivative*[*OF continuous-on-o-vec1*[*OF assms*(*1*)], *of x*$*1*]
unfolding *o-def vec1-dest-vec1* **using** *assms*(*2*) **by** *auto*

lemma *antiderivative-continuous*: **assumes** *continuous-on* {*a..b*::*real*} *f*
obtains *g* **where** ∀ *x*∈ {*a..b*}. (*g has-vector-derivative* (*f*(*x*)::-::*banach*)) (*at x within* {*a..b*})
apply(*rule that*,*rule*) **using** *integral-has-vector-derivative*[*OF assms*] **by** *auto*

23.46 Combined fundamental theorem of calculus.

lemma *antiderivative-integral-continuous*: **fixes** *f*::*real* ⇒ '*a*::*banach* **assumes** *continuous-on* {*a..b*} *f*
obtains *g* **where** ∀ *u*∈{*a..b*}. ∀ *v* ∈ {*a..b*}. *u* ≤ *v* ⟶ ((*f o dest-vec1*) *has-integral* (*g v* − *g u*)) {*vec u..vec v*}
proof− **from** *antiderivative-continuous*[*OF assms*] **guess** *g* **. note** *g*=*this*
show *?thesis* **apply**(*rule that*[*of g*])
proof *safe* **case** *goal1* **have** ∀ *x*∈{*u..v*}. (*g has-vector-derivative f x*) (*at x within* {*u..v*})
apply(*rule*,*rule has-vector-derivative-within-subset*) **apply**(*rule g*[*rule-format*]) **using** *goal1*(*1*−*2*) **by** *auto*
thus *?case* **using** *fundamental-theorem-of-calculus*[*OF goal1*(*3*),*of g o dest-vec1 f o dest-vec1*]
unfolding *o-def vec1-dest-vec1* **by** *auto* **qed qed**

23.47 General "twiddling" for interval-to-interval function image.

lemma *has-integral-twiddle*:
assumes *0* < *r* ∀ *x*. *h*(*g x*) = *x* ∀ *x*. *g*(*h x*) = *x* ∀ *x*. *continuous* (*at x*) *g*
∀ *u v*. ∃ *w z*. *g* ‘ {*u..v*} = {*w..z*}
∀ *u v*. ∃ *w z*. *h* ‘ {*u..v*} = {*w..z*}
∀ *u v*. *content*(*g* ‘ {*u..v*}) = *r* ∗ *content* {*u..v*}
(*f has-integral i*) {*a..b*}
shows ((λ*x*. *f*(*g x*)) *has-integral* (*1* / *r*) ∗*R* *i*) (*h* ‘ {*a..b*})
proof− **{ presume** ∗:{*a..b*} ≠ {} ⟹ *?thesis*
show *?thesis* **apply** *cases* **defer apply**(*rule* ∗,*assumption*)
proof− **case** *goal1* **thus** *?thesis* **unfolding** *goal1 assms*(*8*)[*unfolded goal1 has-integral-empty-eq*] **by** *auto* **qed }**
assume {*a..b*} ≠ {} **from** *assms*(*6*)[*rule-format*,*of a b*] **guess** *w z* **apply**−**by**(*erule exE*)+ **note** *wz*=*this*
have *inj*:*inj g inj h* **unfolding** *inj-on-def* **apply** *safe* **apply**(*rule-tac*[!] *ccontr*)
using *assms*(*2*) **apply**(*erule-tac x*=*x* **in** *allE*) **using** *assms*(*2*) **apply**(*erule-tac x*=*y* **in** *allE*) **defer**
using *assms*(*3*) **apply**(*erule-tac x*=*x* **in** *allE*) **using** *assms*(*3*) **apply**(*erule-tac x*=*y* **in** *allE*) **by** *auto*
show *?thesis* **unfolding** *has-integral-def has-integral-compact-interval-def* **apply**(*subst if-P*) **apply**(*rule*,*rule*,*rule wz*)
proof *safe* **fix** *e*::*real* **assume** *e*:*e*>*0* **hence** *e* ∗ *r* > *0* **using** *assms*(*1*) **by**(*rule*

mult-pos-pos)
from *assms*(*8*)[*unfolded has-integral*,*rule-format*,*OF this*] **guess** *d* **apply**−**by**(*erule exE conjE*)+ **note** *d*=*this*[*rule-format*]
def $d' \equiv \lambda x\ y.\ d\ (g\ x)\ (g\ y)$ **have** d':$\bigwedge x.\ d'\ x = \{y.\ g\ y \in (d\ (g\ x))\}$ **unfolding** *d*′*-def* **by**(*auto simp add*:*mem-def*)
show $\exists\, d.$ *gauge* $d \wedge (\forall\, p.\ p$ *tagged-division-of* h ‘ $\{a..b\} \wedge d$ *fine* $p \longrightarrow$ *norm* $((\sum (x,\ k){\in}p.$ *content* $k *_R f\ (g\ x)) - (1\ /\ r) *_R i) < e)$
proof(*rule-tac x*=*d*′ **in** *exI*,*safe*) **show** *gauge* *d*′ **using** *d*(*1*) **unfolding** *gauge-def* *d*′ **using** *continuous-open-preimage-univ*[*OF assms*(*4*)] **by** *auto*
fix *p* **assume** *as*:*p tagged-division-of h* ‘ $\{a..b\}$ *d*′ *fine p* **note** p = *tagged-division-ofD*[*OF as*(*1*)]
have $(\lambda(x,\ k).\ (g\ x,\ g\ `\ k))$ ‘ *p tagged-division-of* $\{a..b\} \wedge d$ *fine* $(\lambda(x,\ k).\ (g\ x,\ g\ `\ k))$ ‘ *p* **unfolding** *tagged-division-of*
proof *safe* **show** *finite* $((\lambda(x,\ k).\ (g\ x,\ g\ `\ k))\ `\ p)$ **using** *as* **by** *auto*
show *d fine* $(\lambda(x,\ k).\ (g\ x,\ g\ `\ k))$ ‘ *p* **using** *as*(*2*) **unfolding** *fine-def* *d*′ **by** *auto*
fix *x k* **assume** *xk*[*intro*]:$(x,k) \in p$ **show** $g\ x \in g\ `\ k$ **using** *p*(*2*)[*OF xk*] **by** *auto*
show $\exists\, u\ v.\ g\ `\ k = \{u..v\}$ **using** *p*(*4*)[*OF xk*] **using** *assms*(*5*−*6*) **by** *auto*
{ fix *y* **assume** $y \in k$ **thus** $g\ y \in \{a..b\}\ g\ y \in \{a..b\}$ **using** *p*(*3*)[*OF xk*,*unfolded subset-eq*,*rule-format*,*of h* $(g\ y)$]
using *assms*(*2*)[*rule-format*,*of y*] **unfolding** *inj-image-mem-iff*[*OF inj*(*2*)] **by** *auto* **}**
fix *x*′ *k*′ **assume** *xk*′:$(x',k') \in p$ **fix** *z* **assume** $z \in$ *interior* $(g\ `\ k)\ z \in$ *interior* $(g\ `\ k')$
hence ∗:*interior* $(g\ `\ k) \cap$ *interior* $(g\ `\ k') \neq \{\}$ **by** *auto*
have *same*:$(x,\ k) = (x',\ k')$ **apply**−**apply**(*rule ccontr*,*drule p*(*5*)[*OF xk xk*′])
proof− **assume** *as*:*interior k* ∩ *interior k*′ = {} **from** *nonempty-witness*[*OF* ∗] **guess** *z* **.**
hence $z \in g\ `\ (interior\ k \cap interior\ k')$ **using** *interior-image-subset*[*OF assms*(*4*) *inj*(*1*)]
unfolding *image-Int*[*OF inj*(*1*)] **by** *auto* **thus** *False* **using** *as* **by** *blast*
qed thus $g\ x = g\ x'$ **by** *auto*
{ fix *z* **assume** $z \in k$ **thus** $g\ z \in g\ `\ k'$ **using** *same* **by** *auto* **}**
{ fix *z* **assume** $z \in k'$ **thus** $g\ z \in g\ `\ k$ **using** *same* **by** *auto* **}**
next fix *x* **assume** $x \in \{a..b\}$ **hence** $h\ x \in \bigcup\{k.\ \exists\, x.\ (x,\ k) \in p\}$ **using** *p*(*6*) **by** *auto*
then guess *X* **unfolding** *Union-iff* **..** **note** *X*=*this* **from** *this*(*1*) **guess** *y* **unfolding** *mem-Collect-eq* **..**
thus $x \in \bigcup\{k.\ \exists\, x.\ (x,\ k) \in (\lambda(x,\ k).\ (g\ x,\ g\ `\ k))\ `\ p\}$ **apply**−
apply(*rule-tac X*=*g* ‘ *X* **in** *UnionI*) **defer apply**(*rule-tac x*=*h x* **in** *image-eqI*)
using *X*(*2*) *assms*(*3*)[*rule-format*,*of x*] **by** *auto*
qed note ∗∗ = *d*(*2*)[*OF this*] **have** ∗:*inj-on* $(\lambda(x,\ k).\ (g\ x,\ g\ `\ k))\ p$ **using** *inj*(*1*) **unfolding** *inj-on-def* **by** *fastsimp*
have $(\sum (x,\ k){\in}(\lambda(x,\ k).\ (g\ x,\ g\ `\ k))\ `\ p.$ *content* $k *_R f\ x) - i = r *_R (\sum (x,\ k){\in}p.$ *content* $k *_R f\ (g\ x)) - i$ (**is** *?l* = -) **unfolding** *algebra-simps add-left-cancel*
unfolding *setsum-reindex*[*OF* ∗] **apply**(*subst scaleR-right.setsum*) **defer**

apply(*rule setsum-cong2*) **unfolding** *o-def split-paired-all split-conv*
apply(*drule p*(*4*)) **apply** *safe* **unfolding** *assms*(*7*)[*rule-format*] **using** *p*
by *auto*
also have ... = *r* $*_R$ (($\sum$ (*x*, *k*)∈*p*. *content k* $*_R$ *f* (*g x*)) − (*1* / *r*) $*_R$ *i*) (**is**
- = *?r*) **unfolding** *scaleR.diff-right scaleR.scaleR-left*[*THEN sym*]
unfolding *real-scaleR-def* **using** *assms*(*1*) **by** *auto* **finally have** $*$:*?l* = *?r*
.
show *norm* (($\sum$ (*x*, *k*)∈*p*. *content k* $*_R$ *f* (*g x*)) − (*1* / *r*) $*_R$ *i*) < *e* **using**
$**$ **unfolding** $*$ **unfolding** *norm-scaleR*
using *assms*(*1*) **by**(*auto simp add*:*field-simps*) **qed qed qed**

23.48 Special case of a basic affine transformation.

lemma *interval-image-affinity-interval*: **shows** ∃ *u v*. (λ*x*. *m* $*_R$ (*x*::*real*^'*n*) + *c*)
‘ {*a*..*b*} = {*u*..*v*}
unfolding *image-affinity-interval* **by** *auto*

lemmas *Cart-simps* = *Cart-nth.add Cart-nth.minus Cart-nth.zero Cart-nth.diff*
Cart-nth.scaleR real-scaleR-def Cart-lambda-beta
Cart-eq vector-le-def vector-less-def

lemma *setprod-cong2*: **assumes** ⋀*x*. *x* ∈ *A* ⟹ *f x* = *g x* **shows** *setprod f A* =
setprod g A
apply(*rule setprod-cong*) **using** *assms* **by** *auto*

lemma *content-image-affinity-interval*:
content((λ*x*::*real*^'*n*. *m* $*_R$ *x* + *c*) ‘ {*a*..*b*}) = (*abs m*) ^ *CARD*('*n*) $*$ *content*
{*a*..*b*} (**is** *?l* = *?r*)
proof− { **presume** $*$:{*a*..*b*}≠{} ⟹ *?thesis* **show** *?thesis* **apply**(*cases*,*rule* $*$,*assumption*)
unfolding *not-not* **using** *content-empty* **by** *auto* }
assume *as*:{*a*..*b*}≠{} **show** *?thesis* **proof**(*cases m* ≥ *0*)
case *True* **show** *?thesis* **unfolding** *image-affinity-interval if-not-P*[*OF as*]
if-P[*OF True*]
unfolding *content-closed-interval'*[*OF as*] **apply**(*subst content-closed-interval'*)

defer apply(*subst setprod-constant*[*THEN sym*]) **apply**(*rule finite-UNIV*)
unfolding *setprod-timesf*[*THEN sym*]
apply(*rule setprod-cong2*) **using** *True as* **unfolding** *interval-ne-empty Cart-simps*
not-le
by(*auto simp add*:*field-simps intro*:*mult-left-mono*)
next case *False* **show** *?thesis* **unfolding** *image-affinity-interval if-not-P*[*OF as*]
if-not-P[*OF False*]
unfolding *content-closed-interval'*[*OF as*] **apply**(*subst content-closed-interval'*)

defer apply(*subst setprod-constant*[*THEN sym*]) **apply**(*rule finite-UNIV*)
unfolding *setprod-timesf*[*THEN sym*]
apply(*rule setprod-cong2*) **using** *False as* **unfolding** *interval-ne-empty*
Cart-simps not-le
by(*auto simp add*:*field-simps mult-le-cancel-left-neg*) **qed qed**

lemma *has-integral-affinity*: **assumes** (*f has-integral i*) {*a..b*::*real^'n*} *m* ≠ *0*
shows ((λ*x*. *f*(*m* *$_R$ *x* + *c*)) *has-integral* ((*1* / (*abs*(*m*) ^ *CARD*('*n*::*finite*))) *$_R$ i)) ((λ*x*. (*1* / *m*) *$_R$ *x* + −((*1* / *m*) *$_R$ *c*)) ‘ {*a..b*})
apply(*rule has-integral-twiddle*,*safe*) **unfolding** *Cart-eq Cart-simps* **apply**(*rule zero-less-power*)
defer apply(*insert assms*(*2*), *simp add*:*field-simps*) **apply**(*insert assms*(*2*), *simp add*:*field-simps*)
apply(*rule continuous-intros*)+ **apply**(*rule interval-image-affinity-interval*)+ **apply**(*rule content-image-affinity-interval*) **using** *assms* **by** *auto*

lemma *integrable-affinity*: **assumes** *f integrable-on* {*a..b*} *m* ≠ *0*
shows (λ*x*. *f*(*m* *$_R$ *x* + *c*)) *integrable-on* ((λ*x*. (*1* / *m*) *$_R$ *x* + −((*1*/*m*) *$_R$ *c*)) ‘ {*a..b*})
using *assms* **unfolding** *integrable-on-def* **apply** *safe* **apply**(*drule has-integral-affinity*) **by** *auto*

23.49 Special case of stretching coordinate axes separately.

lemma *image-stretch-interval*:
(λ*x*. χ *k*. *m k* * *x*$*k*) ‘ {*a..b*::*real^'n*} =
(*if* {*a..b*} = {} *then* {} *else* {(χ *k*. *min* (*m*(*k*) * *a*$*k*) (*m*(*k*) * *b*$*k*)) .. (χ *k*. *max* (*m*(*k*) * *a*$*k*) (*m*(*k*) * *b*$*k*))}) (**is** *?l* = *?r*)
proof(*cases* {*a..b*}={}) **case** *True* **thus** *?thesis* **unfolding** *True* **by** *auto*
next have *:⋀*P Q*. (∀*i*. *P i*) ∧ (∀*i*. *Q i*) ⟷ (∀*i*. *P i* ∧ *Q i*) **by** *auto*
case *False* **note** *ab* = *this*[*unfolded interval-ne-empty*]
show *?thesis* **apply**−**apply**(*rule set-ext*)
proof− **fix** *x*::*real^'n* **have** **:⋀*P Q*. (∀*i*. *P i* = *Q i*) ⟹ (∀*i*. *P i*) = (∀*i*. *Q i*) **by** *auto*
show *x* ∈ *?l* ⟷ *x* ∈ *?r* **unfolding** *if-not-P*[*OF False*]
unfolding *image-iff mem-interval Bex-def Cart-simps Cart-eq* *
unfolding *lambda-skolem*[*THEN sym*,*of* λ *i xa*. (*a* $ *i* ≤ *xa* ∧ *xa* ≤ *b* $ *i*) ∧ *x* $ *i* = *m i* * *xa*]
proof(*rule* **,*rule*) **fix** *i*::'*n* **show** (∃*xa*. (*a* $ *i* ≤ *xa* ∧ *xa* ≤ *b* $ *i*) ∧ *x* $ *i* = *m i* * *xa*) =
(*min* (*m i* * *a* $ *i*) (*m i* * *b* $ *i*) ≤ *x* $ *i* ∧ *x* $ *i* ≤ *max* (*m i* * *a* $ *i*) (*m i* * *b* $ *i*))
proof(*cases m i* = *0*) **case** *True* **thus** *?thesis* **using** *ab* **by** *auto*
next case *False* **hence** *0* < *m i* ∨ *0* > *m i* **by** *auto* **thus** *?thesis* **apply**−
proof(*erule disjE*) **assume** *as*:*0* < *m i* **hence** *:*min* (*m i* * *a* $ *i*) (*m i* * *b* $ *i*) = *m i* * *a* $ *i*
max (*m i* * *a* $ *i*) (*m i* * *b* $ *i*) = *m i* * *b* $ *i* **using** *ab* **unfolding** *min-def max-def* **by** *auto*
show *?thesis* **unfolding** * **apply** *rule* **defer apply**(*rule-tac x*=*1* / *m i* * *x*$*i* **in** *exI*)
using *as* **by**(*auto simp add*:*field-simps*)
next assume *as*:*0* > *m i* **hence** *:*max* (*m i* * *a* $ *i*) (*m i* * *b* $ *i*) = *m i* * *a* $ *i*
min (*m i* * *a* $ *i*) (*m i* * *b* $ *i*) = *m i* * *b* $ *i* **using** *ab as* **unfolding**

min-def max-def
by(*auto simp add*:*field-simps mult-le-cancel-left-neg intro*: *order-antisym*)
show *?thesis* **unfolding** ∗ **apply** *rule* **defer apply**(*rule-tac x=1 / m i ∗ x$i* **in** *exI*)
using *as* **by**(*auto simp add*:*field-simps*) **qed qed qed qed qed**

lemma *interval-image-stretch-interval*: ∃ *u v*. (λ*x*. χ *k*. *m k ∗ x$k*) ‘ {*a*..*b*::*real^'n*} = {*u*..*v*}
unfolding *image-stretch-interval* **by** *auto*

lemma *content-image-stretch-interval*:
content((λ*x*::*real^'n*. χ *k*. *m k ∗ x$k*) ‘ {*a*..*b*}) = *abs*(*setprod m UNIV*) ∗ *content*({*a*..*b*})
proof(*cases* {*a*..*b*} = {}) **case** *True* **thus** *?thesis*
unfolding *content-def image-is-empty image-stretch-interval if-P*[*OF True*] **by** *auto*
next case *False* **hence** (λ*x*. χ *k*. *m k ∗ x $ k*) ‘ {*a*..*b*} ≠ {} **by** *auto*
thus *?thesis* **using** *False* **unfolding** *content-def image-stretch-interval* **apply**−**unfolding** *interval-bounds' if-not-P*
unfolding *abs-setprod setprod-timesf*[*THEN sym*] **apply**(*rule setprod-cong2*)
unfolding *Cart-lambda-beta*
proof− **fix** *i*::*'n* **have** (*m i < 0* ∨ *m i > 0*) ∨ *m i = 0* **by** *auto*
thus *max* (*m i ∗ a $ i*) (*m i ∗ b $ i*) − *min* (*m i ∗ a $ i*) (*m i ∗ b $ i*) = |*m i*| ∗ (*b $ i* − *a $ i*)
apply−**apply**(*erule disjE*)+ **unfolding** *min-def max-def* **using** *False*[*unfolded interval-ne-empty*,*rule-format*,*of i*]
by(*auto simp add*:*field-simps not-le mult-le-cancel-left-neg mult-le-cancel-left-pos*)
qed qed

lemma *has-integral-stretch*: **assumes** (*f has-integral i*) {*a*..*b*} ∀ *k*. ~(*m k = 0*)
shows ((λ*x*. *f*(χ *k*. *m k ∗ x$k*)) *has-integral*
((*1*/(*abs*(*setprod m UNIV*))) ∗*R* *i*)) ((λ*x*. χ *k*. *1*/(*m k*) ∗ *x$k*) ‘ {*a*..*b*})
apply(*rule has-integral-twiddle*) **unfolding** *zero-less-abs-iff content-image-stretch-interval*
unfolding *image-stretch-interval empty-as-interval Cart-eq* **using** *assms*
proof− **show** ∀ *x*. *continuous* (*at x*) (λ*x*. χ *k*. *m k ∗ x $ k*)
apply(*rule*,*rule linear-continuous-at*) **unfolding** *linear-linear*
unfolding *linear-def Cart-simps Cart-eq* **by**(*auto simp add*:*field-simps*) **qed** *auto*

lemma *integrable-stretch*:
assumes *f integrable-on* {*a*..*b*} ∀ *k*. ~(*m k = 0*)
shows (λ*x*. *f*(χ *k*. *m k ∗ x$k*)) *integrable-on* ((λ*x*. χ *k*. *1*/(*m k*) ∗ *x$k*) ‘ {*a*..*b*})
using *assms* **unfolding** *integrable-on-def* **apply**−**apply**(*erule exE*) **apply**(*drule has-integral-stretch*) **by** *auto*

23.50 even more special cases.

lemma *uminus-interval-vector*[*simp*]:*uminus* ‘ {*a*..*b*} = {−*b* .. −*a*::*real^'n*}
apply(*rule set-ext*,*rule*) **defer unfolding** *image-iff*

apply(*rule-tac x=−x* **in** *bexI*) **by**(*auto simp add*:*vector-le-def minus-le-iff le-minus-iff*)

lemma *has-integral-reflect-lemma*[*intro*]: **assumes** (*f has-integral i*) {*a*..*b*}
shows ((λ*x*. *f*(−*x*)) *has-integral i*) {−*b* .. −*a*}
using *has-integral-affinity*[*OF assms*, *of* −*1 0*] **by** *auto*

lemma *has-integral-reflect*[*simp*]: ((λ*x*. *f*(−*x*)) *has-integral i*) {−*b*..−*a*} ⟷ (*f has-integral i*) ({*a*..*b*})
apply *rule* **apply**(*drule-tac*[!] *has-integral-reflect-lemma*) **by** *auto*

lemma *integrable-reflect*[*simp*]: (λ*x*. *f*(−*x*)) *integrable-on* {−*b*..−*a*} ⟷ *f integrable-on* {*a*..*b*}
unfolding *integrable-on-def* **by** *auto*

lemma *integral-reflect*[*simp*]: *integral* {−*b*..−*a*} (λ*x*. *f*(−*x*)) = *integral* ({*a*..*b*}) *f*
unfolding *integral-def* **by** *auto*

23.51 Stronger form of FCT; quite a tedious proof.

declare *norm-triangle-ineq4*[*intro*]

lemma *bgauge-existence-lemma*: (∀ *x*∈*s*. ∃ *d*::*real*. *0* < *d* ∧ *q d x*) ⟷ (∀ *x*. ∃ *d*>*0*. *x*∈*s* ⟶ *q d x*) **by**(*meson zero-less-one*)

lemma *additive-tagged-division-1'*: **fixes** *f*::*real* ⇒ *'a*::*real-normed-vector*
assumes *a* ≤ *b p tagged-division-of* {*vec1 a*..*vec1 b*}
shows *setsum* (λ(*x*,*k*). *f* (*dest-vec1* (*interval-upperbound k*)) − *f*(*dest-vec1* (*interval-lowerbound k*))) *p* = *f b* − *f a*
using *additive-tagged-division-1*[*OF - assms*(*2*), *of f o dest-vec1*]
unfolding *o-def vec1-dest-vec1* **using** *assms*(*1*) **by** *auto*

lemma *split-minus*[*simp*]:(λ(*x*, *k*). *f x k*) *x* − (λ(*x*, *k*). *g x k*) *x* = (λ(*x*, *k*). *f x k* − *g x k*) *x*
unfolding *split-def* **by**(*rule refl*)

lemma *norm-triangle-le-sub*: *norm x* + *norm y* ≤ *e* ⟹ *norm* (*x* − *y*) ≤ *e*
apply(*subst*(*asm*)(*2*) *norm-minus-cancel*[*THEN sym*])
apply(*drule norm-triangle-le*) **by**(*auto simp add*:*algebra-simps*)

lemma *fundamental-theorem-of-calculus-interior*:
assumes*a* ≤ *b continuous-on* {*a*..*b*} *f* ∀ *x*∈{*a*<..<*b*}. (*f has-vector-derivative f'*(*x*)) (*at x*)
shows ((*f'* *o dest-vec1*) *has-integral* (*f b* − *f a*)) {*vec a*..*vec b*}
proof− { **presume** ∗:*a* < *b* ⟹ *?thesis*
show *?thesis* **proof**(*cases*,*rule* ∗,*assumption*)
assume ¬ *a* < *b* **hence** *a* = *b* **using** *assms*(*1*) **by** *auto*
hence ∗:{*vec a* .. *vec b*} = {*vec b*} *f b* − *f a* = *0* **by**(*auto simp add*: *Cart-eq vector-le-def order-antisym*)
show *?thesis* **unfolding** ∗(*2*) **apply**(*rule has-integral-null*) **unfolding** *content-eq-0-1*

using $*$ ‹a=b› **by** *auto*
qed } **assume** *ab*:$a < b$
let *?P* = $\lambda e.\ \exists d.$ *gauge* $d \wedge (\forall p.$ p *tagged-division-of* $\{$*vec1 a*..*vec1 b*$\}$ $\wedge$ d *fine* $p \longrightarrow$
norm $((\sum (x,\ k){\in}p.$ *content* $k *_R (f' \circ$ *dest-vec1*$)\ x) - (f\ b - f\ a)) \leq e *$ *content* $\{$*vec1 a*..*vec1 b*$\}$)
{ **presume** $\bigwedge e.\ e{>}0 \Longrightarrow$ *?P e* **thus** *?thesis* **unfolding** *has-integral-factor-content* **by** *auto* }
fix *e*::*real* **assume** *e*:$e{>}0$
note *assms*(*3*)[*unfolded has-vector-derivative-def has-derivative-at-alt ball-conj-distrib*]
note *conjunctD2*[*OF this*] **note** *bounded*=*this*(*1*) **and** *this*(*2*)
from *this*(*2*) **have** $\forall x{\in}\{a<..<b\}.\ \exists d{>}0.\ \forall y.$ *norm* $(y - x) < d \longrightarrow$ *norm* $(f\ y - f\ x - (y - x) *_R f'\ x) \leq e/2 *$ *norm* $(y - x)$
apply−**apply** *safe* **apply**(*erule-tac x*=*x* **in** *ballE*,*erule-tac x*=*e*/*2* **in** *allE*) **using** *e* **by** *auto* **note** *this*[*unfolded bgauge-existence-lemma*]
from *choice*[*OF this*] **guess** *d* **..** **note** *conjunctD2*[*OF this*[*rule-format*]] **note** *d* = *this*[*rule-format*]
have *bounded* $(f\ `\ \{a..b\})$ **apply**(*rule compact-imp-bounded compact-continuous-image*)+ **using** *compact-real-interval assms* **by** *auto*
from *this*[*unfolded bounded-pos*] **guess** *B* **..** **note** *B* = *this*[*rule-format*]

have $\exists da.\ 0 < da \wedge (\forall c.\ a \leq c \wedge \{a..c\} \subseteq \{a..b\} \wedge \{a..c\} \subseteq$ *ball a da* $\longrightarrow$ *norm*(*content* $\{$*vec1 a*..*vec1 c*$\} *_R f'\ a - (f\ c - f\ a)) \leq (e * (b - a))$ / *4*)
proof− **have** $a{\in}\{a..b\}$ **using** *ab* **by** *auto*
note *assms*(*2*)[*unfolded continuous-on-eq-continuous-within*,*rule-format*,*OF this*]
note $* =$ *this*[*unfolded continuous-within Lim-within*,*rule-format*] **have** $(e * (b - a))$ / *8* > 0 **using** *e ab* **by**(*auto simp add*:*field-simps*)
from $*$[*OF this*] **guess** *k* **..** **note** *k* = *conjunctD2*[*OF this*,*rule-format*]
have $\exists l.\ 0 < l \wedge$ *norm*$(l *_R f'\ a) \leq (e * (b - a))$ / *8*
proof(*cases* $f'\ a = 0$) **case** *True*
thus *?thesis* **apply**(*rule-tac x*=*1* **in** *exI*) **using** *ab e* **by**(*auto intro*!:*mult-nonneg-nonneg*)

next case *False* **thus** *?thesis*
apply(*rule-tac x*=$(e * (b - a))$ / *8* / *norm* $(f'\ a)$ **in** *exI*)
using *ab e* **by**(*auto simp add*:*field-simps*)
qed then guess *l* **..** **note** *l* = *conjunctD2*[*OF this*]
show *?thesis* **apply**(*rule-tac x*=*min k l* **in** *exI*) **apply** *safe* **unfolding** *min-less-iff-conj* **apply**(*rule*,(*rule l k*)+)
proof− **fix** *c* **assume** *as*:$a \leq c$ $\{a..c\} \subseteq \{a..b\}$ $\{a..c\} \subseteq$ *ball a* $(min\ k\ l)$
note *as'* = *this*[*unfolded subset-eq Ball-def mem-ball dist-real-def mem-interval*]
have *norm* $((c - a) *_R f'\ a - (f\ c - f\ a)) \leq$ *norm* $((c - a) *_R f'\ a) +$ *norm* $(f\ c - f\ a)$ **by**(*rule norm-triangle-ineq4*)
also have ... $\leq e * (b - a)$ / *8* $+ e * (b - a)$ / *8*
proof(*rule add-mono*) **case** *goal1* **have** $|c - a| \leq |l|$ **using** *as'* **by** *auto*
thus *?case* **apply**−**apply**(*rule order-trans*[*OF* - *l*(*2*)]) **unfolding** *norm-scaleR* **apply**(*rule mult-right-mono*) **by** *auto*
next case *goal2* **show** *?case* **apply**(*rule less-imp-le*) **apply**(*cases* $a = c$) **defer**
apply(*rule k*(*2*)[*unfolded dist-norm*]) **using** *as' e ab* **by**(*auto simp*

add:*field-simps*)
qed finally show *norm* (*content* {*vec1 a*..*vec1 c*} $*_R$ *f'* *a* − (*f c* − *f a*)) ≤ *e* ∗ (*b* − *a*) / *4* **unfolding** *content-1'*[*OF as*(*1*)] **by** *auto*
qed qed then guess *da* **.. note** *da*=*conjunctD2*[*OF this*,*rule-format*]

have ∃ *db*>*0*. ∀ *c*≤*b*. {*c*..*b*} ⊆ {*a*..*b*} ∧ {*c*..*b*} ⊆ *ball b db* ⟶ *norm*(*content* {*vec1 c*..*vec1 b*} $*_R$ *f'* *b* − (*f b* − *f c*)) ≤ (*e* ∗ (*b* − *a*)) / *4*
proof− **have** *b*∈{*a*..*b*} **using** *ab* **by** *auto*
note *assms*(*2*)[*unfolded continuous-on-eq-continuous-within*,*rule-format*,*OF this*]
note ∗ = *this*[*unfolded continuous-within Lim-within*,*rule-format*] **have** (*e* ∗ (*b* − *a*)) / *8* > *0* **using** *e ab* **by**(*auto simp add*:*field-simps*)
from ∗[*OF this*] **guess** *k* **.. note** *k* = *conjunctD2*[*OF this*,*rule-format*]
have ∃ *l*. *0* < *l* ∧ *norm*(*l* $*_R$ *f'* *b*) ≤ (*e* ∗ (*b* − *a*)) / *8*
proof(*cases f' b* = *0*) **case** *True*
thus *?thesis* **apply**(*rule-tac x*=*1* **in** *exI*) **using** *ab e* **by**(*auto intro*!:*mult-nonneg-nonneg*)

next case *False* **thus** *?thesis*
apply(*rule-tac x*=(*e* ∗ (*b* − *a*)) / *8* / *norm* (*f'* *b*) **in** *exI*)
using *ab e* **by**(*auto simp add*:*field-simps*)
qed then guess *l* **.. note** *l* = *conjunctD2*[*OF this*]
show *?thesis* **apply**(*rule-tac x*=*min k l* **in** *exI*) **apply** *safe* **unfolding** *min-less-iff-conj* **apply**(*rule*,(*rule l k*)+)
proof− **fix** *c* **assume** *as*:*c* ≤ *b* {*c*..*b*} ⊆ {*a*..*b*} {*c*..*b*} ⊆ *ball b* (*min k l*)
note *as'* = *this*[*unfolded subset-eq Ball-def mem-ball dist-real-def mem-interval*]
have *norm* ((*b* − *c*) $*_R$ *f'* *b* − (*f b* − *f c*)) ≤ *norm* ((*b* − *c*) $*_R$ *f'* *b*) + *norm* (*f b* − *f c*) **by**(*rule norm-triangle-ineq4*)
also have ... ≤ *e* ∗ (*b* − *a*) / *8* + *e* ∗ (*b* − *a*) / *8*
proof(*rule add-mono*) **case** *goal1* **have** |*c* − *b*| ≤ |*l*| **using** *as'* **by** *auto*
thus *?case* **apply**−**apply**(*rule order-trans*[*OF* - *l*(*2*)]) **unfolding** *norm-scaleR* **apply**(*rule mult-right-mono*) **by** *auto*
next case *goal2* **show** *?case* **apply**(*rule less-imp-le*) **apply**(*cases b* = *c*) **defer apply**(*subst norm-minus-commute*)
apply(*rule k*(*2*)[*unfolded dist-norm*]) **using** *as'* *e ab* **by**(*auto simp add*:*field-simps*)
qed finally show *norm* (*content* {*vec1 c*..*vec1 b*} $*_R$ *f'* *b* − (*f b* − *f c*)) ≤ *e* ∗ (*b* − *a*) / *4* **unfolding** *content-1'*[*OF as*(*1*)] **by** *auto*
qed qed then guess *db* **.. note** *db*=*conjunctD2*[*OF this*,*rule-format*]

let *?d* = (λ*x*. *ball x* (*if x*=*vec1 a then da else if x*=*vec b then db else d* (*dest-vec1 x*)))
show *?P e* **apply**(*rule-tac x*=*?d* **in** *exI*)
proof *safe* **case** *goal1* **show** *?case* **apply**(*rule gauge-ball-dependent*) **using** *ab db*(*1*) *da*(*1*) *d*(*1*) **by** *auto*
next case *goal2* **note** *as*=*this* **let** *?A* = {*t*. *fst t* ∈ {*vec1 a*, *vec1 b*}} **note** *p* = *tagged-division-ofD*[*OF goal2*(*1*)]
have *pA*:*p* = (*p* ∩ *?A*) ∪ (*p* − *?A*) *finite* (*p* ∩ *?A*) *finite* (*p* − *?A*) (*p* ∩ *?A*) ∩ (*p* − *?A*) = {} **using** *goal2* **by** *auto*
note ∗ = *additive-tagged-division-1'*[*OF assms*(*1*) *goal2*(*1*), *THEN sym*]
have ∗∗:⋀*n1 s1 n2 s2*::*real*. *n2* ≤ *s2* / *2* ⟹ *n1* − *s1* ≤ *s2* / *2* ⟹ *n1* + *n2*

$\leq$ *s1 + s2* **by** *arith*
show *?case* **unfolding** *content-1'*[*OF assms*(*1*)] **and** $*$[*of* $\lambda x.\ x$] $*$[*of f*] *setsum-subtractf*[*THEN sym*] *split-minus*
unfolding *setsum-right-distrib* **apply**(*subst*(*2*) *pA*,*subst pA*) **unfolding** *setsum-Un-disjoint*[*OF pA*(*2*−)]
proof(*rule norm-triangle-le*,*rule* $**$)
case *goal1* **show** *?case* **apply**(*rule order-trans*,*rule setsum-norm-le*) **apply**(*rule pA*) **defer apply**(*subst divide.setsum*)
proof(*rule order-refl*,*safe*,*unfold not-le o-def split-conv fst-conv*,*rule ccontr*) **fix** *x k* **assume** *as*:(*x*,*k*) $\in$ *p*
e $*$ (*dest-vec1* (*interval-upperbound k*) − *dest-vec1* (*interval-lowerbound k*)) / *2*
< *norm* (*content k* $*_R$ *f'* (*dest-vec1 x*) − (*f* (*dest-vec1* (*interval-upperbound k*)) − *f* (*dest-vec1* (*interval-lowerbound k*))))
from *p*(*4*)[*OF this*(*1*)] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *k*=*this*
hence $\forall i.\ u\$i \leq v\$i$ **and** *uv*:{*u*,*v*}⊆{*u*..*v*} **using** *p*(*2*)[*OF as*(*1*)] **by** *auto*
note *this*(*1*) *this*(*1*)[*unfolded forall-1*]
note *result* = *as*(*2*)[*unfolded k interval-bounds*[*OF this*(*1*)] *content-1*[*OF this*(*2*)]]

assume *as'*:*x* ≠ *vec1 a x* ≠ *vec1 b* **hence** *x*$1 ∈ {*a*<..<*b*} **using** *p*(*2*−*3*)[*OF as*(*1*)] **by**(*auto simp add*: *Cart-eq*) **note** $* = d(2)$[*OF this*]
have *norm* ((*v*$1 − *u*$1) $*_R$ *f'* (*x*$1) − (*f* (*v*$1) − *f* (*u*$1))) =
norm ((*f* (*u*$1) − *f* (*x*$1) − (*u*$1 − *x*$1) $*_R$ *f'* (*x*$1)) − (*f* (*v*$1) − *f* (*x*$1) − (*v*$1 − *x*$1) $*_R$ *f'* (*x*$1)))
apply(*rule arg-cong*[*of* - - *norm*]) **unfolding** *scaleR-left.diff* **by** *auto*
also have ... ≤ *e* / *2* $*$ *norm* (*u*$1 − *x*$1) + *e* / *2* $*$ *norm* (*v*$1 − *x*$1) **apply**(*rule norm-triangle-le-sub*)
apply(*rule add-mono*) **apply**(*rule-tac*[!] $*$) **using** *fineD*[*OF goal2*(*2*) *as*(*1*)] *as'* **unfolding** *k subset-eq*
apply− **apply**(*erule-tac x*=*u* **in** *ballE*,*erule-tac*[*3*] *x*=*v* **in** *ballE*) **using** *uv* **by**(*auto simp add*:*dist-real*)
also have ... ≤ *e* / *2* $*$ *norm* (*v*$1 − *u*$1) **using** *p*(*2*)[*OF as*(*1*)] **unfolding** *k* **by**(*auto simp add*:*field-simps*)
finally have *e* $*$ (*dest-vec1 v* − *dest-vec1 u*) / *2* < *e* $*$ (*dest-vec1 v* − *dest-vec1 u*) / *2*
apply− **apply**(*rule less-le-trans*[*OF result*]) **using** *uv* **by** *auto* **thus** *False* **by** *auto* **qed**

next have $*$:⋀*x s1 s2*::*real*. *0* ≤ *s1* ⟹ *x* ≤ (*s1* + *s2*) / *2* ⟹ *x* − *s1* ≤ *s2* / *2* **by** *auto*
case *goal2* **show** *?case* **apply**(*rule* $*$) **apply**(*rule setsum-nonneg*) **apply**(*rule*,*unfold split-paired-all split-conv*)
defer unfolding *setsum-Un-disjoint*[*OF pA*(*2*−),*THEN sym*] *pA*(*1*)[*THEN sym*] **unfolding** *setsum-right-distrib*[*THEN sym*]
apply(*subst additive-tagged-division-1*[*OF* - *as*(*1*)]) **unfolding** *vec1-dest-vec1* **apply**(*rule assms*)
proof− **fix** *x k* **assume** (*x*,*k*) ∈ *p* ∩ {*t*. *fst t* ∈ {*vec1 a*, *vec1 b*}} **note** *xk*=*IntD1*[*OF this*]

from *p(4)[OF this]* **guess** *u v* **apply−by**(*erule exE*)+ **note** *uv=this*
with *p(2)[OF xk]* **have** $\{u..v\} \neq \{\}$ **by** *auto*
thus $0 \leq e * ((\textit{interval-upperbound}\ k)\$1 - (\textit{interval-lowerbound}\ k)\$1)$
unfolding *uv* **using** *e* **by**(*auto simp add:field-simps*)
next have $*: \bigwedge s\ f\ t\ e.\ \textit{setsum}\ f\ s = \textit{setsum}\ f\ t \Longrightarrow \textit{norm}(\textit{setsum}\ f\ t) \leq e \Longrightarrow \textit{norm}(\textit{setsum}\ f\ s) \leq e$ **by** *auto*
show $\textit{norm}\ (\sum (x,\ k) \in p \cap\ ?A.\ \textit{content}\ k *_R (f' \circ \textit{dest-vec1})\ x - (f\ ((\textit{interval-upperbound}\ k)\$1) - f\ ((\textit{interval-lowerbound}\ k)\$1))) \leq e * (b - a) / 2$
apply(*rule* $*$[**where** $t=p \cap \{t.\ \textit{fst}\ t \in \{\textit{vec1}\ a,\ \textit{vec1}\ b\} \wedge \textit{content}(\textit{snd}\ t) \neq 0\}$])
apply(*rule setsum-mono-zero-right*[*OF pA(2)*]) **defer apply**(*rule*) **unfolding** *split-paired-all split-conv o-def*
proof− **fix** *x k* **assume** $(x,k) \in p \cap \{t.\ \textit{fst}\ t \in \{\textit{vec1}\ a,\ \textit{vec1}\ b\}\} - p \cap \{t.\ \textit{fst}\ t \in \{\textit{vec1}\ a,\ \textit{vec1}\ b\} \wedge \textit{content}\ (\textit{snd}\ t) \neq 0\}$
hence $xk:(x,k) \in p\ \textit{content}\ k = 0$ **by** *auto* **from** *p(4)[OF xk(1)]* **guess** *u v* **apply−by**(*erule exE*)+ **note** *uv=this*
have $k \neq \{\}$ **using** *p(2)[OF xk(1)]* **by** *auto* **hence** $*: u = v$ **using** *xk* **unfolding** *uv content-eq-0-1 interval-eq-empty* **by** *auto*
thus $\textit{content}\ k *_R (f'\ (x\$1)) - (f\ ((\textit{interval-upperbound}\ k)\$1) - f\ ((\textit{interval-lowerbound}\ k)\$1)) = 0$ **using** *xk* **unfolding** *uv* **by** *auto*
next have $*: p \cap \{t.\ \textit{fst}\ t \in \{\textit{vec1}\ a,\ \textit{vec1}\ b\} \wedge \textit{content}(\textit{snd}\ t) \neq 0\} = \{t.\ t \in p \wedge \textit{fst}\ t = \textit{vec1}\ a \wedge \textit{content}(\textit{snd}\ t) \neq 0\} \cup \{t.\ t \in p \wedge \textit{fst}\ t = \textit{vec1}\ b \wedge \textit{content}(\textit{snd}\ t) \neq 0\}$ **by** *blast*
have $**: \bigwedge s\ f.\ \bigwedge e::\textit{real}.\ (\forall x\ y.\ x \in s \wedge y \in s \longrightarrow x = y) \Longrightarrow (\forall x.\ x \in s \longrightarrow \textit{norm}(f\ x) \leq e) \Longrightarrow e>0 \Longrightarrow \textit{norm}(\textit{setsum}\ f\ s) \leq e$
proof(*case-tac* $s=\{\}$) **case** *goal2* **then obtain** *x* **where** $x \in s$ **by** *auto* **hence** $*: s = \{x\}$ **using** *goal2(1)* **by** *auto*
thus *?case* **using** ‹$x \in s$› *goal2(2)* **by** *auto*
qed *auto*
case *goal2* **show** *?case* **apply**(*subst* $*$, *subst setsum-Un-disjoint*) **prefer** *4* **apply**(*rule order-trans*[*of - e * (b − a)/4 + e * (b − a)/4*])
apply(*rule norm-triangle-le*,*rule add-mono*) **apply**(*rule-tac*[*1−2*] $**$)
proof− **let** *?B* $= \lambda x.\ \{t \in p.\ \textit{fst}\ t = \textit{vec1}\ x \wedge \textit{content}\ (\textit{snd}\ t) \neq 0\}$
have *pa*:$\bigwedge k.\ (\textit{vec1}\ a,\ k) \in p \Longrightarrow \exists v.\ k = \{\textit{vec1}\ a\ ..\ v\} \wedge \textit{vec1}\ a \leq v$
proof− **case** *goal1* **guess** *u v* **using** *p(4)[OF goal1]* **apply−by**(*erule exE*)+ **note** *uv=this*
have $*: u \leq v$ **using** *p(2)[OF goal1]* **unfolding** *uv* **by** *auto*
have *u*:$u = \textit{vec1}\ a$ **proof**(*rule ccontr*) **have** $u \in \{u..v\}$ **using** *p(2−3)[OF goal1(1)]* **unfolding** *uv* **by** *auto*
have $u \geq \textit{vec1}\ a$ **using** *p(2−3)[OF goal1(1)]* **unfolding** *uv subset-eq* **by** *auto* **moreover assume** $u \neq \textit{vec1}\ a$ **ultimately**
have $u > \textit{vec1}\ a$ **unfolding** *Cart-simps* **by** *auto*
thus *False* **using** *p(2)[OF goal1(1)]* **unfolding** *uv* **by**(*auto simp add*:*Cart-simps*)
qed thus *?case* **apply**(*rule-tac* $x=v$ **in** *exI*) **unfolding** *uv* **using** $*$ **by** *auto*
qed
have *pb*:$\bigwedge k.\ (\textit{vec1}\ b,\ k) \in p \Longrightarrow \exists v.\ k = \{v\ ..\ \textit{vec1}\ b\} \wedge \textit{vec1}\ b \geq v$

proof− **case** *goal1* **guess** *u v* **using** *p(4)[OF goal1]* **apply**−**by**(*erule exE*)+ **note** *uv=this*
have ∗:*u* ≤ *v* **using** *p(2)[OF goal1]* **unfolding** *uv* **by** *auto*
have *u*:*v* = *vec1 b* **proof**(*rule ccontr*) **have** *u* ∈ {*u*..*v*} **using** *p(2−3)[OF goal1(1)]* **unfolding** *uv* **by** *auto*
have *v* ≤ *vec1 b* **using** *p(2−3)[OF goal1(1)]* **unfolding** *uv subset-eq* **by** *auto* **moreover assume** *v≠vec1 b* **ultimately**
have *v* < *vec1 b* **unfolding** *Cart-simps* **by** *auto*
thus *False* **using** *p(2)[OF goal1(1)]* **unfolding** *uv* **by**(*auto simp add*:*Cart-simps*)
qed thus *?case* **apply**(*rule-tac x=u* **in** *exI*) **unfolding** *uv* **using** ∗ **by** *auto*
qed

show ∀ *x y*. *x* ∈ *?B a* ∧ *y* ∈ *?B a* ⟶ *x* = *y* **apply**(*rule*,*rule*,*rule*,*unfold split-paired-all*)
unfolding *mem-Collect-eq fst-conv snd-conv* **apply** *safe*
proof− **fix** *x k k′* **assume** *k*:(*vec1 a*, *k*) ∈ *p* (*vec1 a*, *k′*) ∈ *p content k* ≠ *0 content k′* ≠ *0*
guess *v* **using** *pa*[*OF k(1)*] **..** **note** *v* = *conjunctD2*[*OF this*]
guess *v′* **using** *pa*[*OF k(2)*] **..** **note** *v′* = *conjunctD2*[*OF this*] **let** *?v* = *vec1* (*min* (*v*$*1*) (*v′*$*1*))
have {*vec1 a* <..< *?v*} ⊆ *k* ∩ *k′* **unfolding** *v v′* **by**(*auto simp add*:*Cart-simps*) **note** *subset-interior*[*OF this*,*unfolded interior-inter*]
moreover have *vec1* ((*a* + *?v*$*1*)/*2*) ∈ {*vec1 a* <..< *?v*} **using** *k(3−)* **unfolding** *v v′ content-eq-0-1 not-le* **by**(*auto simp add*:*Cart-simps*)
ultimately have *vec1* ((*a* + *?v*$*1*)/*2*) ∈ *interior k* ∩ *interior k′* **unfolding** *interior-open*[*OF open-interval*] **by** *auto*
hence ∗:*k* = *k′* **apply**− **apply**(*rule ccontr*) **using** *p(5)*[*OF k(1−2)*] **by** *auto*
{ assume *x*∈*k* **thus** *x*∈*k′* **unfolding** ∗ **. } { assume** *x*∈*k′* **thus** *x*∈*k* **unfolding** ∗ **. }**
qed
show ∀ *x y*. *x* ∈ *?B b* ∧ *y* ∈ *?B b* ⟶ *x* = *y* **apply**(*rule*,*rule*,*rule*,*unfold split-paired-all*)
unfolding *mem-Collect-eq fst-conv snd-conv* **apply** *safe*
proof− **fix** *x k k′* **assume** *k*:(*vec1 b*, *k*) ∈ *p* (*vec1 b*, *k′*) ∈ *p content k* ≠ *0 content k′* ≠ *0*
guess *v* **using** *pb*[*OF k(1)*] **..** **note** *v* = *conjunctD2*[*OF this*]
guess *v′* **using** *pb*[*OF k(2)*] **..** **note** *v′* = *conjunctD2*[*OF this*] **let** *?v* = *vec1* (*max* (*v*$*1*) (*v′*$*1*))
have {*?v* <..< *vec1 b*} ⊆ *k* ∩ *k′* **unfolding** *v v′* **by**(*auto simp add*:*Cart-simps*) **note** *subset-interior*[*OF this*,*unfolded interior-inter*]
moreover have *vec1* ((*b* + *?v*$*1*)/*2*) ∈ {*?v* <..< *vec1 b*} **using** *k(3−)* **unfolding** *v v′ content-eq-0-1 not-le* **by**(*auto simp add*:*Cart-simps*)
ultimately have *vec1* ((*b* + *?v*$*1*)/*2*) ∈ *interior k* ∩ *interior k′* **unfolding** *interior-open*[*OF open-interval*] **by** *auto*
hence ∗:*k* = *k′* **apply**− **apply**(*rule ccontr*) **using** *p(5)*[*OF k(1−2)*] **by** *auto*

{ assume $x{\in}k$ **thus** $x{\in}k'$ **unfolding** $*$ **. } { assume** $x{\in}k'$ **thus** $x{\in}k$ **unfolding** $*$ **. }**
qed

let *?a* = *a* **and** *?b* = *b*
show $\forall x.\ x \in$ *?B a* $\longrightarrow$ *norm* $((\lambda(x,\ k).\ content\ k *_R f'\ (x\$1) - (f\ ((interval\text{-}upperbound\ k)\$1) - f\ ((interval\text{-}lowerbound\ k)\$1)))\ x)$
$\leq e * (b - a)\ /\ 4$ **apply** *safe* **unfolding** *fst-conv snd-conv* **apply** *safe* **unfolding** *vec1-dest-vec1*
proof− **case** *goal1* **guess** *v* **using** *pa*[*OF goal1*(*1*)] **..** **note** *v* = *conjunctD2*[*OF this*]
have *vec1 ?a*∈{*vec1 ?a..v*} **using** *v*(*2*) **by** *auto* **hence** *dest-vec1 v* ≤ *?b* **using** *p*(*3*)[*OF goal1*(*1*)] **unfolding** *subset-eq v* **by** *auto*
moreover have {*?a..dest-vec1 v*} ⊆ *ball ?a da* **using** *fineD*[*OF as*(*2*) *goal1*(*1*)]
apply−**apply**(*subst*(*asm*) *if-P*,*rule refl*) **unfolding** *subset-eq* **apply** *safe* **apply**(*erule-tac x*=*vec1 x* **in** *ballE*)
by(*auto simp add*:*Cart-simps subset-eq dist-real v dist-real-def*)
ultimately
show *?case* **unfolding** *v* **unfolding** *interval-bounds*[*OF v*(*2*)[*unfolded v vector-le-def*]] *vec1-dest-vec1* **apply**−
apply(*rule da*(*2*)[*of v*$*1*,*unfolded vec1-dest-vec1*])
using *goal1 fineD*[*OF as*(*2*) *goal1*(*1*)] **unfolding** *v content-eq-0-1* **by** *auto*
qed
show $\forall x.\ x \in$ *?B b* $\longrightarrow$ *norm* $((\lambda(x,\ k).\ content\ k *_R f'\ (x\$1) - (f\ ((interval\text{-}upperbound\ k)\$1) - f\ ((interval\text{-}lowerbound\ k)\$1)))\ x)$
$\leq e * (b - a)\ /\ 4$ **apply** *safe* **unfolding** *fst-conv snd-conv* **apply** *safe* **unfolding** *vec1-dest-vec1*
proof− **case** *goal1* **guess** *v* **using** *pb*[*OF goal1*(*1*)] **..** **note** *v* = *conjunctD2*[*OF this*]
have *vec1 ?b*∈{*v..vec1 ?b*} **using** *v*(*2*) **by** *auto* **hence** *dest-vec1 v* ≥ *?a* **using** *p*(*3*)[*OF goal1*(*1*)] **unfolding** *subset-eq v* **by** *auto*
moreover have {*dest-vec1 v..?b*} ⊆ *ball ?b db* **using** *fineD*[*OF as*(*2*) *goal1*(*1*)]
apply−**apply**(*subst*(*asm*) *if-P*,*rule refl*) **unfolding** *subset-eq* **apply** *safe* **apply**(*erule-tac x*=*vec1 x* **in** *ballE*) **using** *ab*
by(*auto simp add*:*Cart-simps subset-eq dist-real v dist-real-def*)
ultimately
show *?case* **unfolding** *v* **unfolding** *interval-bounds*[*OF v*(*2*)[*unfolded v vector-le-def*]] *vec1-dest-vec1* **apply**−
apply(*rule db*(*2*)[*of v*$*1*,*unfolded vec1-dest-vec1*])
using *goal1 fineD*[*OF as*(*2*) *goal1*(*1*)] **unfolding** *v content-eq-0-1* **by** *auto*
qed
qed(*insert p*(*1*) *ab e*, *auto simp add*:*field-simps*) **qed** *auto* **qed qed qed**

23.52 Stronger form with finite number of exceptional points.

lemma *fundamental-theorem-of-calculus-interior-strong*: **fixes** *f*::*real* ⇒ 'a::*banach*
assumes*finite s a* ≤ *b continuous-on* {*a*..*b*} *f*
∀ *x*∈{*a*<..<*b*} − *s*. (*f has-vector-derivative f*'(*x*)) (*at x*)
shows ((*f*' *o dest-vec1*) *has-integral* (*f b* − *f a*)) {*vec a*..*vec b*} **using** *assms*
apply−
proof(*induct card s arbitrary*:*s a b*)
case *0* **show** *?case* **apply**(*rule fundamental-theorem-of-calculus-interior*) **using** *0* **by** *auto*
next case (*Suc n*) **from** *this*(*2*) **guess** *c s*' **apply**−**apply**(*subst*(*asm*) *eq-commute*)
unfolding *card-Suc-eq*
apply(*subst*(*asm*)(*2*) *eq-commute*) **by**(*erule exE conjE*)+ **note** *cs* = *this*[*rule-format*]
show *?case* **proof**(*cases c*∈{*a*<..<*b*})
case *False* **thus** *?thesis* **apply**− **apply**(*rule Suc*(*1*)[*OF cs*(*3*) - *Suc*(*4*,*5*)])
apply *safe* **defer**
apply(*rule Suc*(*6*)[*rule-format*]) **using** *Suc*(*3*) **unfolding** *cs* **by** *auto*
next have ∗:*f b* − *f a* = (*f c* − *f a*) + (*f b* − *f c*) **by** *auto*
case *True* **hence** *vec1 a* ≤ *vec1 c vec1 c* ≤ *vec1 b* **by** *auto*
thus *?thesis* **apply**(*subst* ∗) **apply**(*rule has-integral-combine*) **apply** *assumption*+
apply(*rule-tac*[!] *Suc*(*1*)[*OF cs*(*3*)]) **using** *Suc*(*3*) **unfolding** *cs*
proof− **show** *continuous-on* {*a*..*c*} *f continuous-on* {*c*..*b*} *f*
apply(*rule-tac*[!] *continuous-on-subset*[*OF Suc*(*5*)]) **using** *True* **by** *auto*
let *?P* = λ*i j*. ∀ *x*∈{*i*<..<*j*} − *s*'. (*f has-vector-derivative f*' *x*) (*at x*)
show *?P a c ?P c b* **apply** *safe* **apply**(*rule-tac*[!] *Suc*(*6*)[*rule-format*]) **using** *True* **unfolding** *cs* **by** *auto*
qed *auto* **qed qed**

lemma *fundamental-theorem-of-calculus-strong*: **fixes** *f*::*real* ⇒ 'a::*banach*
assumes *finite s a* ≤ *b continuous-on* {*a*..*b*} *f*
∀ *x*∈{*a*..*b*} − *s*. (*f has-vector-derivative f*'(*x*)) (*at x*)
shows ((*f*' *o dest-vec1*) *has-integral* (*f*(*b*) − *f*(*a*))) {*vec1 a*..*vec1 b*}
apply(*rule fundamental-theorem-of-calculus-interior-strong*[*OF assms*(*1*−*3*), *of f* ⌜⌝])
using *assms*(*4*) **by** *auto*

lemma *indefinite-integral-continuous-left*: **fixes** *f*::*real*^*1* ⇒ 'a::*banach*
assumes *f integrable-on* {*a*..*b*} *a* < *c c* ≤ *b* 0 < *e*
obtains *d* **where** 0 < *d* ∀ *t*. *c*$1 − *d* < *t*$1 ∧ *t* ≤ *c* ⟶ *norm*(*integral* {*a*..*c*} *f* − *integral* {*a*..*t*} *f*) < *e*
proof− **have** ∃ *w*>0. ∀ *t*. *c*$1 − *w* < *t*$1 ∧ *t* < *c* ⟶ *norm*(*f c*) ∗ *norm*(*c* − *t*) < *e* / *3*
proof(*cases f c* = *0*) **case** *False* **hence** 0 < *e* / *3* / *norm* (*f c*)
apply−**apply**(*rule divide-pos-pos*) **using** ‹*e*>0› **by** *auto*
thus *?thesis* **apply**−**apply**(*rule*,*rule*,*assumption*,*safe*)
proof− **fix** *t* **assume** *as*:*t* < *c* **and** *c*$1 − *e* / *3* / *norm* (*f c*) < *t*$(*1*::*1*)
hence *c*$1 − *t*$1 < *e* / *3* / *norm* (*f c*) **by** *auto*
hence *norm* (*c* − *t*) < *e* / *3* / *norm* (*f c*) **using** *as* **unfolding** *norm-vector-1 vector-less-def* **by** *auto*

thus *norm (f c) * norm (c − t) < e / 3* **using** *False* **apply**−
apply(*subst mult-commute*) **apply**(*subst pos-less-divide-eq*[*THEN sym*]) **by** *auto*
qed next case *True* **show** *?thesis* **apply**(*rule-tac x=1* **in** *exI*) **unfolding** *True* **using** ‹*e>0*› **by** *auto*
qed then guess *w* **.. note** *w = conjunctD2*[*OF this*,*rule-format*]

have *:e / 3 > 0* **using** *assms* **by** *auto*
have *f integrable-on {a..c}* **apply**(*rule integrable-subinterval*[*OF assms(1)*]) **using** *assms(2−3)* **by** *auto*
from *integrable-integral*[*OF this*,*unfolded has-integral*,*rule-format*,*OF **] **guess** *d1* **..**
note *d1 = conjunctD2*[*OF this*,*rule-format*] **def** *d ≡ λx. ball x w ∩ d1 x*
have *gauge d* **unfolding** *d-def* **using** *w(1) d1* **by** *auto*
note *this*[*unfolded gauge-def*,*rule-format*,*of c*] **note** *conjunctD2*[*OF this*]
from *this(2)*[*unfolded open-contains-ball*,*rule-format*,*OF this(1)*] **guess** *k* **.. note** *k=conjunctD2*[*OF this*]

let *?d = min k (c$1 − a$1)/2* **show** *?thesis* **apply**(*rule that*[*of ?d*])
proof *safe* **show** *?d > 0* **using** *k(1)* **using** *assms(2)* **unfolding** *vector-less-def* **by** *auto*
fix *t* **assume** *as:c$1 − ?d < t$1 t ≤ c* **let** *?thesis = norm (integral {a..c} f − integral {a..t} f) < e*
{ presume *:t < c ⟹ ?thesis*
show *?thesis* **apply**(*cases t = c*) **defer apply**(*rule **)
unfolding *vector-less-def* **apply**(*subst less-le*) **using** ‹*e>0*› *as(2)* **by** *auto*
} assume *t < c*

have *f integrable-on {a..t}* **apply**(*rule integrable-subinterval*[*OF assms(1)*]) **using** *assms(2−3) as(2)* **by** *auto*
from *integrable-integral*[*OF this*,*unfolded has-integral*,*rule-format*,*OF **] **guess** *d2* **..**
note *d2 = conjunctD2*[*OF this*,*rule-format*]
def *d3 ≡ λx. if x ≤ t then d1 x ∩ d2 x else d1 x*
have *gauge d3* **using** *d2(1) d1(1)* **unfolding** *d3-def gauge-def* **by** *auto*
from *fine-division-exists*[*OF this*, *of a t*] **guess** *p* **. note** *p=this*
note *p'=tagged-division-ofD*[*OF this(1)*]
have *pt:∀(x,k)∈p. x$1 ≤ t$1* **proof** *safe* **case** *goal1* **from** *p'(2,3)*[*OF this*] **show** *?case* **by** *auto* **qed**
with *p(2)* **have** *d2 fine p* **unfolding** *fine-def d3-def* **apply** *safe* **apply**(*erule-tac x=(a,b)* **in** *ballE*)+ **by** *auto*
note *d2-fin = d2(2)*[*OF conjI*[*OF p(1) this*]]

have *:{a..c} ∩ {x. x$1 ≤ t$1} = {a..t} {a..c} ∩ {x. x$1 ≥ t$1} = {t..c}*
using *assms(2−3) as* **by**(*auto simp add:field-simps*)
have *p ∪ {(c, {t..c})} tagged-division-of {a..c} ∧ d1 fine p ∪ {(c, {t..c})}* **apply** *rule*
apply(*rule tagged-division-union-interval*[*of - - - 1 t$1*]) **unfolding** * **apply**(*rule p*)

apply(*rule tagged-division-of-self*) **unfolding** *fine-def*
proof *safe* **fix** *x k y* **assume** (*x*,*k*)∈*p* *y*∈*k* **thus** *y*∈*d1 x*
using *p*(*2*) *pt* **unfolding** *fine-def d3-def* **apply**− **apply**(*erule-tac x*=(*x*,*k*) **in** *ballE*)+ **by** *auto*
next fix *x* **assume** *x*∈{*t*..*c*} **hence** *dist c x* < *k* **unfolding** *dist-real*
using *as*(*1*) **by**(*auto simp add*:*field-simps*)
thus *x* ∈ *d1 c* **using** *k*(*2*) **unfolding** *d-def* **by** *auto*
qed(*insert as*(*2*), *auto*) **note** *d1-fin* = *d1*(*2*)[*OF this*]

have ∗:*integral*{*a*..*c*} *f* − *integral* {*a*..*t*} *f* = −(((*c*$*1* − *t*$*1*) ∗*R f c* + (∑(*x*, *k*)∈*p*. *content k* ∗*R f x*)) −
integral {*a*..*c*} *f*) + ((∑(*x*, *k*)∈*p*. *content k* ∗*R f x*) − *integral* {*a*..*t*} *f*) + (*c*$*1* − *t*$*1*) ∗*R f c*
e = (*e*/*3* + *e*/*3*) + *e*/*3* **by** *auto*
have ∗∗:(∑(*x*, *k*)∈*p* ∪ {(*c*, {*t*..*c*})}. *content k* ∗*R f x*) = (*c*$*1* − *t*$*1*) ∗*R f c* + (∑(*x*, *k*)∈*p*. *content k* ∗*R f x*)
proof− **have** ∗∗:⋀*x F*. *F* ∪ {*x*} = *insert x F* **by** *auto*
have (*c*, {*t*..*c*}) ∉ *p* **proof** *safe* **case** *goal1* **from** *p*′(*2*−*3*)[*OF this*]
have *c* ∈ {*a*..*t*} **by** *auto* **thus** *False* **using** ‹*t*<*c*› **unfolding** *vector-less-def* **by** *auto*
qed thus *?thesis* **unfolding** ∗∗ **apply**− **apply**(*subst setsum-insert*) **apply**(*rule p*′)
unfolding *split-conv* **defer apply**(*subst content-1*) **using** *as*(*2*) **by** *auto*
qed

have ∗∗∗:*c*$*1* − *w* < *t*$*1* ∧ *t* < *c*
proof− **have** *c*$*1* − *k* < *t*$*1* **using** ‹*k*>*0*› *as*(*1*) **by**(*auto simp add*:*field-simps*)
moreover have *k* ≤ *w* **apply**(*rule ccontr*) **using** *k*(*2*)
unfolding *subset-eq* **apply**(*erule-tac x*=*c* + *vec* ((*k* + *w*)/*2*) **in** *ballE*)
unfolding *d-def* **using** ‹*k*>*0*› ‹*w*>*0*› **by**(*auto simp add*:*field-simps not-le not-less dist-real*)
ultimately show *?thesis* **using** ‹*t*<*c*› **by**(*auto simp add*:*field-simps*) **qed**

show *?thesis* **unfolding** ∗(*1*) **apply**(*subst* ∗(*2*)) **apply**(*rule norm-triangle-lt add-strict-mono*)+
unfolding *norm-minus-cancel* **apply**(*rule d1-fin*[*unfolded* ∗∗]) **apply**(*rule d2-fin*)
using *w*(*2*)[*OF* ∗∗∗] **unfolding** *norm-scaleR norm-real* **by**(*auto simp add*:*field-simps*)
qed qed

lemma *indefinite-integral-continuous-right*: **fixes** *f*::*real*^*1* ⇒ *'a*::*banach*
assumes *f integrable-on* {*a*..*b*} *a* ≤ *c c* < *b* *0* < *e*
obtains *d* **where** *0* < *d* ∀*t*. *c* ≤ *t* ∧ *t*$*1* < *c*$*1* + *d* ⟶ *norm*(*integral*{*a*..*c*} *f* − *integral*{*a*..*t*} *f*) < *e*
proof− **have** ∗:(λ*x*. *f* (− *x*)) *integrable-on* {− *b*..− *a*} − *b* < − *c* − *c* ≤ − *a*
using *assms* **unfolding** *Cart-simps* **by** *auto*
from *indefinite-integral-continuous-left*[*OF* ∗ ‹*e*>*0*›] **guess** *d* **. note** *d* = *this*
let *?d* = *min d* (*b*$*1* − *c*$*1*)
show *?thesis* **apply**(*rule that*[*of ?d*])

proof *safe* **show** *0 < ?d* **using** *d(1) assms(3)* **unfolding** *Cart-simps* **by** *auto*
fix *t*::-^*1* **assume** *as*:*c ≤ t t$1 < c$1 + ?d*
have ∗:*integral{a..c} f = integral{a..b} f − integral{c..b} f*
integral{a..t} f = integral{a..b} f − integral{t..b} f **unfolding** *algebra-simps*
apply(*rule-tac*[!] *integral-combine*) **using** *assms as* **unfolding** *Cart-simps* **by** *auto*
have *(− c)$1 − d < (− t)$1 ∧ − t ≤ − c* **using** *as* **by** *auto* **note** *d(2)*[*rule-format*,*OF this*]
thus *norm (integral {a..c} f − integral {a..t} f) < e* **unfolding** ∗
unfolding *integral-reflect* **apply**−**apply**(*subst norm-minus-commute*) **by**(*auto simp add*:*algebra-simps*) **qed qed**

declare *dest-vec1-eq*[*simp del*] *not-less*[*simp*] *not-le*[*simp*]

lemma *indefinite-integral-continuous*: **fixes** *f*::*real^1 ⇒ 'a*::*banach*
assumes *f integrable-on {a..b}* **shows** *continuous-on {a..b} (λx. integral {a..x} f)*
proof(*unfold continuous-on-iff*, *safe*) **fix** *x e* **assume** *as*:*x*∈{*a..b*} *0*<(*e*::*real*)
let *?thesis = ∃ d>0. ∀ x'*∈{*a..b*}. *dist x' x < d ⟶ dist (integral {a..x'} f) (integral {a..x} f) < e*
{ presume ∗:*a*<*b ⟹ ?thesis*
show *?thesis* **apply**(*cases*,*rule* ∗,*assumption*)
proof− **case** *goal1* **hence** *{a..b} = {x}* **using** *as(1)* **unfolding** *Cart-simps*
by(*auto simp only*:*field-simps not-less Cart-eq forall-1 mem-interval*)
thus *?case* **using** ‹*e>0*› **by** *auto*
qed } assume *a*<*b*
have *(x=a ∨ x=b) ∨ (a<x ∧ x<b)* **using** *as(1)* **by** (*auto simp add*: *Cart-simps*)
thus *?thesis* **apply**−**apply**(*erule disjE*)+
proof− **assume** *x=a* **have** *a ≤ a* **by** *auto*
from *indefinite-integral-continuous-right*[*OF assms(1) this* ‹*a<b*› ‹*e>0*›] **guess** *d* **. note** *d=this*
show *?thesis* **apply**(*rule*,*rule*,*rule d*,*safe*) **apply**(*subst dist-commute*)
unfolding ‹*x=a*› *dist-norm* **apply**(*rule d(2)*[*rule-format*]) **unfolding** *norm-real* **by** *auto*
next assume *x=b* **have** *b ≤ b* **by** *auto*
from *indefinite-integral-continuous-left*[*OF assms(1)* ‹*a<b*› *this* ‹*e>0*›] **guess** *d* **. note** *d=this*
show *?thesis* **apply**(*rule*,*rule*,*rule d*,*safe*) **apply**(*subst dist-commute*)
unfolding ‹*x=b*› *dist-norm* **apply**(*rule d(2)*[*rule-format*]) **unfolding** *norm-real* **by** *auto*
next assume *a<x ∧ x<b* **hence** *xl*:*a<x x≤b* **and** *xr*:*a≤x x<b* **by**(*auto simp add*: *vector-less-def*)
from *indefinite-integral-continuous-left* [*OF assms(1) xl* ‹*e>0*›] **guess** *d1* **. note** *d1=this*
from *indefinite-integral-continuous-right*[*OF assms(1) xr* ‹*e>0*›] **guess** *d2* **. note** *d2=this*
show *?thesis* **apply**(*rule-tac x=min d1 d2* **in** *exI*)
proof *safe* **show** *0 < min d1 d2* **using** *d1 d2* **by** *auto*
fix *y* **assume** *y*∈{*a..b*} *dist y x < min d1 d2*

thus *dist* (*integral* {*a*..*y*} *f*) (*integral* {*a*..*x*} *f*) < *e* **apply**−**apply**(*subst dist-commute*)
apply(*cases y* < *x*) **unfolding** *dist-norm* **apply**(*rule d1*(*2*)[*rule-format*]) **defer**
apply(*rule d2*(*2*)[*rule-format*]) **unfolding** *Cart-simps not-less norm-real* **by**(*auto simp add*:*field-simps*)
qed qed qed

23.53 This doesn't directly involve integration, but that gives an easy proof.

lemma *has-derivative-zero-unique-strong-interval*: **fixes** *f*::*real* ⇒ *'a*::*banach*
assumes *finite k continuous-on* {*a*..*b*} *f f a* = *y*
∀ *x*∈({*a*..*b*} − *k*). (*f has-derivative* (λ*h*. *0*)) (*at x within* {*a*..*b*}) *x* ∈ {*a*..*b*}
shows *f x* = *y*
proof− **have** *ab*:*a*≤*b* **using** *assms* **by** *auto*
have ∗:(λ*x*. *0*::*'a*) ∘ *dest-vec1* = (λ*x*. *0*) **unfolding** *o-def* **by** *auto* **have** ∗∗:*a* ≤ *x* **using** *assms* **by** *auto*
have ((λ*x*. *0*::*'a*) ∘ *dest-vec1 has-integral f x* − *f a*) {*vec1 a*..*vec1 x*}
apply(*rule fundamental-theorem-of-calculus-interior-strong*[*OF assms*(*1*) ∗∗])
apply(*rule continuous-on-subset*[*OF assms*(*2*)]) **defer**
apply *safe* **unfolding** *has-vector-derivative-def* **apply**(*subst has-derivative-within-open*[*THEN sym*])
apply *assumption* **apply**(*rule open-interval-real*) **apply**(*rule has-derivative-within-subset*[**where** *s*={*a*..*b*}])
using *assms*(*4*) *assms*(*5*) **by** *auto* **note** *this*[*unfolded* ∗]
note *has-integral-unique*[*OF has-integral-0 this*]
thus *?thesis* **unfolding** *assms* **by** *auto* **qed**

23.54 Generalize a bit to any convex set.

lemmas *scaleR-simps* = *scaleR-zero-left scaleR-minus-left scaleR-left-diff-distrib scaleR-zero-right scaleR-minus-right scaleR-right-diff-distrib scaleR-eq-0-iff scaleR-cancel-left scaleR-cancel-right scaleR.add-right scaleR.add-left real-vector-class.scaleR-one*

lemma *has-derivative-zero-unique-strong-convex*: **fixes** *f*::*real*^*'n* ⇒ *'a*::*banach*
assumes *convex s finite k continuous-on s f c* ∈ *s f c* = *y*
∀ *x*∈(*s* − *k*). (*f has-derivative* (λ*h*. *0*)) (*at x within s*) *x* ∈ *s*
shows *f x* = *y*
proof− { **presume** ∗:*x* ≠ *c* ⟹ *?thesis* **show** *?thesis* **apply**(*cases*,*rule* ∗,*assumption*)
unfolding *assms*(*5*)[*THEN sym*] **by** *auto* } **assume** *x*≠*c*
note *conv* = *assms*(*1*)[*unfolded convex-alt*,*rule-format*]
have *as1*:*continuous-on* {*0*..*1*} (*f* ∘ (λ*t*. (*1* − *t*) ∗*R c* + *t* ∗*R x*))
apply(*rule continuous-on-intros*)+ **apply**(*rule continuous-on-subset*[*OF assms*(*3*)])
apply *safe* **apply**(*rule conv*) **using** *assms*(*4*,*7*) **by** *auto*
have ∗:⋀*t xa*. (*1* − *t*) ∗*R c* + *t* ∗*R x* = (*1* − *xa*) ∗*R c* + *xa* ∗*R x* ⟹ *t* = *xa*
proof− **case** *goal1* **hence** (*t* − *xa*) ∗*R x* = (*t* − *xa*) ∗*R c*
unfolding *scaleR-simps* **by**(*auto simp add*:*algebra-simps*)
thus *?case* **using** ‹*x*≠*c*› **by** *auto* **qed**

have *as2*:*finite* $\{t.\ ((1 - t) *_R c + t *_R x) \in k\}$ **using** *assms*(*2*)
apply(*rule finite-surj*[**where** $f{=}\lambda z.\ SOME\ t.\ (1{-}t) *_R c + t *_R x = z$])
apply *safe* **unfolding** *image-iff* **apply** *rule* **defer apply** *assumption*
apply(*rule sym*) **apply**(*rule some-equality*) **defer apply**(*drule* $*$) **by** *auto*
have $(f \circ (\lambda t.\ (1 - t) *_R c + t *_R x))\ 1 = y$
apply(*rule has-derivative-zero-unique-strong-interval*[*OF as2 as1*, *of*])
unfolding *o-def* **using** *assms*(*5*) **defer apply**−**apply**(*rule*)
proof− **fix** *t* **assume** *as*:$t{\in}\{0..1\} - \{t.\ (1 - t) *_R c + t *_R x \in k\}$
have $*{:}c - t *_R c + t *_R x \in s - k$ **apply** *safe* **apply**(*rule conv*[*unfolded scaleR-simps*])
using ‹$x{\in}s$› ‹$c{\in}s$› *as* **by**(*auto simp add*: *algebra-simps*)
have $(f \circ (\lambda t.\ (1 - t) *_R c + t *_R x)$ *has-derivative* $(\lambda x.\ 0) \circ (\lambda z.\ (0 - z *_R c) + z *_R x))$ (*at t within* $\{0..1\}$)
apply(*rule diff-chain-within*) **apply**(*rule has-derivative-add*)
unfolding *scaleR-simps* **apply**(*rule has-derivative-sub*) **apply**(*rule has-derivative-const*)
apply(*rule has-derivative-vmul-within*,*rule has-derivative-id*)+
apply(*rule has-derivative-within-subset*,*rule assms*(*6*)[*rule-format*])
apply(*rule* $*$) **apply** *safe* **apply**(*rule conv*[*unfolded scaleR-simps*]) **using** ‹$x{\in}s$› ‹$c{\in}s$› **by** *auto*
thus $((\lambda xa.\ f\ ((1 - xa) *_R c + xa *_R x))$ *has-derivative* $(\lambda h.\ 0))$ (*at t within* $\{0..1\}$) **unfolding** *o-def* **.**
qed *auto* **thus** *?thesis* **by** *auto* **qed**

23.55 Also to any open connected set with finite set of exceptions. Could generalize to locally convex set with limpt-free set of exceptions.

lemma *has-derivative-zero-unique-strong-connected*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *connected s open s finite k continuous-on s f c* ∈ *s f c* = *y*
∀ *x*∈(*s* − *k*). (*f has-derivative* (λ*h*. *0*)) (*at x within s*) *x*∈*s*
shows *f x* = *y*
proof− **have** $\{x \in s.\ f\ x \in \{y\}\} = \{\} \vee \{x \in s.\ f\ x \in \{y\}\} = s$
apply(*rule assms*(*1*)[*unfolded connected-clopen*,*rule-format*]) **apply** *rule* **defer**
apply(*rule continuous-closed-in-preimage*[*OF assms*(*4*) *closed-sing*])
apply(*rule open-openin-trans*[*OF assms*(*2*)]) **unfolding** *open-contains-ball*
proof *safe* **fix** *x* **assume** *x*∈*s*
from *assms*(*2*)[*unfolded open-contains-ball*,*rule-format*,*OF this*] **guess** *e* **.. note** *e*=*conjunctD2*[*OF this*]
show ∃ *e*>*0*. *ball x e* ⊆ $\{xa \in s.\ f\ xa \in \{f\ x\}\}$ **apply**(*rule*,*rule*,*rule e*)
proof *safe* **fix** *y* **assume** *y*:*y* ∈ *ball x e* **thus** *y*∈*s* **using** *e* **by** *auto*
show *f y* = *f x* **apply**(*rule has-derivative-zero-unique-strong-convex*[*OF convex-ball*])
apply(*rule assms*) **apply**(*rule continuous-on-subset*,*rule assms*) **apply**(*rule e*)+
apply(*subst centre-in-ball*,*rule e*,*rule*) **apply** *safe*
apply(*rule has-derivative-within-subset*) **apply**(*rule assms*(*7*)[*rule-format*])
using *y e* **by** *auto* **qed qed**
thus *?thesis* **using** ‹$x{\in}s$› ‹*f c* = *y*› ‹$c{\in}s$› **by** *auto* **qed**

23.56 Integrating characteristic function of an interval.

lemma *has-integral-restrict-open-subinterval*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes (*f has-integral i*) {*c*..*d*} {*c*..*d*} ⊆ {*a*..*b*}
shows ((λ*x*. *if x* ∈ {*c*<..<*d*} *then f x else 0*) *has-integral i*) {*a*..*b*}
proof− **def** *g* ≡ λ*x*. *if x* ∈{*c*<..<*d*} *then f x else 0*
{ presume ∗:{*c*..*d*}≠{} ⟹ *?thesis*
show *?thesis* **apply**(*cases*,*rule* ∗,*assumption*)
proof− **case** *goal1* **hence** ∗:{*c*<..<*d*} = {} **using** *interval-open-subset-closed*
by *auto*
show *?thesis* **using** *assms*(*1*) **unfolding** ∗ **using** *goal1* **by** *auto*
qed } assume {*c*..*d*}≠{}
from *partial-division-extend-1*[*OF assms*(*2*) *this*] **guess** *p* **. note** *p*=*this*
note *mon* = *monoidal-lifted*[*OF monoidal-monoid*]
note *operat* = *operative-division*[*OF this operative-integral p*(*1*), *THEN sym*]
let *?P* = (*if g integrable-on* {*a*..*b*} *then Some* (*integral* {*a*..*b*} *g*) *else None*) =
Some i
{ presume *?P* **hence** *g integrable-on* {*a*..*b*} ∧ *integral* {*a*..*b*} *g* = *i*
apply− **apply**(*cases*,*subst*(*asm*) *if-P*,*assumption*) **by** *auto*
thus *?thesis* **using** *integrable-integral* **unfolding** *g-def* **by** *auto* **}**

note *iterate-eq-neutral*[*OF mon*,*unfolded neutral-lifted*[*OF monoidal-monoid*]]
note ∗ = *this*[*unfolded neutral-monoid*]
have *iterate*:*iterate* (*lifted op* +) (*p* − {{*c*..*d*}})
(λ*i*. *if g integrable-on i then Some* (*integral i g*) *else None*) = *Some 0*
proof(*rule* ∗,*rule*) **case** *goal1* **hence** *x*∈*p* **by** *auto* **note** *div* = *division-ofD*(*2*−*5*)[*OF*
p(*1*) *this*]
from *div*(*3*) **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
have *interior x* ∩ *interior* {*c*..*d*} = {} **using** *div*(*4*)[*OF p*(*2*)] *goal1* **by** *auto*
hence (*g has-integral 0*) *x* **unfolding** *uv* **apply**−**apply**(*rule has-integral-spike-interior*[**where**
f=λ*x*. *0*])
unfolding *g-def interior-closed-interval* **by** *auto* **thus** *?case* **by** *auto*
qed

have ∗:*p* = *insert* {*c*..*d*} (*p* − {{*c*..*d*}}) **using** *p* **by** *auto*
have ∗∗:*g integrable-on* {*c*..*d*} **apply**(*rule integrable-spike-interior*[**where** *f*=*f*])
unfolding *g-def* **defer apply**(*rule has-integral-integrable*) **using** *assms*(*1*) **by**
auto
moreover have *integral* {*c*..*d*} *g* = *i* **apply**(*rule has-integral-unique*[*OF - assms*(*1*)])
apply(*rule has-integral-spike-interior*[**where** *f*=*g*]) **defer**
apply(*rule integrable-integral*[*OF* ∗∗]) **unfolding** *g-def* **by** *auto*
ultimately show *?P* **unfolding** *operat* **apply**− **apply**(*subst* ∗) **apply**(*subst*
iterate-insert) **apply** *rule*+
unfolding *iterate* **defer apply**(*subst if-not-P*) **defer using** *p* **by** *auto* **qed**

lemma *has-integral-restrict-closed-subinterval*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes (*f has-integral i*) ({*c*..*d*}) {*c*..*d*} ⊆ {*a*..*b*}
shows ((λ*x*. *if x* ∈ {*c*..*d*} *then f x else 0*) *has-integral i*) {*a*..*b*}
proof− **note** *has-integral-restrict-open-subinterval*[*OF assms*]
note ∗ = *has-integral-spike*[*OF negligible-frontier-interval - this*]

show *?thesis* **apply**(*rule* ∗[*of c d*]) **using** *interval-open-subset-closed*[*of c d*] **by** *auto* **qed**

lemma *has-integral-restrict-closed-subintervals-eq*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach* **assumes** {*c*..*d*} ⊆ {*a*..*b*}
shows ((λ*x*. *if x* ∈ {*c*..*d*} *then f x else 0*) *has-integral i*) {*a*..*b*} ⟷ (*f has-integral i*) {*c*..*d*} (**is** *?l* = *?r*)
proof(*cases* {*c*..*d*} = {}) **case** *False* **let** *?g* = λ*x*. *if x* ∈ {*c*..*d*} *then f x else 0*
show *?thesis* **apply** *rule* **defer apply**(*rule has-integral-restrict-closed-subinterval*[*OF - assms*])
proof *assumption* **assume** *?l* **hence** *?g integrable-on* {*c*..*d*}
apply−**apply**(*rule integrable-subinterval*[*OF - assms*]) **by** *auto*
hence ∗:*f integrable-on* {*c*..*d*}**apply**−**apply**(*rule integrable-eq*) **by** *auto*
hence *i* = *integral* {*c*..*d*} *f* **apply**−**apply**(*rule has-integral-unique*)
apply(*rule* ‹*?l*›) **apply**(*rule has-integral-restrict-closed-subinterval*[*OF - assms*]) **by** *auto*
thus *?r* **using** ∗ **by** *auto* **qed qed** *auto*

23.57 Hence we can apply the limit process uniformly to all integrals.

lemma *has-integral'*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach* **shows**
(*f has-integral i*) *s* ⟷ (∀ *e*>*0*. ∃ *B*>*0*. ∀ *a b*. *ball 0 B* ⊆ {*a*..*b*}
⟶ (∃ *z*. ((λ*x*. *if x* ∈ *s then f*(*x*) *else 0*) *has-integral z*) {*a*..*b*} ∧ *norm*(*z* − *i*) < *e*)) (**is** *?l* ⟷ (∀ *e*>*0*. *?r e*))
proof− { **presume** ∗:∃ *a b*. *s* = {*a*..*b*} ⟹ *?thesis*
show *?thesis* **apply**(*cases*,*rule* ∗,*assumption*)
apply(*subst has-integral-alt*) **by** *auto* }
assume ∃ *a b*. *s* = {*a*..*b*} **then guess** *a b* **apply**−**by**(*erule exE*)+ **note** *s*=*this*
from *bounded-interval*[*of a b*, *THEN conjunct1*, *unfolded bounded-pos*] **guess** *B*
..
note *B* = *conjunctD2*[*OF this*,*rule-format*] **show** *?thesis* **apply** *safe*
proof− **fix** *e* **assume** *?l e*>(*0*::*real*)
show *?r e* **apply**(*rule-tac x*=*B*+*1* **in** *exI*) **apply** *safe* **defer apply**(*rule-tac x*=*i* **in** *exI*)
proof fix *c d* **assume** *as*:*ball 0* (*B*+*1*) ⊆ {*c*..*d*::*real^'n*}
thus ((λ*x*. *if x* ∈ *s then f x else 0*) *has-integral i*) {*c*..*d*} **unfolding** *s*
apply−**apply**(*rule has-integral-restrict-closed-subinterval*) **apply**(*rule* ‹*?l*›[*unfolded s*])
apply *safe* **apply**(*drule B*(*2*)[*rule-format*]) **unfolding** *subset-eq* **apply**(*erule-tac x*=*x* **in** *ballE*)
by(*auto simp add*:*dist-norm*)
qed(*insert B* ‹*e*>*0*›, *auto*)
next assume *as*:∀ *e*>*0*. *?r e*
from *this*[*rule-format*,*OF zero-less-one*] **guess** *C* **..** **note** *C*=*conjunctD2*[*OF this*,*rule-format*]
def *c* ≡ (χ *i*. − *max B C*)::*real^'n* **and** *d* ≡ (χ *i*. *max B C*)::*real^'n*
have *c-d*:{*a*..*b*} ⊆ {*c*..*d*} **apply** *safe* **apply**(*drule B*(*2*)) **unfolding** *mem-interval*
proof case *goal1* **thus** *?case* **using** *component-le-norm*[*of x i*] **unfolding** *c-def*

d-def
by(*auto simp add*:*field-simps*) **qed**
have *ball 0 C* ⊆ {*c*..*d*} **apply** *safe* **unfolding** *mem-interval mem-ball dist-norm*

proof case *goal1* **thus** *?case* **using** *component-le-norm*[*of x i*] **unfolding** *c-def d-def* **by** *auto* **qed**
from *C*(*2*)[*OF this*] **have** ∃ *y*. (*f has-integral y*) {*a*..*b*}
unfolding *has-integral-restrict-closed-subintervals-eq*[*OF c-d*,*THEN sym*] **unfolding** *s* **by** *auto*
then guess *y* **.. note** *y*=*this*

have *y* = *i* **proof**(*rule ccontr*) **assume** *y*≠*i* **hence** *0* < *norm* (*y* − *i*) **by** *auto*
from *as*[*rule-format*,*OF this*] **guess** *C* **.. note** *C*=*conjunctD2*[*OF this*,*rule-format*]
def *c* ≡ (χ *i*. − *max B C*)::*real*^'*n* **and** *d* ≡ (χ *i*. *max B C*)::*real*^'*n*
have *c-d*:{*a*..*b*} ⊆ {*c*..*d*} **apply** *safe* **apply**(*drule B*(*2*)) **unfolding** *mem-interval*
proof case *goal1* **thus** *?case* **using** *component-le-norm*[*of x i*] **unfolding** *c-def d-def*
by(*auto simp add*:*field-simps*) **qed**
have *ball 0 C* ⊆ {*c*..*d*} **apply** *safe* **unfolding** *mem-interval mem-ball dist-norm*
proof case *goal1* **thus** *?case* **using** *component-le-norm*[*of x i*] **unfolding** *c-def d-def* **by** *auto* **qed**
note *C*(*2*)[*OF this*] **then guess** *z* **.. note** *z* = *conjunctD2*[*OF this*, *unfolded s*]
note *this*[*unfolded has-integral-restrict-closed-subintervals-eq*[*OF c-d*]]
hence *z* = *y norm* (*z* − *i*) < *norm* (*y* − *i*) **apply**− **apply**(*rule has-integral-unique*[*OF - y*(*1*)]) **.**
thus *False* **by** *auto* **qed**
thus *?l* **using** *y* **unfolding** *s* **by** *auto* **qed qed**

lemma *has-integral-trans*[*simp*]: **fixes** *f*::*real*^'*n* ⇒ *real* **shows**
((λ*x*. *vec1* (*f x*)) *has-integral vec1 i*) *s* ⟷ (*f has-integral i*) *s*
unfolding *has-integral'*[*unfolded has-integral*]
proof case *goal1* **thus** *?case* **apply** *safe*
apply(*erule-tac x*=*e* **in** *allE*,*safe*) **apply**(*rule-tac x*=*B* **in** *exI*,*safe*)
apply(*erule-tac x*=*a* **in** *allE*, *erule-tac x*=*b* **in** *allE*,*safe*)
apply(*rule-tac x*=*dest-vec1 z* **in** *exI*,*safe*) **apply**(*erule-tac x*=*ea* **in** *allE*,*safe*)
apply(*rule-tac x*=*d* **in** *exI*,*safe*) **apply**(*erule-tac x*=*p* **in** *allE*,*safe*)
apply(*subst*(*asm*)(*2*) *norm-vector-1*) **unfolding** *split-def*
unfolding *setsum-component Cart-nth.diff cond-value-iff*[*of dest-vec1*]
Cart-nth.scaleR vec1-dest-vec1 zero-index **apply** *assumption*
apply(*subst*(*asm*)(*2*) *norm-vector-1*) **by** *auto*
next case *goal2* **thus** *?case* **apply** *safe*
apply(*erule-tac x*=*e* **in** *allE*,*safe*) **apply**(*rule-tac x*=*B* **in** *exI*,*safe*)
apply(*erule-tac x*=*a* **in** *allE*, *erule-tac x*=*b* **in** *allE*,*safe*)
apply(*rule-tac x*=*vec1 z* **in** *exI*,*safe*) **apply**(*erule-tac x*=*ea* **in** *allE*,*safe*)
apply(*rule-tac x*=*d* **in** *exI*,*safe*) **apply**(*erule-tac x*=*p* **in** *allE*,*safe*)
apply(*subst norm-vector-1*) **unfolding** *split-def*
unfolding *setsum-component Cart-nth.diff cond-value-iff*[*of dest-vec1*]

Cart-nth.scaleR vec1-dest-vec1 zero-index **apply** *assumption*
apply(*subst norm-vector-1*) **by** *auto* **qed**

lemma *integral-trans*[*simp*]: **assumes** (*f*::*real^'n* ⇒ *real*) *integrable-on s*
shows *integral s* (λ*x*. *vec1* (*f x*)) = *vec1* (*integral s f*)
apply(*rule integral-unique*) **using** *assms* **by** *auto*

lemma *integrable-on-trans*[*simp*]: **fixes** *f*::*real^'n* ⇒ *real* **shows**
(λ*x*. *vec1* (*f x*)) *integrable-on s* ⟷ (*f integrable-on s*)
unfolding *integrable-on-def*
apply(*subst*(*2*) *vec1-dest-vec1*(*1*)[*THEN sym*]) **unfolding** *has-integral-trans*
apply *safe* **defer apply**(*rule-tac x*=*vec1 y* **in** *exI*) **by** *auto*

lemma *has-integral-le*: **fixes** *f*::*real^'n* ⇒ *real*
assumes (*f has-integral i*) *s* (*g has-integral j*) *s* ∀*x*∈*s*. (*f x*) ≤ (*g x*)
shows *i* ≤ *j* **using** *has-integral-component-le*[*of vec1 o f vec1 i s vec1 o g vec1 j 1*]
unfolding *o-def* **using** *assms* **by** *auto*

lemma *integral-le*: **fixes** *f*::*real^'n* ⇒ *real*
assumes *f integrable-on s g integrable-on s* ∀*x*∈*s*. *f x* ≤ *g x*
shows *integral s f* ≤ *integral s g*
using *has-integral-le*[*OF assms*(*1*,*2*)[*unfolded has-integral-integral*] *assms*(*3*)] .

lemma *has-integral-nonneg*: **fixes** *f*::*real^'n* ⇒ *real*
assumes (*f has-integral i*) *s* ∀*x*∈*s*. *0* ≤ *f x* **shows** *0* ≤ *i*
using *has-integral-component-nonneg*[*of vec1 o f vec1 i s 1*]
unfolding *o-def* **using** *assms* **by** *auto*

lemma *integral-nonneg*: **fixes** *f*::*real^'n* ⇒ *real*
assumes *f integrable-on s* ∀*x*∈*s*. *0* ≤ *f x* **shows** *0* ≤ *integral s f*
using *has-integral-nonneg*[*OF assms*(*1*)[*unfolded has-integral-integral*] *assms*(*2*)]
.

23.58 Hence a general restriction property.

lemma *has-integral-restrict*[*simp*]: **assumes** *s* ⊆ *t* **shows**
((λ*x*. *if x* ∈ *s then f x else* (*0*::*'a*::*banach*)) *has-integral i*) *t* ⟷ (*f has-integral i*) *s*
proof− **have** ∗:⋀*x*. (*if x* ∈ *t then if x* ∈ *s then f x else 0 else 0*) = (*if x*∈*s then f x else 0*) **using** *assms* **by** *auto*
show *?thesis* **apply**(*subst*(*2*) *has-integral'*) **apply**(*subst has-integral'*) **unfolding** ∗ **by** *rule* **qed**

lemma *has-integral-restrict-univ*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach* **shows**
((λ*x*. *if x* ∈ *s then f x else 0*) *has-integral i*) *UNIV* ⟷ (*f has-integral i*) *s* **by** *auto*

lemma *has-integral-on-superset*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*

assumes $\forall x.\ ^\sim(x \in s) \longrightarrow f\ x = 0\ s \subseteq t\ (f\ has\text{-}integral\ i)\ s$
shows $(f\ has\text{-}integral\ i)\ t$
proof− **have** $(\lambda x.\ if\ x \in s\ then\ f\ x\ else\ 0) = (\lambda x.\ if\ x \in t\ then\ f\ x\ else\ 0)$
apply(*rule*) **using** *assms*(*1*−*2*) **by** *auto*
thus *?thesis* **apply**− **using** *assms*(*3*) **apply**(*subst has-integral-restrict-univ*[*THEN sym*])
apply− **apply**(*subst*(*asm*) *has-integral-restrict-univ*[*THEN sym*]) **by** *auto* **qed**

lemma *integrable-on-superset*: **fixes** *f*::*real*^'*n* ⇒ '*a*::*banach*
assumes $\forall x.\ ^\sim(x \in s) \longrightarrow f\ x = 0\ s \subseteq t\ f\ integrable\text{-}on\ s$
shows *f integrable-on t*
using *assms* **unfolding** *integrable-on-def* **by**(*auto intro*:*has-integral-on-superset*)

lemma *integral-restrict-univ*[*intro*]: **fixes** *f*::*real*^'*n* ⇒ '*a*::*banach*
shows *f integrable-on s* ⟹ *integral UNIV* $(\lambda x.\ if\ x \in s\ then\ f\ x\ else\ 0)$ = *integral s f*
apply(*rule integral-unique*) **unfolding** *has-integral-restrict-univ* **by** *auto*

lemma *integrable-restrict-univ*: **fixes** *f*::*real*^'*n* ⇒ '*a*::*banach* **shows**
$(\lambda x.\ if\ x \in s\ then\ f\ x\ else\ 0)$ *integrable-on UNIV* ⟷ *f integrable-on s*
unfolding *integrable-on-def* **by** *auto*

lemma *negligible-on-intervals*: *negligible s* ⟷ $(\forall\ a\ b.\ negligible(s \cap \{a..b\}))$ (**is** *?l* = *?r*)
proof assume *?r* **show** *?l* **unfolding** *negligible-def*
proof *safe* **case** *goal1* **show** *?case* **apply**(*rule has-integral-negligible*[*OF* ‹*?r*›[*rule-format*,*of a b*]])
unfolding *indicator-def* **by** *auto* **qed qed** *auto*

lemma *has-integral-spike-set-eq*: **fixes** *f*::*real*^'*n* ⇒ '*a*::*banach*
assumes *negligible*$((s - t) \cup (t - s))$ **shows** $((f\ has\text{-}integral\ y)\ s \longleftrightarrow (f\ has\text{-}integral\ y)\ t)$
unfolding *has-integral-restrict-univ*[*THEN sym*,*of f*] **apply**(*rule has-integral-spike-eq*[*OF assms*]) **by** (*safe*, *auto split*: *split-if-asm*)

lemma *has-integral-spike-set*[*dest*]: **fixes** *f*::*real*^'*n* ⇒ '*a*::*banach*
assumes *negligible*$((s - t) \cup (t - s))$ $(f\ has\text{-}integral\ y)\ s$
shows $(f\ has\text{-}integral\ y)\ t$
using *assms has-integral-spike-set-eq* **by** *auto*

lemma *integrable-spike-set*[*dest*]: **fixes** *f*::*real*^'*n* ⇒ '*a*::*banach*
assumes *negligible*$((s - t) \cup (t - s))$ *f integrable-on s*
shows *f integrable-on t* **using** *assms*(*2*) **unfolding** *integrable-on-def*
unfolding *has-integral-spike-set-eq*[*OF assms*(*1*)] **.**

lemma *integrable-spike-set-eq*: **fixes** *f*::*real*^'*n* ⇒ '*a*::*banach*
assumes *negligible*$((s - t) \cup (t - s))$
shows (*f integrable-on s* ⟷ *f integrable-on t*)
apply(*rule*,*rule-tac*[!] *integrable-spike-set*) **using** *assms* **by** *auto*

23.59 More lemmas that are useful later.

lemma *has-integral-subset-component-le*: **fixes** f::*real^'n* $\Rightarrow$ *real^'m*
assumes $s \subseteq t$ (f *has-integral* i) s (f *has-integral* j) t $\forall x{\in}t.\ 0 \le f(x)\$k$
shows $i\$k \le j\$k$
proof− **note** *has-integral-restrict-univ*[*THEN sym*, *of* f]
note *assms*(2−3)[*unfolded this*] **note** $*$ = *has-integral-component-le*[*OF this*]
show *?thesis* **apply**(*rule* $*$) **using** *assms*(1,4) **by** *auto* **qed**

lemma *has-integral-subset-le*: **fixes** f::*real^'n* $\Rightarrow$ *real*
assumes $s \subseteq t$ (f *has-integral* i) s (f *has-integral* j) t $\forall x{\in}t.\ 0 \le f(x)$
shows $i \le j$ **using** *has-integral-subset-component-le*[*OF assms*(1), *of vec1 o f vec1 i vec1 j 1*]
unfolding *o-def* **using** *assms* **by** *auto*

lemma *integral-subset-component-le*: **fixes** f::*real^'n* $\Rightarrow$ *real^'m*
assumes $s \subseteq t$ f *integrable-on* s f *integrable-on* t $\forall x \in t.\ 0 \le f(x)\$k$
shows (*integral* s f)$\$k \le$ (*integral* t f)$\$k$
apply(*rule has-integral-subset-component-le*) **using** *assms* **by** *auto*

lemma *integral-subset-le*: **fixes** f::*real^'n* $\Rightarrow$ *real*
assumes $s \subseteq t$ f *integrable-on* s f *integrable-on* t $\forall x \in t.\ 0 \le f(x)$
shows (*integral* s f) $\le$ (*integral* t f)
apply(*rule has-integral-subset-le*) **using** *assms* **by** *auto*

lemma *has-integral-alt'*: **fixes** f::*real^'n* $\Rightarrow$ *'a*::*banach*
shows (f *has-integral* i) $s \longleftrightarrow$ ($\forall a\ b.$ ($\lambda x.$ *if* $x \in s$ *then* $f\ x$ *else* 0) *integrable-on* $\{a..b\}$) $\wedge$
($\forall e{>}0.\ \exists B{>}0.\ \forall a\ b.$ *ball* 0 $B \subseteq \{a..b\} \longrightarrow$ *norm*(*integral* $\{a..b\}$ ($\lambda x.$ *if* $x \in s$ *then* $f\ x$ *else* 0) − i) $< e$) (**is** *?l* = *?r*)
proof assume *?r*
show *?l* **apply**− **apply**(*subst has-integral'*)
proof *safe* **case** *goal1* **from** ‹*?r*›[*THEN conjunct2*,*rule-format*,*OF this*] **guess** B **..** **note** B=*conjunctD2*[*OF this*]
show *?case* **apply**(*rule*,*rule*,*rule* B,*safe*)
apply(*rule-tac* x=*integral* $\{a..b\}$ ($\lambda x.$ *if* $x \in s$ *then* $f\ x$ *else* 0) **in** *exI*)
apply(*drule* B(2)[*rule-format*]) **using** *integrable-integral*[*OF* ‹*?r*›[*THEN conjunct1*,*rule-format*]] **by** *auto*
qed next
assume *?l* **note** *as* = *this*[*unfolded has-integral'*[*of* f],*rule-format*]
let *?f* = $\lambda x.$ *if* $x \in s$ *then* $f\ x$ *else* 0
show *?r* **proof** *safe* **fix** a b::*real^'n*
from *as*[*OF zero-less-one*] **guess** B **..** **note** B=*conjunctD2*[*OF this*,*rule-format*]
let *?a* = (χ i. *min* ($a\$i$) ($-B$))::*real^'n* **and** *?b* = (χ i. *max* ($b\$i$) B)::*real^'n*
show *?f integrable-on* $\{a..b\}$ **apply**(*rule integrable-subinterval*[*of* - *?a ?b*])
proof− **have** *ball* 0 $B \subseteq \{$*?a*..*?b*$\}$ **apply** *safe* **unfolding** *mem-ball mem-interval dist-norm*
proof case *goal1* **thus** *?case* **using** *component-le-norm*[*of* x i] **by**(*auto simp add*:*field-simps*) **qed**
from B(2)[*OF this*] **guess** z **..** **note** *conjunct1*[*OF this*]

thus *?f integrable-on {?a..?b}* **unfolding** *integrable-on-def* **by** *auto*
show $\{a..b\} \subseteq \{?a..?b\}$ **apply** *safe* **unfolding** *mem-interval* **apply**(*rule*,*erule-tac* $x{=}i$ **in** *allE*) **by** *auto* **qed**

fix *e*::*real* **assume** $e{>}0$ **from** *as*[*OF this*] **guess** B **.. note** $B{=}$*conjunctD2*[*OF this*,*rule-format*]
show $\exists B{>}0.\ \forall a\ b.$ *ball 0 B* $\subseteq \{a..b\} \longrightarrow$
norm (*integral* $\{a..b\}$ $(\lambda x.$ *if* $x \in s$ *then f x else 0*$) - i) < e$
proof(*rule*,*rule*,*rule B*,*safe*) **case** *goal1* **from** *B*(*2*)[*OF this*] **guess** *z* **.. note** *z*=*conjunctD2*[*OF this*]
from *integral-unique*[*OF this*(*1*)] **show** *?case* **using** *z*(*2*) **by** *auto* **qed qed qed**

23.60 Continuity of the integral (for a 1-dimensional interval).

lemma *integrable-alt*: **fixes** *f*::*real^'n* $\Rightarrow$ *'a*::*banach* **shows**
f integrable-on s $\longleftrightarrow$
$(\forall a\ b.$ $(\lambda x.$ *if* $x \in s$ *then f x else 0*$)$ *integrable-on* $\{a..b\}) \wedge$
$(\forall e{>}0.\ \exists B{>}0.\ \forall a\ b\ c\ d.$ *ball 0 B* $\subseteq \{a..b\} \wedge$ *ball 0 B* $\subseteq \{c..d\}$
$\longrightarrow$ *norm*(*integral* $\{a..b\}$ $(\lambda x.$ *if* $x \in s$ *then f x else 0*$) -$
integral $\{c..d\}$ $(\lambda x.$ *if* $x \in s$ *then f x else 0*$)) < e)$ (**is** *?l* = *?r*)
proof assume *?l* **then guess** *y* **unfolding** *integrable-on-def* **.. note** *this*[*unfolded has-integral-alt'*[*of f*]]
note *y*=*conjunctD2*[*OF this*,*rule-format*] **show** *?r* **apply** *safe* **apply**(*rule y*)
proof− **case** *goal1* **hence** $e/2 > 0$ **by** *auto* **from** *y*(*2*)[*OF this*] **guess** B **.. note** $B{=}$*conjunctD2*[*OF this*,*rule-format*]
show *?case* **apply**(*rule*,*rule*,*rule B*)
proof *safe* **case** *goal1* **show** *?case* **apply**(*rule norm-triangle-half-l*)
using *B*(*2*)[*OF goal1*(*1*)] *B*(*2*)[*OF goal1*(*2*)] **by** *auto* **qed qed**

next assume *?r* **note** *as* = *conjunctD2*[*OF this*,*rule-format*]
have *Cauchy* $(\lambda n.$ *integral* $(\{(\chi\ i.$ − *real n*$)$ *..* $(\chi\ i.$ *real n*$)\})$ $(\lambda x.$ *if* $x \in s$ *then f x else 0*$))$
proof(*unfold Cauchy-def*,*safe*) **case** *goal1*
from *as*(*2*)[*OF this*] **guess** B **.. note** B = *conjunctD2*[*OF this*,*rule-format*]
from *real-arch-simple*[*of B*] **guess** N **.. note** N = *this*
{ fix *n* **assume** *n*:$n \geq N$ **have** *ball 0 B* $\subseteq \{\chi\ i.$ − *real n*$..\chi\ i.$ *real n*$\}$ **apply** *safe*
unfolding *mem-ball mem-interval dist-norm*
proof case *goal1* **thus** *?case* **using** *component-le-norm*[*of x i*]
using *n N* **by**(*auto simp add*:*field-simps*) **qed }**
thus *?case* **apply**−**apply**(*rule-tac x*=*N* **in** *exI*) **apply** *safe* **unfolding** *dist-norm* **apply**(*rule B*(*2*)) **by** *auto*
qed from *this*[*unfolded convergent-eq-cauchy*[*THEN sym*]] **guess** *i* **..**
note *i* = *this*[*unfolded Lim-sequentially*, *rule-format*]

show *?l* **unfolding** *integrable-on-def has-integral-alt'*[*of f*] **apply**(*rule-tac x*=*i* **in** *exI*)

apply *safe* **apply**(*rule as(1)*[*unfolded integrable-on-def*])
proof− **case** *goal1* **hence** ∗:*e*/2 > 0 **by** *auto*
from *i*[*OF this*] **guess** *N* **.. note** *N* =*this*[*rule-format*]
from *as*(2)[*OF* ∗] **guess** *B* **.. note** *B*=*conjunctD2*[*OF this*,*rule-format*] **let** *?B* = *max* (*real N*) *B*
show *?case* **apply**(*rule-tac x*=*?B* **in** *exI*)
proof *safe* **show** 0 < *?B* **using** *B*(*1*) **by** *auto*
fix *a b* **assume** *ab*:*ball 0 ?B* ⊆ {*a*..*b*::*real*^′*n*}
from *real-arch-simple*[*of ?B*] **guess** *n* **.. note** *n*=*this*
show *norm* (*integral* {*a*..*b*} (λ*x*. *if x* ∈ *s then f x else 0*) − *i*) < *e*
apply(*rule norm-triangle-half-l*) **apply**(*rule B*(*2*)) **defer apply**(*subst norm-minus-commute*)
apply(*rule N*[*unfolded dist-norm*, *of n*])
proof *safe* **show** *N* ≤ *n* **using** *n* **by** *auto*
fix *x*::*real*^′*n* **assume** *x*:*x* ∈ *ball 0 B* **hence** *x*∈ *ball 0 ?B* **by** *auto*
thus *x*∈{*a*..*b*} **using** *ab* **by** *blast*
show *x*∈{χ *i*. − *real n*..χ *i*. *real n*} **using** *x* **unfolding** *mem-interval mem-ball dist-norm* **apply**−
proof case *goal1* **thus** *?case* **using** *component-le-norm*[*of x i*]
using *n* **by**(*auto simp add*:*field-simps*) **qed qed qed qed qed**

lemma *integrable-altD*: **fixes** *f*::*real*^′*n* ⇒ ′*a*::*banach*
assumes *f integrable-on s*
shows ⋀*a b*. (λ*x*. *if x* ∈ *s then f x else 0*) *integrable-on* {*a*..*b*}
⋀*e*. *e*>0 ⟹ ∃ *B*>0. ∀ *a b c d*. *ball 0 B* ⊆ {*a*..*b*} ∧ *ball 0 B* ⊆ {*c*..*d*}
⟶ *norm*(*integral* {*a*..*b*} (λ*x*. *if x* ∈ *s then f x else 0*) − *integral* {*c*..*d*} (λ*x*. *if x* ∈ *s then f x else 0*)) < *e*
using *assms*[*unfolded integrable-alt*[*of f*]] **by** *auto*

lemma *integrable-on-subinterval*: **fixes** *f*::*real*^′*n* ⇒ ′*a*::*banach*
assumes *f integrable-on s* {*a*..*b*} ⊆ *s* **shows** *f integrable-on* {*a*..*b*}
apply(*rule integrable-eq*) **defer apply**(*rule integrable-altD*(*1*)[*OF assms*(*1*)])
using *assms*(*2*) **by** *auto*

23.61 A straddling criterion for integrability.

lemma *integrable-straddle-interval*: **fixes** *f*::*real*^′*n* ⇒ *real*
assumes ∀ *e*>0. ∃ *g h i j*. (*g has-integral i*) ({*a*..*b*}) ∧ (*h has-integral j*) ({*a*..*b*}) ∧
norm(*i* − *j*) < *e* ∧ (∀ *x*∈{*a*..*b*}. (*g x*) ≤ (*f x*) ∧ (*f x*) ≤(*h x*))
shows *f integrable-on* {*a*..*b*}
proof(*subst integrable-cauchy*,*safe*)
case *goal1* **hence** *e*:*e*/3 > 0 **by** *auto* **note** *assms*[*rule-format*,*OF this*]
then guess *g h i j* **apply**− **by**(*erule exE conjE*)+ **note** *obt* = *this*
from *obt*(*1*)[*unfolded has-integral*[*of g*], *rule-format*, *OF e*] **guess** *d1* **.. note** *d1*=*conjunctD2*[*OF this*,*rule-format*]
from *obt*(*2*)[*unfolded has-integral*[*of h*], *rule-format*, *OF e*] **guess** *d2* **.. note** *d2*=*conjunctD2*[*OF this*,*rule-format*]
show *?case* **apply**(*rule-tac x*=λ*x*. *d1 x* ∩ *d2 x* **in** *exI*) **apply**(*rule conjI gauge-inter*

d1 d2)+ **unfolding** *fine-inter*
proof *safe* **have** ∗∗:⋀*i j g1 g2 h1 h2 f1 f2*. *g1 − h2 ≤ f1 − f2 ⟹ f1 − f2 ≤ h1 − g2 ⟹*
abs(*i − j*) *< e / 3 ⟹ abs*(*g2 − i*) *< e / 3 ⟹ abs*(*g1 − i*) *< e / 3 ⟹*
abs(*h2 − j*) *< e / 3 ⟹ abs*(*h1 − j*) *< e / 3 ⟹ abs*(*f1 − f2*) *< e* **using** ‹*e*>*0*› **by** *arith*
case *goal1* **note** *tagged-division-ofD*(*2*−*4*) **note** ∗ = *this*[*OF goal1*(*1*)] *this*[*OF goal1*(*4*)]

have (∑(*x*, *k*)∈*p1*. *content k* ∗*R f x*) − (∑(*x*, *k*)∈*p1*. *content k* ∗*R g x*) ≥ *0*
0 ≤ (∑(*x*, *k*)∈*p2*. *content k* ∗*R h x*) − (∑(*x*, *k*)∈*p2*. *content k* ∗*R f x*)
(∑(*x*, *k*)∈*p2*. *content k* ∗*R f x*) − (∑(*x*, *k*)∈*p2*. *content k* ∗*R g x*) ≥ *0*
0 ≤ (∑(*x*, *k*)∈*p1*. *content k* ∗*R h x*) − (∑(*x*, *k*)∈*p1*. *content k* ∗*R f x*)
unfolding *setsum-subtractf*[*THEN sym*] **apply**− **apply**(*rule-tac*[!] *setsum-nonneg*)
apply *safe* **unfolding** *real-scaleR-def mult.diff-right*[*THEN sym*]
apply(*rule-tac*[!] *mult-nonneg-nonneg*)
proof− **fix** *a b* **assume** *ab*:(*a*,*b*) ∈ *p1*
show *0* ≤ *content b* **using** ∗(*3*)[*OF ab*] **apply** *safe* **using** *content-pos-le* **.**
thus *0* ≤ *content b* **.**
show *0* ≤ *f a − g a 0* ≤ *h a − f a* **using** ∗(*1*−*2*)[*OF ab*] **using** *obt*(*4*)[*rule-format*,*of a*] **by** *auto*
next fix *a b* **assume** *ab*:(*a*,*b*) ∈ *p2*
show *0* ≤ *content b* **using** ∗(*6*)[*OF ab*] **apply** *safe* **using** *content-pos-le* **.**
thus *0* ≤ *content b* **.**
show *0* ≤ *f a − g a 0* ≤ *h a − f a* **using** ∗(*4*−*5*)[*OF ab*] **using** *obt*(*4*)[*rule-format*,*of a*] **by** *auto* **qed**

thus *?case* **apply**− **unfolding** *real-norm-def* **apply**(*rule* ∗∗) **defer defer**
unfolding *real-norm-def*[*THEN sym*] **apply**(*rule obt*(*3*))
apply(*rule d1*(*2*)[*OF conjI*[*OF goal1*(*4*,*5*)]])
apply(*rule d1*(*2*)[*OF conjI*[*OF goal1*(*1*,*2*)]])
apply(*rule d2*(*2*)[*OF conjI*[*OF goal1*(*4*,*6*)]])
apply(*rule d2*(*2*)[*OF conjI*[*OF goal1*(*1*,*3*)]]) **by** *auto* **qed qed**

lemma *integrable-straddle*: **fixes** *f*::*real*^*'n* ⇒ *real*
assumes ∀ *e*>*0*. ∃ *g h i j*. (*g has-integral i*) *s* ∧ (*h has-integral j*) *s* ∧
norm(*i − j*) *< e* ∧ (∀ *x*∈*s*. (*g x*) ≤(*f x*) ∧(*f x*) ≤(*h x*))
shows *f integrable-on s*
proof− **have** ⋀*a b*. (λ*x*. *if x* ∈ *s then f x else 0*) *integrable-on* {*a*..*b*}
proof(*rule integrable-straddle-interval*,*safe*) **case** *goal1* **hence** ∗:*e*/*4* *> 0* **by** *auto*
from *assms*[*rule-format*,*OF this*] **guess** *g h i j* **apply**−**by**(*erule exE conjE*)+ **note** *obt*=*this*
note *obt*(*1*)[*unfolded has-integral-alt'*[*of g*]] **note** *conjunctD2*[*OF this*, *rule-format*]
note *g* = *this*(*1*) **and** *this*(*2*)[*OF* ∗] **from** *this*(*2*) **guess** *B1* **.. note** *B1* = *conjunctD2*[*OF this*,*rule-format*]
note *obt*(*2*)[*unfolded has-integral-alt'*[*of h*]] **note** *conjunctD2*[*OF this*, *rule-format*]
note *h* = *this*(*1*) **and** *this*(*2*)[*OF* ∗] **from** *this*(*2*) **guess** *B2* **.. note** *B2* = *conjunctD2*[*OF this*,*rule-format*]

def $c \equiv \chi\ i.\ min\ (a\$i)\ (-\ (max\ B1\ B2))$ **and** $d \equiv \chi\ i.\ max\ (b\$i)\ (max\ B1\ B2)$
have $*$:*ball 0 B1* $\subseteq \{c..d\}$ *ball 0 B2* $\subseteq \{c..d\}$ **apply** *safe* **unfolding** *mem-ball mem-interval dist-norm*
proof(*rule-tac*[!] *allI*)
case *goal1* **thus** *?case* **using** *component-le-norm*[*of x i*] **unfolding** *c-def d-def* **by** *auto* **next**
case *goal2* **thus** *?case* **using** *component-le-norm*[*of x i*] **unfolding** *c-def d-def* **by** *auto* **qed**
have $**$:$\bigwedge$*ch cg ag ah*::*real*. $norm(ah - ag) \leq norm(ch - cg) \Longrightarrow norm(cg - i) < e\ /\ 4 \Longrightarrow$
$norm(ch - j) < e\ /\ 4 \Longrightarrow norm(ag - ah) < e$
using *obt*(*3*) **unfolding** *real-norm-def* **by** *arith*
show *?case* **apply**(*rule-tac x*=$\lambda x.$ *if x* $\in$ *s then g x else 0* **in** *exI*)
apply(*rule-tac x*=$\lambda x.$ *if x* $\in$ *s then h x else 0* **in** *exI*)
apply(*rule-tac x*=*integral* $\{a..b\}$ $(\lambda x.$ *if x* $\in$ *s then g x else 0*) **in** *exI*)
apply(*rule-tac x*=*integral* $\{a..b\}$ $(\lambda x.$ *if x* $\in$ *s then h x else 0*) **in** *exI*)
apply *safe* **apply**(*rule-tac*[*1*−*2*] *integrable-integral*,*rule g*,*rule h*)
apply(*rule* $**$[*OF - B1*(*2*)[*OF* $*$(*1*)] *B2*(*2*)[*OF* $*$(*2*)]])
proof− **have** $*$:$\bigwedge$*x f g*. (*if x* $\in$ *s then f x else 0*) − (*if x* $\in$ *s then g x else 0*) = (*if x* $\in$ *s then f x* − *g x else* (*0*::*real*)) **by** *auto*
note $**$ = *abs-of-nonneg*[*OF integral-nonneg*[*OF integrable-sub*, *OF h g*]]
show *norm* (*integral* $\{a..b\}$ $(\lambda x.$ *if x* $\in$ *s then h x else 0*) −
integral $\{a..b\}$ $(\lambda x.$ *if x* $\in$ *s then g x else 0*))
$\leq$ *norm* (*integral* $\{c..d\}$ $(\lambda x.$ *if x* $\in$ *s then h x else 0*) −
integral $\{c..d\}$ $(\lambda x.$ *if x* $\in$ *s then g x else 0*))
unfolding *integral-sub*[*OF h g*,*THEN sym*] *real-norm-def* **apply**(*subst* $**$) **defer apply**(*subst* $**$) **defer**
apply(*rule has-integral-subset-le*) **defer apply**(*rule integrable-integral integrable-sub h g*)+
proof *safe* **fix** *x* **assume** *x*$\in\{a..b\}$ **thus** *x*$\in\{c..d\}$ **unfolding** *mem-interval c-def d-def*
apply − **apply** *rule* **apply**(*erule-tac x*=*i* **in** *allE*) **by** *auto*
qed(*insert obt*(*4*), *auto*) **qed**(*insert obt*(*4*), *auto*) **qed note** *interv* = *this*

show *?thesis* **unfolding** *integrable-alt*[*of f*] **apply** *safe* **apply**(*rule interv*)
proof− **case** *goal1* **hence** $*$:*e*/*3* > *0* **by** *auto*
from *assms*[*rule-format*,*OF this*] **guess** *g h i j* **apply**−**by**(*erule exE conjE*)+ **note** *obt*=*this*
note *obt*(*1*)[*unfolded has-integral-alt'*[*of g*]] **note** *conjunctD2*[*OF this*, *rule-format*]
note *g* = *this*(*1*) **and** *this*(*2*)[*OF* $*$] **from** *this*(*2*) **guess** *B1* **..** **note** *B1* = *conjunctD2*[*OF this*,*rule-format*]
note *obt*(*2*)[*unfolded has-integral-alt'*[*of h*]] **note** *conjunctD2*[*OF this*, *rule-format*]
note *h* = *this*(*1*) **and** *this*(*2*)[*OF* $*$] **from** *this*(*2*) **guess** *B2* **..** **note** *B2* = *conjunctD2*[*OF this*,*rule-format*]
show *?case* **apply**(*rule-tac x*=*max B1 B2* **in** *exI*) **apply** *safe* **apply**(*rule min-max.less-supI1*,*rule B1*)
proof− **fix** *a b c d*::*real*^'*n* **assume** *as*:*ball 0* (*max B1 B2*) $\subseteq \{a..b\}$ *ball 0* (*max B1 B2*) $\subseteq \{c..d\}$

have $**$:*ball 0 B1* $\subseteq$ *ball* (*0*::*real^'n*) (*max B1 B2*) *ball 0 B2* $\subseteq$ *ball* (*0*::*real^'n*) (*max B1 B2*) **by** *auto*
have $*$:$\bigwedge$*ga gc ha hc fa fc*::*real*. *abs*(*ga* − *i*) < *e* / *3* ∧ *abs*(*gc* − *i*) < *e* / *3* ∧ *abs*(*ha* − *j*) < *e* / *3* ∧
abs(*hc* − *j*) < *e* / *3* ∧ *abs*(*i* − *j*) < *e* / *3* ∧ *ga* ≤ *fa* ∧ *fa* ≤ *ha* ∧ *gc* ≤ *fc* ∧ *fc* ≤ *hc*⟹ *abs*(*fa* − *fc*) < *e* **by** *smt*
show *norm* (*integral* {*a*..*b*} (λ*x*. *if x* ∈ *s then f x else 0*) − *integral* {*c*..*d*} (λ*x*. *if x* ∈ *s then f x else 0*)) < *e*
unfolding *real-norm-def* **apply**(*rule* $*$, *safe*) **unfolding** *real-norm-def*[*THEN sym*]
apply(*rule B1*(*2*),*rule order-trans*,*rule* $**$,*rule as*(*1*))
apply(*rule B1*(*2*),*rule order-trans*,*rule* $**$,*rule as*(*2*))
apply(*rule B2*(*2*),*rule order-trans*,*rule* $**$,*rule as*(*1*))
apply(*rule B2*(*2*),*rule order-trans*,*rule* $**$,*rule as*(*2*))
apply(*rule obt*) **apply**(*rule-tac*[!] *integral-le*) **using** *obt*
by(*auto intro*!: *h g interv*) **qed qed qed**

23.62 Adding integrals over several sets.

lemma *has-integral-union*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes (*f has-integral i*) *s* (*f has-integral j*) *t negligible*(*s* ∩ *t*)
shows (*f has-integral* (*i* + *j*)) (*s* ∪ *t*)
proof− **note** $*$ = *has-integral-restrict-univ*[*THEN sym*, *of f*]
show *?thesis* **unfolding** $*$ **apply**(*rule has-integral-spike*[*OF assms*(*3*)])
defer apply(*rule has-integral-add*[*OF assms*(*1*−*2*)[*unfolded* $*$]]) **by** *auto* **qed**

lemma *has-integral-unions*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *finite t* ∀*s*∈*t*. (*f has-integral* (*i s*)) *s* ∀*s*∈*t*. ∀*s'*∈*t*. ~(*s* = *s'*) ⟶ *negligible*(*s* ∩ *s'*)
shows (*f has-integral* (*setsum i t*)) (⋃*t*)
proof− **note** $*$ = *has-integral-restrict-univ*[*THEN sym*, *of f*]
have $**$:*negligible* (⋃((λ(*a*,*b*). *a* ∩ *b*) ' {(*a*,*b*). *a* ∈ *t* ∧ *b* ∈ {*y*. *y* ∈ *t* ∧ ~(*a* = *y*)}}))
apply(*rule negligible-unions*) **apply**(*rule finite-imageI*) **apply**(*rule finite-subset*[*of* - *t* × *t*]) **defer**
apply(*rule finite-cartesian-product*[*OF assms*(*1*,*1*)]) **using** *assms*(*3*) **by** *auto*
note *assms*(*2*)[*unfolded* $*$] **note** *has-integral-setsum*[*OF assms*(*1*) *this*]
thus *?thesis* **unfolding** $*$ **apply**−**apply**(*rule has-integral-spike*[*OF* $**$]) **defer apply** *assumption*
proof *safe* **case** *goal1* **thus** *?case*
proof(*cases x*∈⋃*t*) **case** *True* **then guess** *s* **unfolding** *Union-iff* **.. note** *s*=*this*
hence $*$:∀*b*∈*t*. *x* ∈ *b* ⟷ *b* = *s* **using** *goal1*(*3*) **by** *blast*
show *?thesis* **unfolding** *if-P*[*OF True*] **apply**(*rule trans*) **defer**
apply(*rule setsum-cong2*) **apply**(*subst* $*$, *assumption*) **apply**(*rule refl*)
unfolding *setsum-delta*[*OF assms*(*1*)] **using** *s* **by** *auto* **qed** *auto* **qed qed**

23.63 In particular adding integrals over a division, maybe not of an interval.

lemma *has-integral-combine-division*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *d division-of s* ∀ *k*∈*d*. (*f has-integral* (*i k*)) *k*
shows (*f has-integral* (*setsum i d*)) *s*
proof− **note** *d* = *division-ofD*[*OF assms*(*1*)]
show *?thesis* **unfolding** *d*(*6*)[*THEN sym*] **apply**(*rule has-integral-unions*)
apply(*rule d assms*)+ **apply**(*rule*,*rule*,*rule*)
proof− **case** *goal1* **from** *d*(*4*)[*OF this*(*1*)] *d*(*4*)[*OF this*(*2*)]
guess *a c b d* **apply**−**by**(*erule exE*)+ **note** *obt*=*this*
from *d*(*5*)[*OF goal1*] **show** *?case* **unfolding** *obt interior-closed-interval*
apply−**apply**(*rule negligible-subset*[*of* ({*a*..*b*}−{*a*<..<*b*}) ∪ ({*c*..*d*}−{*c*<..<*d*})])
apply(*rule negligible-union negligible-frontier-interval*)+ **by** *auto* **qed qed**

lemma *integral-combine-division-bottomup*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *d division-of s* ∀ *k*∈*d*. *f integrable-on k*
shows *integral s f* = *setsum* (λ*i*. *integral i f*) *d*
apply(*rule integral-unique*) **apply**(*rule has-integral-combine-division*[*OF assms*(*1*)])
using *assms*(*2*) **unfolding** *has-integral-integral* **.**

lemma *has-integral-combine-division-topdown*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *f integrable-on s d division-of k k* ⊆ *s*
shows (*f has-integral* (*setsum* (λ*i*. *integral i f*) *d*)) *k*
apply(*rule has-integral-combine-division*[*OF assms*(*2*)])
apply *safe* **unfolding** *has-integral-integral*[*THEN sym*]
proof− **case** *goal1* **from** *division-ofD*(*2*,*4*)[*OF assms*(*2*) *this*]
show *?case* **apply** *safe* **apply**(*rule integrable-on-subinterval*)
apply(*rule assms*) **using** *assms*(*3*) **by** *auto* **qed**

lemma *integral-combine-division-topdown*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *f integrable-on s d division-of s*
shows *integral s f* = *setsum* (λ*i*. *integral i f*) *d*
apply(*rule integral-unique*,*rule has-integral-combine-division-topdown*) **using** *assms*
by *auto*

lemma *integrable-combine-division*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *d division-of s* ∀ *i*∈*d*. *f integrable-on i*
shows *f integrable-on s*
using *assms*(*2*) **unfolding** *integrable-on-def*
by(*metis has-integral-combine-division*[*OF assms*(*1*)])

lemma *integrable-on-subdivision*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *d division-of i f integrable-on s i* ⊆ *s*
shows *f integrable-on i*
apply(*rule integrable-combine-division assms*)+
proof *safe* **case** *goal1* **note** *division-ofD*(*2*,*4*)[*OF assms*(*1*) *this*]
thus *?case* **apply** *safe* **apply**(*rule integrable-on-subinterval*[*OF assms*(*2*)])
using *assms*(*3*) **by** *auto* **qed**

23.64 Also tagged divisions.

lemma *has-integral-combine-tagged-division*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *p tagged-division-of s* ∀(*x*,*k*) ∈ *p*. (*f has-integral* (*i k*)) *k*
shows (*f has-integral* (*setsum* (λ(*x*,*k*). *i k*) *p*)) *s*
proof− **have** ∗:(*f has-integral* (*setsum* (λ*k*. *integral k f*) (*snd* ‘ *p*))) *s*
apply(*rule has-integral-combine-division*) **apply**(*rule division-of-tagged-division*[*OF assms*(*1*)])
using *assms*(*2*) **unfolding** *has-integral-integral*[*THEN sym*] **by**(*safe*,*auto*)
thus *?thesis* **apply**− **apply**(*rule subst*[**where** *P*=λ*i*. (*f has-integral i*) *s*]) **defer**
apply *assumption*
apply(*rule trans*[*of* - *setsum* (λ(*x*,*k*). *integral k f*) *p*]) **apply**(*subst eq-commute*)
apply(*rule setsum-over-tagged-division-lemma*[*OF assms*(*1*)]) **apply**(*rule integral-null*,*assumption*)
apply(*rule setsum-cong2*) **using** *assms*(*2*) **by** *auto* **qed**

lemma *integral-combine-tagged-division-bottomup*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *p tagged-division-of* {*a*..*b*} ∀(*x*,*k*)∈*p*. *f integrable-on k*
shows *integral* {*a*..*b*} *f* = *setsum* (λ(*x*,*k*). *integral k f*) *p*
apply(*rule integral-unique*) **apply**(*rule has-integral-combine-tagged-division*[*OF assms*(*1*)])
using *assms*(*2*) **by** *auto*

lemma *has-integral-combine-tagged-division-topdown*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *f integrable-on* {*a*..*b*} *p tagged-division-of* {*a*..*b*}
shows (*f has-integral* (*setsum* (λ(*x*,*k*). *integral k f*) *p*)) {*a*..*b*}
apply(*rule has-integral-combine-tagged-division*[*OF assms*(*2*)])
proof *safe* **case** *goal1* **note** *tagged-division-ofD*(*3*−*4*)[*OF assms*(*2*) *this*]
thus *?case* **using** *integrable-subinterval*[*OF assms*(*1*)] **by** *auto* **qed**

lemma *integral-combine-tagged-division-topdown*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *f integrable-on* {*a*..*b*} *p tagged-division-of* {*a*..*b*}
shows *integral* {*a*..*b*} *f* = *setsum* (λ(*x*,*k*). *integral k f*) *p*
apply(*rule integral-unique*,*rule has-integral-combine-tagged-division-topdown*) **using** *assms* **by** *auto*

23.65 Henstock's lemma.

lemma *henstock-lemma-part1*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *f integrable-on* {*a*..*b*} *0* < *e gauge d*
(∀*p*. *p tagged-division-of* {*a*..*b*} ∧ *d fine p* ⟶ *norm* (*setsum* (λ(*x*,*k*). *content k* ∗*R f x*) *p* − *integral*({*a*..*b*}) *f*) < *e*)
and *p*:*p tagged-partial-division-of* {*a*..*b*} *d fine p*
shows *norm*(*setsum* (λ(*x*,*k*). *content k* ∗*R f x* − *integral k f*) *p*) ≤ *e* (**is** *?x* ≤ *e*)
proof− { **presume** ⋀*k*. *0*<*k* ⟹ *?x* ≤ *e* + *k* **thus** *?thesis* **by** *arith* }
fix *k*::*real* **assume** *k*:*k*>*0* **note** *p'* = *tagged-partial-division-ofD*[*OF p*(*1*)]
have ⋃ *snd* ‘ *p* ⊆ {*a*..*b*} **using** *p'*(*3*) **by** *fastsimp*
note *partial-division-of-tagged-division*[*OF p*(*1*)] *this*
from *partial-division-extend-interval*[*OF this*] **guess** *q* **. note** *q*=*this* **and** *q'* = *division-ofD*[*OF this*(*2*)]
def *r* ≡ *q* − *snd* ‘ *p* **have** *snd* ‘ *p* ∩ *r* = {} **unfolding** *r-def* **by** *auto*

have *r*:*finite r* **using** *q′* **unfolding** *r-def* **by** *auto*

have ∀ *i*∈*r*. ∃ *p*. *p tagged-division-of i* ∧ *d fine p* ∧
norm(*setsum* (λ(*x*,*j*). *content j* ∗*R f x*) *p* − *integral i f*) < *k* / (*real* (*card r*) + *1*)
proof *safe* **case** *goal1* **hence** *i*:*i* ∈ *q* **unfolding** *r-def* **by** *auto*
from *q′*(*4*)[*OF this*] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
have ∗:*k* / (*real* (*card r*) + *1*) > *0* **apply**(*rule divide-pos-pos*,*rule k*) **by** *auto*
have *f integrable-on* {*u*..*v*} **apply**(*rule integrable-subinterval*[*OF assms*(*1*)])
using *q′*(*2*)[*OF i*] **unfolding** *uv* **by** *auto*
note *integrable-integral*[*OF this*, *unfolded has-integral*[*of f*]]
from *this*[*rule-format*,*OF* ∗] **guess** *dd* **..** **note** *dd*=*conjunctD2*[*OF this*,*rule-format*]
note *gauge-inter*[*OF* ‹*gauge d*› *dd*(*1*)] **from** *fine-division-exists*[*OF this*,*of u v*]
guess *qq* **.**
thus *?case* **apply**(*rule-tac x*=*qq* **in** *exI*) **using** *dd*(*2*)[*of qq*] **unfolding** *fine-inter uv* **by** *auto* **qed**
from *bchoice*[*OF this*] **guess** *qq* **..** **note** *qq*=*this*[*rule-format*]

let *?p* = *p* ∪ ⋃ *qq* ‘ *r* **have** *norm* ((∑ (*x*, *k*)∈*?p*. *content k* ∗*R f x*) − *integral* {*a*..*b*} *f*) < *e*
apply(*rule assms*(*4*)[*rule-format*])
proof show *d fine ?p* **apply**(*rule fine-union*,*rule p*) **apply**(*rule fine-unions*) **using** *qq* **by** *auto*
note ∗ = *tagged-partial-division-of-union-self*[*OF p*(*1*)]
have *p* ∪ ⋃ *qq* ‘ *r tagged-division-of* ⋃ *snd* ‘ *p* ∪ ⋃ *r*
proof(*rule tagged-division-union*[*OF* ∗ *tagged-division-unions*])
show *finite r* **by** *fact* **case** *goal2* **thus** *?case* **using** *qq* **by** *auto*
next case *goal3* **thus** *?case* **apply**(*rule*,*rule*,*rule*) **apply**(*rule q′*(*5*)) **unfolding** *r-def* **by** *auto*
next case *goal4* **thus** *?case* **apply**(*rule inter-interior-unions-intervals*) **apply**(*fact*,*rule*)
apply(*rule*,*rule q′*) **defer apply**(*rule*,*subst Int-commute*)
apply(*rule inter-interior-unions-intervals*) **apply**(*rule finite-imageI*,*rule p′*,*rule*) **defer**
apply(*rule*,*rule q′*) **using** *q*(*1*) *p′* **unfolding** *r-def* **by** *auto* **qed**
moreover have ⋃ *snd* ‘ *p* ∪ ⋃ *r* = {*a*..*b*} {*qq i* |*i*. *i* ∈ *r*} = *qq* ‘ *r*
unfolding *Union-Un-distrib*[*THEN sym*] *r-def* **using** *q* **by** *auto*
ultimately show *?p tagged-division-of* {*a*..*b*} **by** *fastsimp* **qed**

hence *norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗*R f x*) + (∑ (*x*, *k*)∈⋃ *qq* ‘ *r*. *content k* ∗*R f x*) −
integral {*a*..*b*} *f*) < *e* **apply**(*subst setsum-Un-zero*[*THEN sym*]) **apply**(*rule p′*) **prefer** *3*
apply *assumption* **apply** *rule* **apply**(*rule finite-imageI*,*rule r*) **apply** *safe* **apply**(*drule qq*)
proof− **fix** *x l k* **assume** *as*:(*x*,*l*)∈*p* (*x*,*l*)∈*qq k k*∈*r*
note *qq*[*OF this*(*3*)] **note** *tagged-division-ofD*(*3*,*4*)[*OF conjunct1*[*OF this*] *as*(*2*)]
from *this*(*2*) **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*

have $l{\in}snd\ `\ p$ **unfolding** *image-iff* **apply**(*rule-tac* $x{=}(x,l)$ **in** *bexI*) **using** *as* **by** *auto*
hence $l{\in}q\ k{\in}q\ l{\neq}k$ **using** *as*(*1*,*3*) *q*(*1*) **unfolding** *r-def* **by** *auto*
note $q'(5)[OF\ this]$ **hence** *interior* $l = \{\}$ **using** *subset-interior*$[OF\ \langle l \subseteq k\rangle]$ **by** *blast*
thus *content* $l *_R f\ x = 0$ **unfolding** *uv content-eq-0-interior*[*THEN sym*] **by** *auto* **qed** *auto*

hence *norm* $((\sum(x,\ k){\in}p.\ content\ k *_R f\ x) + setsum\ (setsum\ (\lambda(x,\ k).\ content\ k *_R f\ x))$
$(qq\ `\ r) - integral\ \{a..b\}\ f) < e$ **apply**(*subst*(*asm*) *setsum-UNION-zero*)
prefer *4* **apply** *assumption* **apply**(*rule finite-imageI*,*fact*)
unfolding *split-paired-all split-conv image-iff* **defer apply**(*erule bexE*)+
proof− **fix** *x m k l T1 T2* **assume** $(x,m){\in}T1\ (x,m){\in}T2\ T1{\neq}T2\ k{\in}r\ l{\in}r\ T1 = qq\ k\ T2 = qq\ l$
note *as* = *this*(*1*−*5*)[*unfolded this*(*6*−)] **note** *kl* = *tagged-division-ofD*(*3*,*4*)[*OF qq*[*THEN conjunct1*]]
from *this*(*2*)[*OF as*(*4*,*1*)] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
have $*$:*interior* $(k \cap l) = \{\}$ **unfolding** *interior-inter* **apply**(*rule q'*)
using *as* **unfolding** *r-def* **by** *auto*
have *interior m* $= \{\}$ **unfolding** *subset-empty*[*THEN sym*] **unfolding** $*$[*THEN sym*]
apply(*rule subset-interior*) **using** *kl*(*1*)[*OF as*(*4*,*1*)] *kl*(*1*)[*OF as*(*5*,*2*)] **by** *auto*
thus *content m* $*_R$ *f x* $= 0$ **unfolding** *uv content-eq-0-interior*[*THEN sym*] **by** *auto*
qed(*insert qq*, *auto*)

hence $**$:*norm* $((\sum(x,\ k){\in}p.\ content\ k *_R f\ x) + setsum\ (setsum\ (\lambda(x,\ k).\ content\ k *_R f\ x) \circ qq)\ r -$
$integral\ \{a..b\}\ f) < e$ **apply**(*subst*(*asm*) *setsum-reindex-nonzero*) **apply** *fact*
apply(*rule setsum-0'*,*rule*) **unfolding** *split-paired-all split-conv* **defer apply** *assumption*
proof− **fix** *k l x m* **assume** *as*:$k{\in}r\ l{\in}r\ k{\neq}l\ qq\ k = qq\ l\ (x,m){\in}qq\ k$
note *tagged-division-ofD*(*6*)[*OF qq*[*THEN conjunct1*]] **from** *this*[*OF as*(*1*)] *this*[*OF as*(*2*)]
show *content m* $*_R$ *f x* $= 0$ **using** *as*(*3*) **unfolding** *as* **by** *auto* **qed**

have $*$:$\bigwedge ir\ ip\ i\ cr\ cp.\ norm((cp + cr) - i) < e \Longrightarrow norm(cr - ir) < k \Longrightarrow$
$ip + ir = i \Longrightarrow norm(cp - ip) \leq e + k$
proof− **case** *goal1* **thus** *?case* **using** *norm-triangle-le*[*of cp* + *cr* − *i* − (*cr* − *ir*)]
unfolding *goal1*(*3*)[*THEN sym*] *norm-minus-cancel* **by**(*auto simp add*:*algebra-simps*)
qed

have *?x* = $norm\ ((\sum(x,\ k){\in}p.\ content\ k *_R f\ x) - (\sum(x,\ k){\in}p.\ integral\ k\ f))$
unfolding *split-def setsum-subtractf* **..**
also have ... $\leq e + k$ **apply**(*rule* $*$[*OF* $**$, **where** *ir*=*setsum* $(\lambda k.\ integral\ k\ f)$ *r*])

proof− **case** *goal2* **have** ∗:$(\sum (x,\ k){\in}p.\ integral\ k\ f) = (\sum k{\in}snd\ `\ p.\ integral\ k\ f)$
apply(*subst setsum-reindex-nonzero*) **apply** *fact*
unfolding *split-paired-all snd-conv split-def o-def*
proof− **fix** *x l y m* **assume** *as*:$(x,l){\in}p\ (y,m){\in}p\ (x,l){\neq}(y,m)\ l = m$
from *p′*(*4*)[*OF as*(*1*)] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
show *integral l f = 0* **unfolding** *uv* **apply**(*rule integral-unique*)
apply(*rule has-integral-null*) **unfolding** *content-eq-0-interior*
using *p′*(*5*)[*OF as*(*1*−*3*)] **unfolding** *uv as*(*4*)[*THEN sym*] **by** *auto*
qed *auto*
show *?case* **unfolding** *integral-combine-division-topdown*[*OF assms*(*1*) *q*(*2*)] ∗ *r-def*
apply(*rule setsum-Un-disjoint′*[*THEN sym*]) **using** *q*(*1*) *q′*(*1*) *p′*(*1*) **by** *auto*
next **case** *goal1* **have** ∗:$k * real\ (card\ r)\ /\ (1 + real\ (card\ r)) < k$ **using** *k* **by**(*auto simp add*:*field-simps*)
show *?case* **apply**(*rule le-less-trans*[*of - setsum* (λ*x*. *k* / (*real* (*card r*) + *1*)) *r*])
unfolding *setsum-subtractf*[*THEN sym*] **apply**(*rule setsum-norm-le*,*fact*)
apply *rule* **apply**(*drule qq*) **defer unfolding** *divide-inverse setsum-left-distrib*[*THEN sym*]
unfolding *divide-inverse*[*THEN sym*] **using** ∗ **by**(*auto simp add*:*field-simps real-eq-of-nat*)
qed finally show *?x* ≤ *e* + *k* **. qed**

lemma *henstock-lemma-part2*: **fixes** *f*::*real^′m* ⇒ *real^′n*
assumes *f integrable-on* {*a*..*b*} *0* < *e gauge d*
∀ *p*. *p tagged-division-of* {*a*..*b*} ∧ *d fine p* ⟶ *norm* (*setsum* (λ(*x*,*k*). *content k* ∗$_R$ *f x*) *p* −
integral({*a*..*b*}) *f*) < *e* *p tagged-partial-division-of* {*a*..*b*} *d fine p*
shows *setsum* (λ(*x*,*k*). *norm*(*content k* ∗$_R$ *f x* − *integral k f*)) *p* ≤ *2* ∗ *real* (*CARD*(*′n*)) ∗ *e*
unfolding *split-def* **apply**(*rule vsum-norm-allsubsets-bound*) **defer**
apply(*rule henstock-lemma-part1*[*unfolded split-def*,*OF assms*(*1*−*3*)])
apply *safe* **apply**(*rule assms*[*rule-format*,*unfolded split-def*]) **defer**
apply(*rule tagged-partial-division-subset*,*rule assms*,*assumption*)
apply(*rule fine-subset*,*assumption*,*rule assms*) **using** *assms*(*5*) **by** *auto*

lemma *henstock-lemma*: **fixes** *f*::*real^′m* ⇒ *real^′n*
assumes *f integrable-on* {*a*..*b*} *e*>*0*
obtains *d* **where** *gauge d*
∀ *p*. *p tagged-partial-division-of* {*a*..*b*} ∧ *d fine p*
⟶ *setsum* (λ(*x*,*k*). *norm*(*content k* ∗$_R$ *f x* − *integral k f*)) *p* < *e*
proof− **have** ∗:$e\ /\ (2 * (real\ CARD(′n) + 1)) > 0$ **apply**(*rule divide-pos-pos*) **using** *assms*(*2*) **by** *auto*
from *integrable-integral*[*OF assms*(*1*),*unfolded has-integral*[*of f*],*rule-format*,*OF this*]
guess *d* **.. note** *d* = *conjunctD2*[*OF this*] **show** *thesis* **apply**(*rule that*,*rule d*)
proof *safe* **case** *goal1* **note** ∗ = *henstock-lemma-part2*[*OF assms*(*1*) ∗ *d this*]
show *?case* **apply**(*rule le-less-trans*[*OF* ∗]) **using** ‹*e*>*0*› **by**(*auto simp add*:*field-simps*)

qed qed

23.66 monotone convergence (bounded interval first).

lemma *monotone-convergence-interval*: **fixes** f::*nat* ⇒ *real^'n* ⇒ *real^1*
assumes ∀ *k*. (*f k*) *integrable-on* {*a*..*b*}
∀ *k*. ∀ *x*∈{*a*..*b*}. *dest-vec1*(*f k x*) ≤ *dest-vec1*(*f* (*Suc k*) *x*)
∀ *x*∈{*a*..*b*}. ((λ*k*. *f k x*) −−−> *g x*) *sequentially*
bounded {*integral* {*a*..*b*} (*f k*) | *k* . *k* ∈ *UNIV*}
shows *g integrable-on* {*a*..*b*} ∧ ((λ*k*. *integral* ({*a*..*b*}) (*f k*)) −−−> *integral* ({*a*..*b*}) *g*) *sequentially*
proof(*case-tac*[!] *content* {*a*..*b*} = *0*) **assume** *as*:*content* {*a*..*b*} = *0*
show *?thesis* **using** *integrable-on-null*[*OF as*] **unfolding** *integral-null*[*OF as*]
using *Lim-const* **by** *auto*
next assume *ab*:*content* {*a*..*b*} ≠ *0*
have *fg*:∀ *x*∈{*a*..*b*}. ∀ *k*. (*f k x*)$*1* ≤ (*g x*)$*1*
proof *safe* **case** *goal1* **note** *assms*(*3*)[*rule-format*,*OF this*]
note ∗ = *Lim-component-ge*[*OF this trivial-limit-sequentially*]
show *?case* **apply**(*rule* ∗) **unfolding** *eventually-sequentially*
apply(*rule-tac x*=*k* **in** *exI*) **apply**− **apply**(*rule transitive-stepwise-le*)
using *assms*(*2*)[*rule-format*,*OF goal1*] **by** *auto* **qed**
have ∃ *i*. ((λ*k*. *integral* ({*a*..*b*}) (*f k*)) −−−> *i*) *sequentially*
apply(*rule bounded-increasing-convergent*) **defer**
apply *rule* **apply**(*rule integral-component-le*) **apply** *safe*
apply(*rule assms*(*1*−*2*)[*rule-format*])+ **using** *assms*(*4*) **by** *auto*
then guess *i* **..** **note** *i*=*this*
have *i'*:⋀*k*. *dest-vec1*(*integral*({*a*..*b*}) (*f k*)) ≤ *dest-vec1 i*
apply(*rule Lim-component-ge*,*rule i*) **apply**(*rule trivial-limit-sequentially*)
unfolding *eventually-sequentially* **apply**(*rule-tac x*=*k* **in** *exI*)
apply(*rule transitive-stepwise-le*) **prefer** *3* **apply**(*rule integral-dest-vec1-le*)
apply(*rule assms*(*1*−*2*)[*rule-format*])+ **using** *assms*(*2*) **by** *auto*

have (*g has-integral i*) {*a*..*b*} **unfolding** *has-integral*
proof *safe* **case** *goal1* **note** *e*=*this*
hence ∀ *k*. (∃ *d*. *gauge d* ∧ (∀ *p*. *p tagged-division-of* {*a*..*b*} ∧ *d fine p* ⟶
norm ((∑ (*x*, *ka*)∈*p*. *content ka* ∗$_R$ *f k x*) − *integral* {*a*..*b*} (*f k*)) < *e* / *2* ^ (*k* + *2*)))
apply−**apply**(*rule*,*rule assms*(*1*)[*unfolded has-integral-integral has-integral*,*rule-format*])
apply(*rule divide-pos-pos*) **by** *auto*
from *choice*[*OF this*] **guess** *c* **..** **note** *c*=*conjunctD2*[*OF this*[*rule-format*],*rule-format*]

have ∃ *r*. ∀ *k*≥*r*. *0* ≤ *i*$*1* − *dest-vec1*(*integral* {*a*..*b*} (*f k*)) ∧
i$*1* − *dest-vec1*(*integral* {*a*..*b*} (*f k*)) < *e* / *4*
proof− **case** *goal1* **have** *e*/*4* > *0* **using** *e* **by** *auto*
from *i*[*unfolded Lim-sequentially*,*rule-format*,*OF this*] **guess** *r* **..**
thus *?case* **apply**(*rule-tac x*=*r* **in** *exI*) **apply** *rule*
apply(*erule-tac x*=*k* **in** *allE*)
proof− **case** *goal1* **thus** *?case* **using** *i'*[*of k*] **unfolding** *dist-real* **by** *auto*
qed qed

then guess r **.. note** r=*conjunctD2*[*OF this*[*rule-format*]]

have $\forall x{\in}\{a..b\}.\ \exists n{\geq}r.\ \forall k{\geq}n.\ 0 \leq (g\ x)\$1 - (f\ k\ x)\$1 \wedge$
$(g\ x)\$1 - (f\ k\ x)\$1 < e\ /\ (4 * content(\{a..b\}))$
proof case *goal1* **have** $e\ /\ (4 * content\ \{a..b\}) > 0$ **apply**(*rule divide-pos-pos*,*fact*)
using *ab content-pos-le*[*of a b*] **by** *auto*
from *assms*(*3*)[*rule-format*,*OF goal1*,*unfolded Lim-sequentially*,*rule-format*,*OF this*]
guess n **.. note** n=*this*
thus *?case* **apply**(*rule-tac* x=$n + r$ **in** *exI*) **apply** *safe* **apply**(*erule-tac*[*2*−*3*] x=k **in** *allE*)
unfolding *dist-real* **using** *fg*[*rule-format*,*OF goal1*] **by**(*auto simp add*:*field-simps*)
qed
from *bchoice*[*OF this*] **guess** m **.. note** m=*conjunctD2*[*OF this*[*rule-format*],*rule-format*]
def $d \equiv \lambda x.\ c\ (m\ x)\ x$

show *?case* **apply**(*rule-tac* x=d **in** *exI*)
proof *safe* **show** *gauge d* **using** *c*(*1*) **unfolding** *gauge-def d-def* **by** *auto*
next fix p **assume** *p*:*p tagged-division-of* $\{a..b\}$ *d fine p*
note p'=*tagged-division-ofD*[*OF p*(*1*)]
have $\exists a.\ \forall x{\in}p.\ m\ (fst\ x) \leq a$ **by**(*rule upper-bound-finite-set*,*fact*)
then guess s **.. note** s=*this*
have $*$:$\forall\ a\ b\ c\ d.\ norm(a - b) \leq e\ /\ 4 \wedge norm(b - c) < e\ /\ 2 \wedge$
$norm(c - d) < e\ /\ 4 \longrightarrow norm(a - d) < e$
proof *safe* **case** *goal1* **thus** *?case* **using** *norm-triangle-lt*[*of a − b b − c 3*∗ *e*/*4*]
norm-triangle-lt[*of a − b* + (*b − c*) *c − d e*] **unfolding** *norm-minus-cancel*
by(*auto simp add*:*algebra-simps*) **qed**
show *norm* $((\sum(x,\ k){\in}p.\ content\ k *_R g\ x) - i) < e$ **apply**(*rule* $*$[*rule-format*,**where**
b=$\sum(x,\ k){\in}p.\ content\ k *_R f\ (m\ x)\ x$ **and** c=$\sum(x,\ k){\in}p.\ integral\ k\ (f\ (m\ x))$])
proof *safe* **case** *goal1*
show *?case* **apply**(*rule order-trans*[*of* - $\sum(x,\ k){\in}p.\ content\ k * (e\ /\ (4 * content\ \{a..b\}))$])
unfolding *setsum-subtractf*[*THEN sym*] **apply**(*rule order-trans*,*rule setsum-norm*[*OF p*′(*1*)])
apply(*rule setsum-mono*) **unfolding** *split-paired-all split-conv*
unfolding *split-def setsum-left-distrib*[*THEN sym*] *scaleR.diff-right*[*THEN sym*]
unfolding *additive-content-tagged-division*[*OF p*(*1*), *unfolded split-def*]
proof− **fix** $x\ k$ **assume** *xk*:$(x,k) \in p$ **hence** x:$x{\in}\{a..b\}$ **using** p'(*2*−*3*)[*OF xk*] **by** *auto*
from p'(*4*)[*OF xk*] **guess** $u\ v$ **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
show $norm\ (content\ k *_R (g\ x - f\ (m\ x)\ x)) \leq content\ k * (e\ /\ (4 * content\ \{a..b\}))$
unfolding *norm-scaleR uv* **unfolding** *abs-of-nonneg*[*OF content-pos-le*]
apply(*rule mult-left-mono*) **unfolding** *norm-real* **using** *m*(*2*)[*OF x*,*of m x*] **by** *auto*

qed(*insert ab*,*auto*)

next case *goal2* **show** *?case* **apply**(*rule le-less-trans*[*of - norm* ($\sum j = 0..s$. $\sum (x,\ k) \in \{xk \in p.\ m\ (fst\ xk) = j\}$. *content k* $*_R$ *f* (*m x*) *x* − *integral k* (*f* (*m x*)))])
apply(*subst setsum-group*) **apply** *fact* **apply**(*rule finite-atLeastAtMost*)
defer
apply(*subst split-def*)+ **unfolding** *setsum-subtractf* **apply** *rule*
proof− **show** *norm* ($\sum j = 0..s$. $\sum (x,\ k) \in \{xk \in p.$
m (*fst xk*) = *j*}. *content k* $*_R$ *f* (*m x*) *x* − *integral k* (*f* (*m x*))) < *e* / *2*
apply(*rule le-less-trans*[*of - setsum* ($\lambda i.\ e\ /\ 2\hat{}(i+2)$) {*0..s*}])
apply(*rule setsum-norm-le*[*OF finite-atLeastAtMost*])
proof show ($\sum i = 0..s.\ e\ /\ 2\ \hat{}\ (i + 2)$) < *e* / *2*
unfolding *power-add divide-inverse inverse-mult-distrib*
unfolding *setsum-right-distrib*[*THEN sym*] *setsum-left-distrib*[*THEN sym*]
unfolding *power-inverse sum-gp* **apply**(*rule mult-strict-left-mono*[*OF - e*])
unfolding *power2-eq-square* **by** *auto*
fix *t* **assume** *t*∈{*0..s*}
show *norm* ($\sum (x,\ k) \in \{xk \in p.\ m\ (fst\ xk) = t\}$. *content k* $*_R$ *f* (*m x*) *x* −
integral k (*f* (*m x*))) ≤ *e* / *2* ^ (*t* + *2*)**apply**(*rule order-trans*[*of - norm*(*setsum* ($\lambda(x,k)$. *content k* $*_R$ *f t x* − *integral k* (*f t*)) {*xk* ∈ *p*. *m* (*fst xk*) = *t*})])
apply(*rule eq-refl*) **apply**(*rule arg-cong*[**where** *f*=*norm*]) **apply**(*rule setsum-cong2*) **defer**
apply(*rule henstock-lemma-part1*) **apply**(*rule assms*(*1*)[*rule-format*])
apply(*rule divide-pos-pos*,*rule e*) **defer** **apply** *safe* **apply**(*rule c*)+
apply *rule* **apply** *assumption*+ **apply**(*rule tagged-partial-division-subset*[*of p*])
apply(*rule p*(*1*)[*unfolded tagged-division-of-def*,*THEN conjunct1*])
defer
unfolding *fine-def* **apply** *safe* **apply**(*drule p*(*2*)[*unfolded fine-def*,*rule-format*])
unfolding *d-def* **by** *auto* **qed**
qed(*insert s*, *auto*)

next case *goal3*
note *comb* = *integral-combine-tagged-division-topdown*[*OF assms*(*1*)[*rule-format*] *p*(*1*)]
have *∗*:⋀*sr sx ss ks kr*::*real*^*1*. *kr* = *sr* ⟶ *ks* = *ss* ⟶ *ks* ≤ *i* ∧ *sr* ≤ *sx* ∧ *sx* ≤ *ss* ∧ *0* ≤ *i*$*1* − *kr*$*1*
∧ *i*$*1* − *kr*$*1* < *e*/*4* ⟶ *abs*(*sx*$*1* − *i*$*1*) < *e*/*4* **unfolding** *Cart-eq forall-1 vector-le-def* **by** *arith*
show *?case* **unfolding** *norm-real Cart-nth.diff* **apply**(*rule ∗*[*rule-format*],*safe*)
apply(*rule comb*[*of r*],*rule comb*[*of s*]) **unfolding** *vector-le-def forall-1 setsum-component*
apply(*rule i'*) **apply**(*rule-tac*[*1*−*2*] *setsum-mono*) **unfolding** *split-paired-all split-conv*

apply(*rule-tac*[*1*−*2*] *integral-component-le*[*OF*])
proof *safe* **show** *0* ≤ *i*$*1* − (*integral* {*a*..*b*} (*f r*))$*1* **using** *r*(*1*) **by** *auto*
show *i*$*1* − (*integral* {*a*..*b*} (*f r*))$*1* < *e* / *4* **using** *r*(*2*) **by** *auto*
fix *x k* **assume** *xk*:(*x*,*k*)∈*p* **from** *p*′(*4*)[*OF this*] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
show *f r integrable-on k f s integrable-on k f* (*m x*) *integrable-on k f* (*m x*) *integrable-on k*
unfolding *uv* **apply**(*rule-tac*[!] *integrable-on-subinterval*[*OF assms*(*1*)[*rule-format*]])
using *p*′(*3*)[*OF xk*] **unfolding** *uv* **by** *auto*
fix *y* **assume** *y*∈*k* **hence** *y*∈{*a*..*b*} **using** *p*′(*3*)[*OF xk*] **by** *auto*
hence ∗:⋀*m*. ∀*n*≥*m*. (*f m y*)$*1* ≤ (*f n y*)$*1* **apply**−**apply**(*rule transitive-stepwise-le*) **using** *assms*(*2*) **by** *auto*
show (*f r y*)$*1* ≤ (*f* (*m x*) *y*)$*1* (*f* (*m x*) *y*)$*1* ≤ (*f s y*)$*1*
apply(*rule-tac*[!] ∗[*rule-format*]) **using** *s*[*rule-format*,*OF xk*] *m*(*1*)[*of x*] *p*′(*2*−*3*)[*OF xk*] **by** *auto*
qed qed qed qed note ∗ = *this*

have *integral* {*a*..*b*} *g* = *i* **apply**(*rule integral-unique*) **using** ∗ **.**
thus *?thesis* **using** *i* ∗ **by** *auto* **qed**

lemma *monotone-convergence-increasing*: **fixes** *f*::*nat* ⇒ *real*^′*n* ⇒ *real*^*1*
assumes ∀*k*. (*f k*) *integrable-on s* ∀*k*. ∀*x*∈*s*. (*f k x*)$*1* ≤ (*f* (*Suc k*) *x*)$*1*
∀*x*∈*s*. ((λ*k*. *f k x*) −−−> *g x*) *sequentially bounded* {*integral s* (*f k*)| *k*. *True*}
shows *g integrable-on s* ∧ ((λ*k*. *integral s* (*f k*)) −−−> *integral s g*) *sequentially*
proof− **have** *lem*:⋀*f*::*nat* ⇒ *real*^′*n* ⇒ *real*^*1*. ⋀ *g s*. ∀*k*.∀*x*∈*s*. *0*≤*dest-vec1* (*f k x*) ⟹ ∀*k*. (*f k*) *integrable-on s* ⟹
∀*k*. ∀*x*∈*s*. (*f k x*)$*1* ≤ (*f* (*Suc k*) *x*)$*1* ⟹ ∀*x*∈*s*. ((λ*k*. *f k x*) −−−> *g x*) *sequentially* ⟹
bounded {*integral s* (*f k*)| *k*. *True*} ⟹ *g integrable-on s* ∧ ((λ*k*. *integral s* (*f k*)) −−−> *integral s g*) *sequentially*
proof− **case** *goal1* **note** *assms*=*this*[*rule-format*]
have ∀*x*∈*s*. ∀*k*. *dest-vec1*(*f k x*) ≤ *dest-vec1*(*g x*) **apply** *safe* **apply**(*rule Lim-component-ge*)
apply(*rule goal1*(*4*)[*rule-format*],*assumption*) **apply**(*rule trivial-limit-sequentially*)
unfolding *eventually-sequentially* **apply**(*rule-tac x*=*k* **in** *exI*)
apply(*rule transitive-stepwise-le*) **using** *goal1*(*3*) **by** *auto* **note** *fg*=*this*[*rule-format*]

have ∃*i*. ((λ*k*. *integral s* (*f k*)) −−−> *i*) *sequentially* **apply**(*rule bounded-increasing-convergent*)
apply(*rule goal1*(*5*)) **apply**(*rule*,*rule integral-component-le*) **apply**(*rule goal1*(*2*)[*rule-format*])+
using *goal1*(*3*) **by** *auto* **then guess** *i* **.. note** *i*=*this*
have ⋀*k*. ∀*x*∈*s*. ∀*n*≥*k*. *f k x* ≤ *f n x* **apply**(*rule*,*rule transitive-stepwise-le*) **using** *goal1*(*3*) **by** *auto*
hence *i*′:∀*k*. (*integral s* (*f k*))$*1* ≤ *i*$*1* **apply**−**apply**(*rule*,*rule Lim-component-ge*)
apply(*rule i*,*rule trivial-limit-sequentially*) **unfolding** *eventually-sequentially*
apply(*rule-tac x*=*k* **in** *exI*,*safe*) **apply**(*rule integral-dest-vec1-le*)
apply(*rule goal1*(*2*)[*rule-format*])+ **by** *auto*

note *int* = *assms*(*2*)[*unfolded integrable-alt*[*of - s*],*THEN conjunct1*,*rule-format*]
have *ifif*:⋀*k t*. (λ*x*. *if x* ∈ *t then if x* ∈ *s then f k x else 0 else 0*) =

```
    (λx. if x ∈ t∩s then f k x else 0) apply(rule ext) by auto
  have int':⋀k a b. f k integrable-on {a..b} ∩ s apply(subst integrable-restrict-univ[THEN
sym])
    apply(subst ifif[THEN sym]) apply(subst integrable-restrict-univ) using int
.
   have ⋀a b. (λx. if x ∈ s then g x else 0) integrable-on {a..b} ∧
    ((λk. integral {a..b} (λx. if x ∈ s then f k x else 0)) ---->
    integral {a..b} (λx. if x ∈ s then g x else 0)) sequentially
   proof(rule monotone-convergence-interval,safe)
    case goal1 show ?case using int .
   next case goal2 thus ?case apply-apply(cases x∈s) using assms(3) by
auto
   next case goal3 thus ?case apply-apply(cases x∈s) using assms(4) by
auto
   next case goal4 note * = integral-dest-vec1-nonneg[unfolded vector-le-def
forall-1 zero-index]
     have ⋀k. norm (integral {a..b} (λx. if x ∈ s then f k x else 0)) ≤ norm
(integral s (f k))
      unfolding norm-real apply(subst abs-of-nonneg) apply(rule *[OF int])
      apply(safe,case-tac x∈s) apply(drule assms(1)) prefer 3
       apply(subst abs-of-nonneg) apply(rule *[OF assms(2) goal1(1)[THEN
spec]])
      apply(subst integral-restrict-univ[THEN sym,OF int])
      unfolding ifif unfolding integral-restrict-univ[OF int']
     apply(rule integral-subset-component-le[OF - int' assms(2)]) using assms(1)
by auto
    thus ?case using assms(5) unfolding bounded-iff apply safe
      apply(rule-tac x=aa in exI,safe) apply(erule-tac x=integral s (f k) in
ballE)
      apply(rule order-trans) apply assumption by auto qed note g = con-
junctD2[OF this]

   have (g has-integral i) s unfolding has-integral-alt' apply safe apply(rule
g(1))
  proof- case goal1 hence e/4>0 by auto
     from i[unfolded Lim-sequentially,rule-format,OF this] guess N .. note
N=this
    note assms(2)[of N,unfolded has-integral-integral has-integral-alt'[of f N]]
     from this[THEN conjunct2,rule-format,OF ‹e/4>0›] guess B .. note
B=conjunctD2[OF this]
    show ?case apply(rule,rule,rule B,safe)
    proof- fix a b::real^'n assume ab:ball 0 B ⊆ {a..b}
     from ‹e>0› have e/2>0 by auto
      from g(2)[unfolded Lim-sequentially,of a b,rule-format,OF this] guess M
.. note M=this
      have **:norm (integral {a..b} (λx. if x ∈ s then f N x else 0) - i) < e/2
     apply(rule norm-triangle-half-l) using B(2)[rule-format,OF ab] N[rule-format,of
N]
       unfolding dist-norm apply-defer apply(subst norm-minus-commute)
```

by *auto*
 have $*$:$\bigwedge$*f1 f2 g. abs(f1 − i$1) < e / 2 ⟶ abs(f2 − g) < e / 2 ⟶ f1 ≤ f2 ⟶ f2 ≤ i$1*
 ⟶ *abs(g − i$1) < e* **by** *arith*
 show *norm (integral {a..b} (λx. if x ∈ s then g x else 0) − i) < e*
 unfolding *norm-real Cart-simps* **apply**(*rule* $*$[*rule-format*])
 apply(*rule* $**$[*unfolded norm-real Cart-simps*])
 apply(*rule M*[*rule-format*,*of M + N*,*unfolded dist-real*]) **apply**(*rule le-add1*)
 apply(*rule integral-component-le*[*OF int int*]) **defer**
 apply(*rule order-trans*[*OF - i'*[*rule-format*,*of M + N*]])
 proof *safe* **case** *goal2* **have** $\bigwedge$*m. x∈s ⟹ ∀ n≥m. (f m x)$1 ≤ (f n x)$1*
 apply(*rule transitive-stepwise-le*) **using** *assms(3)* **by** *auto* **thus** *?case* **by** *auto*
 next case *goal1* **show** *?case* **apply**(*subst integral-restrict-univ*[*THEN sym*,*OF int*])
 unfolding *ifif integral-restrict-univ*[*OF int'*]
 apply(*rule integral-subset-component-le*[*OF - int'*]) **using** *assms* **by** *auto*
 qed qed qed
 thus *?case* **apply** *safe* **defer apply**(*drule integral-unique*) **using** *i* **by** *auto*
qed

 have *sub*:$\bigwedge$*k. integral s (λx. f k x − f 0 x) = integral s (f k) − integral s (f 0)*
 apply(*subst integral-sub*) **apply**(*rule assms(1)*[*rule-format*])+ **by** *rule*
 have $\bigwedge$*x m. x∈s ⟹ ∀ n≥m. dest-vec1 (f m x) ≤ dest-vec1 (f n x)* **apply**(*rule transitive-stepwise-le*)
 using *assms(2)* **by** *auto* **note** $*$ = *this*[*rule-format*]
 have *(λx. g x − f 0 x) integrable-on s ∧((λk. integral s (λx. f (Suc k) x − f 0 x)) −−−>*
 integral s (λx. g x − f 0 x)) sequentially **apply**(*rule lem*,*safe*)
 proof− **case** *goal1* **thus** *?case* **using** $*$[*of x 0 Suc k*] **by** *auto*
 next case *goal2* **thus** *?case* **apply**(*rule integrable-sub*) **using** *assms(1)* **by** *auto*
 next case *goal3* **thus** *?case* **using** $*$[*of x Suc k Suc (Suc k)*] **by** *auto*
 next case *goal4* **thus** *?case* **apply**−**apply**(*rule Lim-sub*)
 using *seq-offset*[*OF assms(3)*[*rule-format*],*of x 1*] **by** *auto*
 next case *goal5* **thus** *?case* **using** *assms(4)* **unfolding** *bounded-iff*
 apply *safe* **apply**(*rule-tac x=a + norm (integral s (λx. f 0 x))* **in** *exI*)
 apply *safe* **apply**(*erule-tac x=integral s (λx. f (Suc k) x)* **in** *ballE*) **unfolding** *sub*
 apply(*rule order-trans*[*OF norm-triangle-ineq4*]) **by** *auto* **qed**
 note *conjunctD2*[*OF this*] **note** *Lim-add*[*OF this*(*2*) *Lim-const*[*of integral s (f 0)*]]
 integrable-add[*OF this*(*1*) *assms*(*1*)[*rule-format*,*of 0*]]
 thus *?thesis* **unfolding** *sub* **apply**−**apply** *rule* **defer apply**(*subst*(*asm*) *integral-sub*)
 using *assms(1)* **apply** *auto* **apply**(*rule seq-offset-rev*[**where** *k=1*]) **by** *auto*
qed

lemma *monotone-convergence-decreasing*: **fixes** *f*::*nat* ⇒ *real^'n* ⇒ *real^1*
 assumes ∀ *k. (f k) integrable-on s* ∀ *k.* ∀ *x∈s. (f (Suc k) x)$1 ≤ (f k x)$1*

$\forall x{\in}s.\ ((\lambda k.\ f\ k\ x)\ ---{>}\ g\ x)$ *sequentially bounded* $\{$*integral* $s\ (f\ k)|\ k.\ True\}$
shows g *integrable-on* $s \wedge ((\lambda k.\ integral\ s\ (f\ k))\ ---{>}\ integral\ s\ g)$ *sequentially*
proof− **note** *assm* = *assms*[*rule-format*]
have $*$:$\{$*integral* $s\ (\lambda x.\ -\ f\ k\ x)\ |k.\ True\} = op\ *_R\ -1\ `\ \{$*integral* $s\ (f\ k)|\ k.\ True\}$
apply *safe* **unfolding** *image-iff* **apply**(*rule-tac x*=*integral s* (*f k*) **in** *bexI*)
prefer *3*
apply(*rule-tac x*=*k* **in** *exI*) **unfolding** *integral-neg*[*OF assm*(*1*)] **by** *auto*
have $(\lambda x.\ -\ g\ x)$ *integrable-on* $s \wedge ((\lambda k.\ integral\ s\ (\lambda x.\ -\ f\ k\ x))$
$---{>}$ *integral* $s\ (\lambda x.\ -\ g\ x))$ *sequentially* **apply**(*rule monotone-convergence-increasing*)
apply(*safe*,*rule integrable-neg*) **apply**(*rule assm*) **defer apply**(*rule Lim-neg*)
apply(*rule assm*,*assumption*) **unfolding** $*$ **apply**(*rule bounded-scaling*) **using** *assm* **by** *auto*
note $*$ = *conjunctD2*[*OF this*]
show *?thesis* **apply** *rule* **using** *integrable-neg*[*OF* $*$(*1*)] **defer**
using *Lim-neg*[*OF* $*$(*2*)] **apply**− **unfolding** *integral-neg*[*OF assm*(*1*)]
unfolding *integral-neg*[*OF* $*$(*1*),*THEN sym*] **by** *auto* **qed**

lemma *monotone-convergence-increasing-real*: **fixes** f::*nat* $\Rightarrow$ *real*^'*n* $\Rightarrow$ *real*
assumes $\forall k.\ (f\ k)$ *integrable-on* $s\ \ \forall k.\ \forall x{\in}s.\ (f\ (Suc\ k)\ x) \geq (f\ k\ x)$
$\forall x{\in}s.\ ((\lambda k.\ f\ k\ x)\ ---{>}\ g\ x)$ *sequentially bounded* $\{$*integral* $s\ (f\ k)|\ k.\ True\}$
shows g *integrable-on* $s \wedge ((\lambda k.\ integral\ s\ (f\ k))\ ---{>}\ integral\ s\ g)$ *sequentially*
proof− **have** $*$:$\{$*integral* $s\ (\lambda x.\ vec1\ (f\ k\ x))\ |k.\ True\} =\ vec1\ `\ \{$*integral* $s\ (f\ k)\ |k.\ True\}$
unfolding *integral-trans*[*OF assms*(*1*)[*rule-format*]] **by** *auto*
have *vec1* $\circ$ g *integrable-on* $s \wedge ((\lambda k.\ integral\ s\ (vec1 \circ f\ k))\ ---{>}\ integral\ s\ (vec1 \circ g))$ *sequentially*
apply(*rule monotone-convergence-increasing*) **unfolding** *o-def integrable-on-trans*
unfolding *vec1-dest-vec1* **apply**(*rule assms*)+ **unfolding** *Lim-trans* **unfolding** $*$ **using** *assms*(*3*,*4*) **by** *auto*
thus *?thesis* **unfolding** *o-def* **unfolding** *integral-trans*[*OF assms*(*1*)[*rule-format*]] **by** *auto* **qed**

lemma *monotone-convergence-decreasing-real*: **fixes** f::*nat* $\Rightarrow$ *real*^'*n* $\Rightarrow$ *real*
assumes $\forall k.\ (f\ k)$ *integrable-on* $s\ \ \forall k.\ \forall x{\in}s.\ (f\ (Suc\ k)\ x) \leq (f\ k\ x)$
$\forall x{\in}s.\ ((\lambda k.\ f\ k\ x)\ ---{>}\ g\ x)$ *sequentially bounded* $\{$*integral* $s\ (f\ k)|\ k.\ True\}$
shows g *integrable-on* $s \wedge ((\lambda k.\ integral\ s\ (f\ k))\ ---{>}\ integral\ s\ g)$ *sequentially*
proof− **have** $*$:$\{$*integral* $s\ (\lambda x.\ vec1\ (f\ k\ x))\ |k.\ True\} =\ vec1\ `\ \{$*integral* $s\ (f\ k)\ |k.\ True\}$
unfolding *integral-trans*[*OF assms*(*1*)[*rule-format*]] **by** *auto*
have *vec1* $\circ$ g *integrable-on* $s \wedge ((\lambda k.\ integral\ s\ (vec1 \circ f\ k))\ ---{>}\ integral\ s\ (vec1 \circ g))$ *sequentially*
apply(*rule monotone-convergence-decreasing*) **unfolding** *o-def integrable-on-trans*
unfolding *vec1-dest-vec1* **apply**(*rule assms*)+ **unfolding** *Lim-trans* **unfolding** $*$ **using** *assms*(*3*,*4*) **by** *auto*
thus *?thesis* **unfolding** *o-def* **unfolding** *integral-trans*[*OF assms*(*1*)[*rule-format*]] **by** *auto* **qed**

23.67 absolute integrability (this is the same as Lebesgue integrability).

definition *absolutely-integrable-on* (**infixr** *absolutely'-integrable'-on 46*) **where**
f absolutely-integrable-on s ⟷ *f integrable-on s* ∧ (λ*x*. (*norm*(*f x*))) *integrable-on s*

lemma *absolutely-integrable-onI*[*intro?*]:
f integrable-on s ⟹ (λ*x*. (*norm*(*f x*))) *integrable-on s* ⟹ *f absolutely-integrable-on s*
unfolding *absolutely-integrable-on-def* **by** *auto*

lemma *absolutely-integrable-onD*[*dest*]: **assumes** *f absolutely-integrable-on s*
shows *f integrable-on s* (λ*x*. (*norm*(*f x*))) *integrable-on s*
using *assms* **unfolding** *absolutely-integrable-on-def* **by** *auto*

lemma *absolutely-integrable-on-trans*[*simp*]: **fixes** *f*::*real^'n* ⇒ *real* **shows**
(*vec1 o f*) *absolutely-integrable-on s* ⟷ *f absolutely-integrable-on s*
unfolding *absolutely-integrable-on-def o-def* **by** *auto*

lemma *integral-norm-bound-integral*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *f integrable-on s g integrable-on s* ∀*x*∈*s*. *norm*(*f x*) ≤ *g x*
shows *norm*(*integral s f*) ≤ (*integral s g*)
proof− **have** ∗:⋀*x y*. (∀*e*::*real*. 0 < *e* ⟶ *x* < *y* + *e*) ⟶ *x* ≤ *y* **apply**(*safe*,*rule ccontr*)
apply(*erule-tac x*=*x* − *y* **in** *allE*) **by** *auto*
have ⋀*e sg dsa dia ig*. *norm*(*sg*) ≤ *dsa* ⟶ *abs*(*dsa* − *dia*) < *e* / 2 ⟶ *norm*(*sg* − *ig*) < *e* / 2
⟶ *norm*(*ig*) < *dia* + *e*
proof *safe* **case** *goal1* **show** *?case* **apply**(*rule le-less-trans*[*OF norm-triangle-sub*[*of ig sg*]])
apply(*subst real-sum-of-halves*[*of e*,*THEN sym*]) **unfolding** *add-assoc*[*symmetric*]
apply(*rule add-le-less-mono*) **defer apply**(*subst norm-minus-commute*,*rule goal1*)
apply(*rule order-trans*[*OF goal1*(*1*)]) **using** *goal1*(*2*) **by** *arith*
qed note *norm*=*this*[*rule-format*]
have *lem*:⋀*f*::*real^'n* ⇒ *'a*. ⋀ *g a b*. *f integrable-on* {*a*..*b*} ⟹ *g integrable-on* {*a*..*b*} ⟹
∀*x*∈{*a*..*b*}. *norm*(*f x*) ≤ (*g x*) ⟹ *norm*(*integral*({*a*..*b*}) *f*) ≤ (*integral*({*a*..*b*}) *g*)
proof(*rule* ∗[*rule-format*]) **case** *goal1* **hence** ∗:*e*/*2*>*0* **by** *auto*
from *integrable-integral*[*OF goal1*(*1*),*unfolded has-integral*[*of f*],*rule-format*,*OF* ∗]
guess *d1* .. **note** *d1* = *conjunctD2*[*OF this*,*rule-format*]
from *integrable-integral*[*OF goal1*(*2*),*unfolded has-integral*[*of g*],*rule-format*,*OF* ∗]
guess *d2* .. **note** *d2* = *conjunctD2*[*OF this*,*rule-format*]
note *gauge-inter*[*OF d1*(*1*) *d2*(*1*)]
from *fine-division-exists*[*OF this*, *of a b*] **guess** *p* . **note** *p*=*this*
show *?case* **apply**(*rule norm*) **defer**

```
    apply(rule d2(2)[OF conjI[OF p(1)],unfolded real-norm-def]) defer
    apply(rule d1(2)[OF conjI[OF p(1)]]) defer apply(rule setsum-norm-le)
  proof safe fix x k assume (x,k)∈p note as = tagged-division-ofD(2-4)[OF
p(1) this]
    from this(3) guess u v apply-by(erule exE)+ note uv=this
   show norm (content k *R f x) ≤ content k *R g x unfolding uv norm-scaleR
     unfolding abs-of-nonneg[OF content-pos-le] real-scaleR-def
     apply(rule mult-left-mono) using goal1(3) as by auto
  qed(insert p[unfolded fine-inter],auto) qed

 { presume ⋀e. 0 < e ⟹ norm (integral s f) < integral s g + e
  thus ?thesis apply-apply(rule *[rule-format]) by auto }
 fix e::real assume e>0 hence e:e/2 > 0 by auto
 note assms(1)[unfolded integrable-alt[of f]] note f=this[THEN conjunct1,rule-format]
 note assms(2)[unfolded integrable-alt[of g]] note g=this[THEN conjunct1,rule-format]
 from integrable-integral[OF assms(1),unfolded has-integral'[of f],rule-format,OF
e]
 guess B1 .. note B1=conjunctD2[OF this[rule-format],rule-format]
 from integrable-integral[OF assms(2),unfolded has-integral'[of g],rule-format,OF
e]
 guess B2 .. note B2=conjunctD2[OF this[rule-format],rule-format]
 from bounded-subset-closed-interval[OF bounded-ball, of 0::real^'n max B1 B2]
 guess a b apply-by(erule exE)+ note ab=this[unfolded ball-max-Un]

 have ball 0 B1 ⊆ {a..b} using ab by auto
 from B1(2)[OF this] guess z .. note z=conjunctD2[OF this]
 have ball 0 B2 ⊆ {a..b} using ab by auto
 from B2(2)[OF this] guess w .. note w=conjunctD2[OF this]

 show norm (integral s f) < integral s g + e apply(rule norm)
  apply(rule lem[OF f g, of a b]) unfolding integral-unique[OF z(1)] integral-unique[OF
w(1)]
   defer apply(rule w(2)[unfolded real-norm-def],rule z(2))
  apply safe apply(case-tac x∈s) unfolding if-P apply(rule assms(3)[rule-format])
by auto qed

lemma integral-norm-bound-integral-component: fixes f::real^'n ⇒ 'a::banach
 assumes f integrable-on s g integrable-on s  ∀x∈s. norm(f x) ≤ (g x)$k
 shows norm(integral s f) ≤ (integral s g)$k
proof- have norm (integral s f) ≤ integral s ((λx. x $ k) o g)
   apply(rule integral-norm-bound-integral[OF assms(1)])
   apply(rule integrable-linear[OF assms(2)],rule)
   unfolding o-def by(rule assms)
 thus ?thesis unfolding o-def integral-component-eq[OF assms(2)] . qed

lemma has-integral-norm-bound-integral-component: fixes f::real^'n ⇒ 'a::banach
 assumes (f has-integral i) s (g has-integral j) s ∀x∈s. norm(f x) ≤ (g x)$k
 shows norm(i) ≤ j$k using integral-norm-bound-integral-component[of f s g k]
 unfolding integral-unique[OF assms(1)] integral-unique[OF assms(2)]
```

using *assms* **by** *auto*

lemma *absolutely-integrable-le*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *f absolutely-integrable-on s*
shows *norm*(*integral s f*) ≤ *integral s* (λ*x*. *norm*(*f x*))
apply(*rule integral-norm-bound-integral*) **using** *assms* **by** *auto*

lemma *absolutely-integrable-0*[*intro*]: (λ*x*. *0*) *absolutely-integrable-on s*
unfolding *absolutely-integrable-on-def* **by** *auto*

lemma *absolutely-integrable-cmul*[*intro*]:
f absolutely-integrable-on s ⟹ (λ*x*. *c* *$_R$ *f x*) *absolutely-integrable-on s*
unfolding *absolutely-integrable-on-def* **using** *integrable-cmul*[*of f s c*]
using *integrable-cmul*[*of* λ*x*. *norm* (*f x*) *s abs c*] **by** *auto*

lemma *absolutely-integrable-neg*[*intro*]:
f absolutely-integrable-on s ⟹ (λ*x*. −*f*(*x*)) *absolutely-integrable-on s*
apply(*drule absolutely-integrable-cmul*[**where** *c*=−*1*]) **by** *auto*

lemma *absolutely-integrable-norm*[*intro*]:
f absolutely-integrable-on s ⟹ (λ*x*. *norm*(*f x*)) *absolutely-integrable-on s*
unfolding *absolutely-integrable-on-def* **by** *auto*

lemma *absolutely-integrable-abs*[*intro*]:
f absolutely-integrable-on s ⟹ (λ*x*. *abs*(*f x*::*real*)) *absolutely-integrable-on s*
apply(*drule absolutely-integrable-norm*) **unfolding** *real-norm-def* **.**

lemma *absolutely-integrable-on-subinterval*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach* **shows**
f absolutely-integrable-on s ⟹ {*a*..*b*} ⊆ *s* ⟹ *f absolutely-integrable-on* {*a*..*b*}
unfolding *absolutely-integrable-on-def* **by**(*meson integrable-on-subinterval*)

lemma *absolutely-integrable-bounded-variation*: **fixes** *f*::*real^'n* ⇒ *'a*::*banach*
assumes *f absolutely-integrable-on UNIV*
obtains *B* **where** ∀ *d*. *d division-of* (⋃ *d*) ⟶ *setsum* (λ*k*. *norm*(*integral k f*))
d ≤ *B*
apply(*rule that*[*of integral UNIV* (λ*x*. *norm* (*f x*))])
proof *safe* **case** *goal1* **note** *d* = *division-ofD*[*OF this*(*2*)]
have (∑ *k*∈*d*. *norm* (*integral k f*)) ≤ (∑ *i*∈*d*. *integral i* (λ*x*. *norm* (*f x*)))
apply(*rule setsum-mono*,*rule absolutely-integrable-le*) **apply**(*drule d*(*4*),*safe*)
apply(*rule absolutely-integrable-on-subinterval*[*OF assms*]) **by** *auto*
also have ... ≤ *integral* (⋃ *d*) (λ*x*. *norm* (*f x*))
apply(*subst integral-combine-division-topdown*[*OF - goal1*(*2*)])
using *integrable-on-subdivision*[*OF goal1*(*2*)] **using** *assms* **by** *auto*
also have ... ≤ *integral UNIV* (λ*x*. *norm* (*f x*))
apply(*rule integral-subset-le*)
using *integrable-on-subdivision*[*OF goal1*(*2*)] **using** *assms* **by** *auto*
finally show *?case* **. qed**

lemma *helplemma*:

assumes *setsum* (λx. *norm*(*f x* − *g x*)) *s* < *e finite s*
shows *abs*(*setsum* (λx. *norm*(*f x*)) *s* − *setsum* (λx. *norm*(*g x*)) *s*) < *e*
unfolding *setsum-subtractf*[*THEN sym*] **apply**(*rule le-less-trans*[*OF setsum-abs*])
apply(*rule le-less-trans*[*OF - assms*(*1*)]) **apply**(*rule setsum-mono*)
using *norm-triangle-ineq3* **.**

lemma *bounded-variation-absolutely-integrable-interval*:
fixes *f*::*real^'n* ⇒ *real^'m* **assumes** *f integrable-on* {*a*..*b*}
∀ *d*. *d division-of* {*a*..*b*} ⟶ *setsum* (λk. *norm*(*integral k f*)) *d* ≤ *B*
shows *f absolutely-integrable-on* {*a*..*b*}
proof− **let** *?S* = (λ*d*. *setsum* (λk. *norm*(*integral k f*)) *d*) ‘ {*d*. *d division-of* {*a*..*b*} } **def** *i* ≡ *Sup ?S*
have *i*:*isLub UNIV ?S i* **unfolding** *i-def* **apply**(*rule Sup*)
apply(*rule elementary-interval*) **defer apply**(*rule-tac x*=*B* **in** *exI*)
apply(*rule setleI*) **using** *assms*(*2*) **by** *auto*
show *?thesis* **apply**(*rule*,*rule assms*) **apply** *rule* **apply**(*subst has-integral*[*of - i*])
proof *safe* **case** *goal1* **hence** *i* − *e* / *2* ∉ *Collect* (*isUb UNIV* (*setsum* (λk. *norm* (*integral k f*)) ‘
{*d*. *d division-of* {*a*..*b*}})) **using** *isLub-ubs*[*OF i*,*rule-format*]
unfolding *setge-def ubs-def* **by** *auto*
hence ∃ *y*. *y division-of* {*a*..*b*} ∧ *i* − *e* / *2* < (∑ *k*∈*y*. *norm* (*integral k f*))
unfolding *mem-Collect-eq isUb-def setle-def* **by** *simp* **then guess** *d* **.. note** *d*=*conjunctD2*[*OF this*]
note *d'* = *division-ofD*[*OF this*(*1*)]

have ∀ *x*. ∃ *e*>*0*. ∀ *i*∈*d*. *x* ∉ *i* ⟶ *ball x e* ∩ *i* = {}
proof case *goal1* **have** ∃ *da*>*0*. ∀ *xa*∈⋃{*i* ∈ *d*. *x* ∉ *i*}. *da* ≤ *dist x xa*
apply(*rule separate-point-closed*) **apply**(*rule closed-Union*)
apply(*rule finite-subset*[*OF - d'*(*1*)]) **apply** *safe* **apply**(*drule d'*(*4*)) **by** *auto*
thus *?case* **apply** *safe* **apply**(*rule-tac x*=*da* **in** *exI*,*safe*)
apply(*erule-tac x*=*xa* **in** *ballE*) **by** *auto*
qed from *choice*[*OF this*] **guess** *k* **.. note** *k*=*conjunctD2*[*OF this*[*rule-format*],*rule-format*]

have *e*/*2* > *0* **using** *goal1* **by** *auto*
from *henstock-lemma*[*OF assms*(*1*) *this*] **guess** *g* **. note** *g*=*this*[*rule-format*]
let *?g* = λx. *g x* ∩ *ball x* (*k x*)
show *?case* **apply**(*rule-tac x*=*?g* **in** *exI*) **apply** *safe*
proof− **show** *gauge ?g* **using** *g*(*1*) **unfolding** *gauge-def* **using** *k*(*1*) **by** *auto*
fix *p* **assume** *p tagged-division-of* {*a*..*b*} *?g fine p*
note *p* = *this*(*1*) *conjunctD2*[*OF this*(*2*)[*unfolded fine-inter*]]
note *p'* = *tagged-division-ofD*[*OF p*(*1*)]
def *p'* ≡ {(*x*,*k*) | *x k*. ∃ *i l*. *x* ∈ *i* ∧ *i* ∈ *d* ∧ (*x*,*l*) ∈ *p* ∧ *k* = *i* ∩ *l*}
have *gp'*:*g fine p'* **using** *p*(*2*) **unfolding** *p'-def fine-def* **by** *auto*
have *p''*:*p' tagged-division-of* {*a*..*b*} **apply**(*rule tagged-division-ofI*)
proof− **show** *finite p'* **apply**(*rule finite-subset*[*of* - (λ(*k*,(*x*,*l*)). (*x*,*k* ∩ *l*))
‘ {(*k*,*xl*) | *k xl*. *k* ∈ *d* ∧ *xl* ∈ *p*}]) **unfolding** *p'-def*
defer apply(*rule finite-imageI*,*rule finite-product-dependent*[*OF d'*(*1*) *p'*(*1*)])

apply *safe* **unfolding** *image-iff* **apply**(*rule-tac x=(i,x,l)* **in** *bexI*) **by** *auto*
fix *x k* **assume** $(x,k)\in p'$
hence $\exists\, i\ l.\ x \in i \wedge i \in d \wedge (x,\ l) \in p \wedge k = i \cap l$ **unfolding** *p'-def* **by** *auto*
then guess *i l* **apply**−**by**(*erule exE*)+ **note** *il=conjunctD4*[*OF this*]
show $x\in k\ k\subseteq\{a..b\}$ **using** *p'(2−3)*[*OF il(3)*] *il* **by** *auto*
show $\exists\, a\ b.\ k = \{a..b\}$ **unfolding** *il* **using** *p'(4)*[*OF il(3)*] *d'(4)*[*OF il(2)*]
apply *safe* **unfolding** *inter-interval* **by** *auto*
next fix *x1 k1* **assume** $(x1,k1)\in p'$
hence $\exists\, i\ l.\ x1 \in i \wedge i \in d \wedge (x1,\ l) \in p \wedge k1 = i \cap l$ **unfolding** *p'-def* **by** *auto*
then guess *i1 l1* **apply**−**by**(*erule exE*)+ **note** *il1=conjunctD4*[*OF this*]
fix *x2 k2* **assume** $(x2,k2)\in p'$
hence $\exists\, i\ l.\ x2 \in i \wedge i \in d \wedge (x2,\ l) \in p \wedge k2 = i \cap l$ **unfolding** *p'-def* **by** *auto*
then guess *i2 l2* **apply**−**by**(*erule exE*)+ **note** *il2=conjunctD4*[*OF this*]
assume $(x1,\ k1) \neq (x2,\ k2)$
hence $interior(i1) \cap interior(i2) = \{\} \vee interior(l1) \cap interior(l2) = \{\}$
using *d'(5)*[*OF il1(2) il2(2)*] *p'(5)*[*OF il1(3) il2(3)*] **unfolding** *il1 il2* **by** *auto*
thus $interior\ k1 \cap interior\ k2 = \{\}$ **unfolding** *il1 il2* **by** *auto*
next have $*:\forall (x,\ X) \in p'.\ X \subseteq \{a..b\}$ **unfolding** *p'-def* **using** *d'* **by** *auto*
show $\bigcup\{k.\ \exists x.\ (x,\ k) \in p'\} = \{a..b\}$ **apply** *rule* **apply**(*rule Union-least*)
unfolding *mem-Collect-eq* **apply**(*erule exE*) **apply**(*drule* $*$[*rule-format*]) **apply** *safe*
proof− **fix** *y* **assume** $y{:}y\in\{a..b\}$
hence $\exists x\ l.\ (x,\ l) \in p \wedge y\in l$ **unfolding** *p'(6)*[*THEN sym*] **by** *auto*
then guess *x l* **apply**−**by**(*erule exE*)+ **note** *xl=conjunctD2*[*OF this*]
hence $\exists k.\ k\in d \wedge y\in k$ **using** *y* **unfolding** *d'(6)*[*THEN sym*] **by** *auto*
then guess *i* **.. note** $i = conjunctD2[OF\ this]$
have $x\in i$ **using** *fineD*[*OF p(3) xl(1)*] **using** *k(2)*[*OF i(1), of x*] **using** *i(2) xl(2)* **by** *auto*
thus $y\in\bigcup\{k.\ \exists x.\ (x,\ k) \in p'\}$ **unfolding** *p'-def Union-iff* **apply**(*rule-tac x=i ∩ l* **in** *bexI*)
defer unfolding *mem-Collect-eq* **apply**(*rule-tac x=x* **in** *exI*)+ **apply**(*rule-tac x=i∩l* **in** *exI*)
apply *safe* **apply**(*rule-tac x=i* **in** *exI*) **apply**(*rule-tac x=l* **in** *exI*) **using** *i xl* **by** *auto*
qed qed

hence $(\sum (x,\ k){\in}p'.\ norm\ (content\ k *_R f\ x - integral\ k\ f)) < e\ /\ 2$
apply−**apply**(*rule g(2)*[*rule-format*]) **unfolding** *tagged-division-of-def* **apply** *safe* **using** *gp'* **.**
hence $**$: $|(\sum (x,k){\in}p'.\ norm\ (content\ k *_R f\ x)) - (\sum (x,k){\in}p'.\ norm\ (integral\ k\ f))| < e\ /\ 2$
unfolding *split-def* **apply**(*rule helplemma*) **using** *p''* **by** *auto*

have $p'alt{:}p' = \{(x,(i \cap l)) \mid x\ i\ l.\ (x,l) \in p \wedge i \in d \wedge {\sim}(i \cap l = \{})\}$
proof *safe* **case** *goal2*

have $x{\in}i$ **using** *fineD*[*OF p*(*3*) *goal2*(*1*)] *k*(*2*)[*OF goal2*(*2*), *of x*] *goal2*(*4*−)
by *auto*
hence $(x,\ i \cap l) \in p'$ **unfolding** *p′-def* **apply** *safe*
apply(*rule-tac x*=*x* **in** *exI*,*rule-tac x*=*i*∩*l* **in** *exI*) **apply** *safe* **using** *goal2*
by *auto*
thus *?case* **using** *goal2*(*3*) **by** *auto*
next fix *x k* **assume** (*x*,*k*)∈*p′*
hence $\exists\, i\ l.\ x \in i \wedge i \in d \wedge (x,\ l) \in p \wedge k = i \cap l$ **unfolding** *p′-def* **by**
auto
then guess *i l* **apply**−**by**(*erule exE*)+ **note** *il*=*conjunctD4*[*OF this*]
thus $\exists\, y\ i\ l.\ (x,\ k) = (y,\ i \cap l) \wedge (y,\ l) \in p \wedge i \in d \wedge i \cap l \neq \{\}$
apply(*rule-tac x*=*x* **in** *exI*,*rule-tac x*=*i* **in** *exI*,*rule-tac x*=*l* **in** *exI*)
using *p′*(*2*)[*OF il*(*3*)] **by** *auto*
qed
have *sum-p′*:$(\sum (x,\ k){\in}p'.\ norm\ (integral\ k\ f)) = (\sum k{\in}snd\ `\ p'.\ norm$
(*integral k f*))
apply(*subst setsum-over-tagged-division-lemma*[*OF p″*,*of* $\lambda k.\ norm\ (integral$
k f)])
unfolding *norm-eq-zero* **apply**(*rule integral-null*,*assumption*) **..**
note *snd-p* = *division-ofD*[*OF division-of-tagged-division*[*OF p*(*1*)]]

have ∗:$\bigwedge sni\ sni'\ sf\ sf'.\ abs(sf' - sni') < e\ /\ 2 \longrightarrow i - e\ /\ 2 < sni \wedge sni'$
$\leq i \wedge$
$sni \leq sni' \wedge sf' = sf \longrightarrow abs(sf - i) < e$ **by** *arith*
show $norm\ ((\sum (x,\ k){\in}p.\ content\ k *_R norm\ (f\ x)) - i) < e$
unfolding *real-norm-def* **apply**(*rule* ∗[*rule-format*,*OF* ∗∗],*safe*) **apply**(*rule*
d(*2*))
proof− **case** *goal1* **show** *?case* **unfolding** *sum-p′*
apply(*rule isLubD2*[*OF i*]) **using** *division-of-tagged-division*[*OF p″*] **by**
auto
next case *goal2* **have** ∗:$\{k \cap l \mid k\ l.\ k \in d \wedge l \in snd\ `\ p\} =$
$(\lambda(k,l).\ k \cap l)\ `\ \{(k,l) | k\ l.\ k \in d \wedge l \in snd\ `\ p\}$ **by** *auto*
have $(\sum k{\in}d.\ norm\ (integral\ k\ f)) \leq (\sum i{\in}d.\ \sum l{\in}snd\ `\ p.\ norm\ (integral$
$(i \cap l)\ f))$
proof(*rule setsum-mono*) **case** *goal1* **note** *k*=*this*
from *d′*(*4*)[*OF this*] **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
def $d' \equiv \{\{u..v\} \cap l \mid l.\ l \in snd\ `\ p \wedge\ \sim(\{u..v\} \cap l = \{\})\}$ **note** *uvab* =
d′(*2*)[*OF k*[*unfolded uv*]]
have *d′ division-of* {*u*..*v*} **apply**(*subst d′-def*) **apply**(*rule division-inter-1*)

apply(*rule division-of-tagged-division*[*OF p*(*1*)]) **using** *uvab* **.**
hence *norm* (*integral k f*) ≤ *setsum* (λ*k*. *norm* (*integral k f*)) *d′*
unfolding *uv* **apply**(*subst integral-combine-division-topdown*[*of* - - *d′*])
apply(*rule integrable-on-subinterval*[*OF assms*(*1*) *uvab*]) **apply** *assumption*
apply(*rule setsum-norm-le*) **by** *auto*
also have ... = $(\sum k{\in}\{k \cap l \mid l.\ l \in snd\ `\ p\}.\ norm\ (integral\ k\ f))$
apply(*rule setsum-mono-zero-left*) **apply**(*subst simple-image*)
apply(*rule finite-imageI*)+ **apply** *fact* **unfolding** *d′-def uv* **apply** *blast*
proof case *goal1* **hence** $i \in \{\{u..v\} \cap l \mid l.\ l \in snd\ `\ p\}$ **by** *auto*

from *this*[*unfolded mem-Collect-eq*] **guess** *l* **.. note** *l*=*this*
hence {*u*..*v*} ∩ *l* = {} **using** *goal1* **by** *auto* **thus** *?case* **using** *l* **by** *auto*
qed also have ... = (∑ *l*∈*snd* ' *p*. *norm* (*integral* (*k* ∩ *l*) *f*)) **unfolding** *simple-image*
apply(*rule setsum-reindex-nonzero*[*unfolded o-def*])**apply**(*rule finite-imageI*,*rule p'*)
proof− **case** *goal1* **have** *interior* (*k* ∩ *l*) ⊆ *interior* (*l* ∩ *y*) **apply**(*subst*(*2*) *interior-inter*)
apply(*rule Int-greatest*) **defer apply**(*subst goal1*(*4*)) **by** *auto*
hence ∗:*interior* (*k* ∩ *l*) = {} **using** *snd-p*(*5*)[*OF goal1*(*1*−*3*)] **by** *auto*
from *d'*(*4*)[*OF k*] *snd-p*(*4*)[*OF goal1*(*1*)] **guess** *u1 v1 u2 v2* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
show *?case* **using** ∗ **unfolding** *uv inter-interval content-eq-0-interior*[*THEN sym*] **by** *auto*
qed finally show *?case* **.**
qed also have ... = (∑ (*i*,*l*)∈{(*i*, *l*) |*i l*. *i* ∈ *d* ∧ *l* ∈ *snd* ' *p*}. *norm* (*integral* (*i*∩*l*) *f*))
apply(*subst sum-sum-product*[*THEN sym*],*fact*) **using** *p'*(*1*) **by** *auto*
also have ... = (∑ *x*∈{(*i*, *l*) |*i l*. *i* ∈ *d* ∧ *l* ∈ *snd* ' *p*}. *norm* (*integral* (*split op* ∩ *x*) *f*))
unfolding *split-def* **..**
also have ... = (∑ *k*∈{*i* ∩ *l* |*i l*. *i* ∈ *d* ∧ *l* ∈ *snd* ' *p*}. *norm* (*integral k f*))
unfolding ∗ **apply**(*rule setsum-reindex-nonzero*[*THEN sym*,*unfolded o-def*])
apply(*rule finite-product-dependent*) **apply**(*fact*,*rule finite-imageI*,*rule p'*)
unfolding *split-paired-all mem-Collect-eq split-conv o-def*
proof− **note** ∗ = *division-ofD*(*4*,*5*)[*OF division-of-tagged-division*,*OF p*(*1*)]
fix *l1 l2 k1 k2* **assume** *as*:(*l1*, *k1*) ≠ (*l2*, *k2*) *l1* ∩ *k1* = *l2* ∩ *k2*
∃ *i l*. (*l1*, *k1*) = (*i*, *l*) ∧ *i* ∈ *d* ∧ *l* ∈ *snd* ' *p*
∃ *i l*. (*l2*, *k2*) = (*i*, *l*) ∧ *i* ∈ *d* ∧ *l* ∈ *snd* ' *p*
hence *l1* ∈ *d k1* ∈ *snd* ' *p* **by** *auto* **from** *d'*(*4*)[*OF this*(*1*)] ∗(*1*)[*OF this*(*2*)]
guess *u1 v1 u2 v2* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
have *l1* ≠ *l2* ∨ *k1* ≠ *k2* **using** *as* **by** *auto*
hence (*interior*(*k1*) ∩ *interior*(*k2*) = {} ∨ *interior*(*l1*) ∩ *interior*(*l2*) = {}) **apply**−
apply(*erule disjE*) **apply**(*rule disjI2*) **apply**(*rule d'*(*5*)) **prefer** *4* **apply**(*rule disjI1*)
apply(*rule* ∗) **using** *as* **by** *auto*
moreover have *interior*(*l1* ∩ *k1*) = *interior*(*l2* ∩ *k2*) **using** *as*(*2*) **by** *auto*
ultimately have *interior*(*l1* ∩ *k1*) = {} **by** *auto*
thus *norm* (*integral* (*l1* ∩ *k1*) *f*) = *0* **unfolding** *uv inter-interval*
unfolding *content-eq-0-interior*[*THEN sym*] **by** *auto*
qed also have ... = (∑ (*x*, *k*)∈*p'*. *norm* (*integral k f*)) **unfolding** *sum-p'*
apply(*rule setsum-mono-zero-right*) **apply**(*subst* ∗)
apply(*rule finite-imageI*[*OF finite-product-dependent*]) **apply** *fact*
apply(*rule finite-imageI*[*OF p'*(*1*)]) **apply** *safe*
proof− **case** *goal2* **have** *ia* ∩ *b* = {} **using** *goal2* **unfolding** *p'alt image-iff*

Bex-def not-ex
apply(*erule-tac x*=(*a*,*ia*∩*b*) **in** *allE*) **by** *auto* **thus** *?case* **by** *auto*
next case *goal1* **thus** *?case* **unfolding** *p′-def* **apply** *safe*
apply(*rule-tac x*=*i* **in** *exI*,*rule-tac x*=*l* **in** *exI*) **unfolding** *snd-conv image-iff*
apply *safe* **apply**(*rule-tac x*=(*a*,*l*) **in** *bexI*) **by** *auto*
qed finally show *?case* **.**

next case *goal3*
let *?S* = {(*x*, *i* ∩ *l*) |*x i l*. (*x*, *l*) ∈ *p* ∧ *i* ∈ *d*}
have *Sigma-alt*:⋀*s t*. *s* × *t* = {(*i*, *j*) |*i j*. *i* ∈ *s* ∧ *j* ∈ *t*} **by** *auto*
have ∗:*?S* = (λ(*xl*,*i*). (*fst xl*, *snd xl* ∩ *i*)) ‘ (*p* × *d*)
apply *safe* **unfolding** *image-iff* **apply**(*rule-tac x*=((*x*,*l*),*i*) **in** *bexI*) **by** *auto*
note *pdfin* = *finite-cartesian-product*[*OF p′*(*1*) *d′*(*1*)]
have (∑ (*x*, *k*)∈*p′*. *norm* (*content k* ∗*R f x*)) = (∑ (*x*, *k*)∈*?S*. |*content k*| ∗ *norm* (*f x*))
unfolding *norm-scaleR* **apply**(*rule setsum-mono-zero-left*)
apply(*subst* ∗, *rule finite-imageI*) **apply** *fact* **unfolding** *p′alt* **apply** *blast*
apply *safe* **apply**(*rule-tac x*=*x* **in** *exI*,*rule-tac x*=*i* **in** *exI*,*rule-tac x*=*l* **in** *exI*) **by** *auto*
also have ... = (∑ ((*x*,*l*),*i*)∈*p* × *d*. |*content* (*l* ∩ *i*)| ∗ *norm* (*f x*)) **unfolding** ∗
apply(*subst setsum-reindex-nonzero*,*fact*) **unfolding** *split-paired-all*
unfolding *o-def split-def snd-conv fst-conv mem-Sigma-iff Pair-eq* **apply**(*erule-tac conjE*)+
proof− **fix** *x1 l1 k1 x2 l2 k2* **assume** *as*:(*x1*,*l1*)∈*p* (*x2*,*l2*)∈*p k1*∈*d k2*∈*d*
x1 = *x2 l1* ∩ *k1* = *l2* ∩ *k2* ¬ ((*x1* = *x2* ∧ *l1* = *l2*) ∧ *k1* = *k2*)
from *d′*(*4*)[*OF as*(*3*)] *p′*(*4*)[*OF as*(*1*)] **guess** *u1 v1 u2 v2* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
from *as* **have** *l1* ≠ *l2* ∨ *k1* ≠ *k2* **by** *auto*
hence (*interior*(*k1*) ∩ *interior*(*k2*) = {} ∨ *interior*(*l1*) ∩ *interior*(*l2*) = {})
apply−**apply**(*erule disjE*) **apply**(*rule disjI2*) **defer apply**(*rule disjI1*)
apply(*rule d′*(*5*)[*OF as*(*3*−*4*)],*assumption*) **apply**(*rule p′*(*5*)[*OF as*(*1*−*2*)]) **by** *auto*
moreover have *interior*(*l1* ∩ *k1*) = *interior*(*l2* ∩ *k2*) **unfolding** *as* **..**
ultimately have *interior* (*l1* ∩ *k1*) = {} **by** *auto*
thus |*content* (*l1* ∩ *k1*)| ∗ *norm* (*f x1*) = *0* **unfolding** *uv inter-interval*
unfolding *content-eq-0-interior*[*THEN sym*] **by** *auto*
qed *safe* **also have** ... = (∑ (*x*, *k*)∈*p*. *content k* ∗*R norm* (*f x*)) **unfolding** *Sigma-alt*
apply(*subst sum-sum-product*[*THEN sym*]) **apply**(*rule p′*, *rule*,*rule d′*)
apply(*rule setsum-cong2*) **unfolding** *split-paired-all split-conv*
proof− **fix** *x l* **assume** *as*:(*x*,*l*)∈*p*
note *xl* = *p′*(*2*−*4*)[*OF this*] **from** *this*(*3*) **guess** *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
have (∑ *i*∈*d*. |*content* (*l* ∩ *i*)|) = (∑ *k*∈*d*. *content* (*k* ∩ {*u*..*v*}))
apply(*rule setsum-cong2*) **apply**(*drule d′*(*4*),*safe*) **apply**(*subst Int-commute*)

unfolding *inter-interval uv* **apply**(*subst abs-of-nonneg*) **by** *auto*
also have ... = *setsum content* {*k*∩{*u..v*}| *k*. *k*∈*d*} **unfolding** *simple-image*
apply(*rule setsum-reindex-nonzero*[*unfolded o-def*,*THEN sym*]) **apply**(*rule d′*)
proof− **case** *goal1* **from** *d′*(*4*)[*OF this*(*1*)] *d′*(*4*)[*OF this*(*2*)]
guess *u1 v1 u2 v2* **apply**− **by**(*erule exE*)+ **note** *uv*=*this*
have {} = *interior* ((*k* ∩ *y*) ∩ {*u..v*}) **apply**(*subst interior-inter*)
using *d′*(*5*)[*OF goal1*(*1*−*3*)] **by** *auto*
also have ... = *interior* (*y* ∩ (*k* ∩ {*u..v*})) **by** *auto*
also have ... = *interior* (*k* ∩ {*u..v*}) **unfolding** *goal1*(*4*) **by** *auto*
finally show *?case* **unfolding** *uv inter-interval content-eq-0-interior* **..**
qed also have ... = *setsum content* {{*u..v*} ∩ *k* |*k*. *k* ∈ *d* ∧ ~({*u..v*} ∩ *k* = {})}
apply(*rule setsum-mono-zero-right*) **unfolding** *simple-image*
apply(*rule finite-imageI*,*rule d′*) **apply** *blast* **apply** *safe*
apply(*rule-tac x*=*k* **in** *exI*)
proof− **case** *goal1* **from** *d′*(*4*)[*OF this*(*1*)] **guess** *a b* **apply**−**by**(*erule exE*)+ **note** *ab*=*this*
have *interior* (*k* ∩ {*u..v*}) ≠ {} **using** *goal1*(*2*)
unfolding *ab inter-interval content-eq-0-interior* **by** *auto*
thus *?case* **using** *goal1*(*1*) **using** *interior-subset*[*of k* ∩ {*u..v*}] **by** *auto*
qed finally show (∑ *i*∈*d*. |*content* (*l* ∩ *i*)| ∗ *norm* (*f x*)) = *content l* ∗$_R$ *norm* (*f x*)
unfolding *setsum-left-distrib*[*THEN sym*] *real-scaleR-def* **apply** −
apply(*subst*(*asm*) *additive-content-division*[*OF division-inter-1*[*OF d*(*1*)]])
using *xl*(*2*)[*unfolded uv*] **unfolding** *uv* **by** *auto*
qed finally show *?case* **.**
qed qed qed qed

lemma *bounded-variation-absolutely-integrable*: **fixes** *f*::*real^′n* ⇒ *real^′m*
assumes *f integrable-on UNIV* ∀ *d*. *d division-of* (⋃ *d*) ⟶ *setsum* (λ*k*. *norm*(*integral k f*)) *d* ≤ *B*
shows *f absolutely-integrable-on UNIV*
proof(*rule absolutely-integrable-onI*,*fact*,*rule*)
let *?S* = (λ*d*. *setsum* (λ*k*. *norm*(*integral k f*)) *d*) ‘ {*d*. *d division-of* (⋃ *d*)} **def** *i* ≡ *Sup ?S*
have *i*:*isLub UNIV ?S i* **unfolding** *i-def* **apply**(*rule Sup*)
apply(*rule elementary-interval*) **defer apply**(*rule-tac x*=*B* **in** *exI*)
apply(*rule setleI*) **using** *assms*(*2*) **by** *auto*
have *f-int*:⋀*a b*. *f absolutely-integrable-on* {*a..b*}
apply(*rule bounded-variation-absolutely-integrable-interval*[**where** *B*=*B*])
apply(*rule integrable-on-subinterval*[*OF assms*(*1*)]) **defer apply** *safe*
apply(*rule assms*(*2*)[*rule-format*]) **by** *auto*
show ((λ*x*. *norm* (*f x*)) *has-integral i*) *UNIV* **apply**(*subst has-integral-alt′*,*safe*)
proof− **case** *goal1* **show** *?case* **using** *f-int*[*of a b*] **by** *auto*
next case *goal2* **have** ∃ *y*∈*setsum* (λ*k*. *norm* (*integral k f*)) ‘ {*d*. *d division-of* ⋃ *d*}. ¬ *y* ≤ *i* − *e*
proof(*rule ccontr*) **case** *goal1* **hence** *i* ≤ *i* − *e* **apply**−
apply(*rule isLub-le-isUb*[*OF i*]) **apply**(*rule isUbI*) **unfolding** *setle-def* **by**

auto
thus *False* **using** *goal2* **by** *auto*
qed then guess K **.. note** $* = this[unfolded\ image\text{-}iff\ not\text{-}le]$
from *this(1)* **guess** *d* **.. note** *this[unfolded mem-Collect-eq]*
note $d = this(1)\ *(2)[unfolded\ this(2)]$ **note** $d'=division\text{-}ofD[OF\ this(1)]$
have *bounded* $(\bigcup d)$ **by**(*rule elementary-bounded*,*fact*)
from *this[unfolded bounded-pos]* **guess** K **.. note** $K=conjunctD2[OF\ this]$
show *?case* **apply**(*rule-tac* $x=K + 1$ **in** *exI*,*safe*)
proof− **fix** *a b* **assume** $ab:ball\ 0\ (K + 1) \subseteq \{a..b::real\hat{}'n\}$
have $*:\forall s\ s1.\ i - e < s1 \wedge s1 \leq s \wedge s < i + e \longrightarrow abs(s - i) < (e::real)$
by *arith*
show $norm\ (integral\ \{a..b\}\ (\lambda x.\ if\ x \in UNIV\ then\ norm\ (f\ x)\ else\ 0) - i) < e$
unfolding *real-norm-def* **apply**(*rule* *[*rule-format*],*safe*) **apply**(*rule d(2)*)
proof− **case** *goal1* **have** $(\sum k{\in}d.\ norm\ (integral\ k\ f)) \leq setsum\ (\lambda k.\ integral\ k\ (\lambda x.\ norm\ (f\ x)))\ d$
apply(*rule setsum-mono*) **apply**(*rule absolutely-integrable-le*)
apply(*drule d'(4)*,*safe*) **by**(*rule f-int*)
also have ... = $integral\ (\bigcup d)\ (\lambda x.\ norm(f\ x))$
apply(*rule integral-combine-division-bottomup*[*THEN sym*])
apply(*rule d*) **unfolding** *forall-in-division*[*OF d(1)*] **using** *f-int* **by** *auto*
also have ... $\leq$ $integral\ \{a..b\}\ (\lambda x.\ if\ x \in UNIV\ then\ norm\ (f\ x)\ else\ 0)$
proof− **case** *goal1* **have** $\bigcup d \subseteq \{a..b\}$ **apply** *rule* **apply**(*drule K(2)*[*rule-format*])

apply(*rule ab*[*unfolded subset-eq*,*rule-format*]) **by**(*auto simp add*:*dist-norm*)
thus *?case* **apply**− **apply**(*subst if-P*,*rule*) **apply**(*rule integral-subset-le*)
defer
apply(*rule integrable-on-subdivision*[*of - - -* $\{a..b\}$])
apply(*rule d*) **using** *f-int*[*of a b*] **by** *auto*
qed finally show *?case* **.**

next note $f = absolutely\text{-}integrable\text{-}onD[OF\ f\text{-}int[of\ a\ b]]$
note $* = this(2)[unfolded\ has\text{-}integral\text{-}integral\ has\text{-}integral[of\ \lambda x.\ norm\ (f\ x)],rule\text{-}format]$
have $e/2{>}0$ **using** ‹$e{>}0$› **by** *auto* **from** $*[OF\ this]$ **guess** *d1* **.. note** $d1=conjunctD2[OF\ this]$
from $henstock\text{-}lemma[OF\ f(1)\ ‹e/2{>}0›]$ **guess** *d2* **. note** $d2=this$
from $fine\text{-}division\text{-}exists[OF\ gauge\text{-}inter[OF\ d1(1)\ d2(1)],\ of\ a\ b]$ **guess** *p*
.
note $p=this(1)\ conjunctD2[OF\ this(2)[unfolded\ fine\text{-}inter]]$
have $*:\bigwedge sf\ sf'\ si\ di.\ sf' = sf \longrightarrow si \leq i \longrightarrow abs(sf - si) < e\ /\ 2 \longrightarrow abs(sf' - di) < e\ /\ 2 \longrightarrow di < i + e$ **by** *arith*
show $integral\ \{a..b\}\ (\lambda x.\ if\ x \in UNIV\ then\ norm\ (f\ x)\ else\ 0) < i + e$ **apply**(*subst if-P*,*rule*)
proof(*rule* *[*rule-format*])
show $|(\sum (x,k){\in}p.\ norm\ (content\ k *_R f\ x)) - (\sum (x,k){\in}p.\ norm\ (integral\ k\ f))| < e\ /\ 2$
unfolding *split-def* **apply**(*rule helplemma*) **using** *d2(2)*[*rule-format*,*of p*]

using *p(1,3)* **unfolding** *tagged-division-of-def split-def* **by** *auto*
show *abs* $((\sum (x,\ k){\in}p.\ content\ k *_R norm\ (f\ x)) - integral\ \{a..b\}\ (\lambda x.\ norm(f\ x))) < e\ /\ 2$
using *d1(2)*[*rule-format*,*OF conjI*[*OF p(1,2)*]] **unfolding** *real-norm-def*
.
show $(\sum (x,\ k){\in}p.\ content\ k *_R norm\ (f\ x)) = (\sum (x,\ k){\in}p.\ norm\ (content\ k *_R f\ x))$
apply(*rule setsum-cong2*) **unfolding** *split-paired-all split-conv*
apply(*drule tagged-division-ofD(4)*[*OF p(1)*]) **unfolding** *norm-scaleR*
apply(*subst abs-of-nonneg*) **by** *auto*
show $(\sum (x,\ k){\in}p.\ norm\ (integral\ k\ f)) \le i$
apply(*subst setsum-over-tagged-division-lemma*[*OF p(1)*]) **defer apply**(*rule isLubD2*[*OF i*])
unfolding *image-iff* **apply**(*rule-tac x=snd ' p* **in** *bexI*) **unfolding** *mem-Collect-eq* **defer**
apply(*rule partial-division-of-tagged-division*[*of - {a..b}*])
using *p(1)* **unfolding** *tagged-division-of-def* **by** *auto*
qed qed qed(*insert K*,*auto*) **qed qed**

lemma *absolutely-integrable-restrict-univ*:
$(\lambda x.\ if\ x \in s\ then\ f\ x\ else\ (0::'a::banach))\ absolutely\text{-}integrable\text{-}on\ UNIV \longleftrightarrow f\ absolutely\text{-}integrable\text{-}on\ s$
unfolding *absolutely-integrable-on-def if-distrib norm-zero integrable-restrict-univ*
..

lemma *absolutely-integrable-add*[*intro*]: **fixes** *f g*::*real^'n* ⇒ *real^'m*
assumes *f absolutely-integrable-on s g absolutely-integrable-on s*
shows $(\lambda x.\ f(x) + g(x))\ absolutely\text{-}integrable\text{-}on\ s$
proof− **let** *?P* = $\bigwedge f\ g$::*real^'n* ⇒ *real^'m*. *f absolutely-integrable-on UNIV* ⟹ *g absolutely-integrable-on UNIV* ⟹ $(\lambda x.\ f(x) + g(x))\ absolutely\text{-}integrable\text{-}on\ UNIV$
{ presume *as*:*PROP ?P* **note** *a = absolutely-integrable-restrict-univ*[*THEN sym*]
have ∗:$\bigwedge x.\ (if\ x \in s\ then\ f\ x\ else\ 0) + (if\ x \in s\ then\ g\ x\ else\ 0) = (if\ x \in s\ then\ f\ x + g\ x\ else\ 0)$ **by** *auto*
show *?thesis* **apply**(*subst a*) **using** *as*[*OF assms*[*unfolded a*[*of f*] *a*[*of g*]]] **unfolding** ∗ **. }**
fix *f g*::*real^'n* ⇒ *real^'m* **assume** *assms*:*f absolutely-integrable-on UNIV g absolutely-integrable-on UNIV*
note *absolutely-integrable-bounded-variation*
from *this*[*OF assms(1)*] *this*[*OF assms(2)*] **guess** *B1 B2* **. note** *B*=*this*[*rule-format*]
show $(\lambda x.\ f(x) + g(x))\ absolutely\text{-}integrable\text{-}on\ UNIV$
apply(*rule bounded-variation-absolutely-integrable*[*of - B1+B2*])
apply(*rule integrable-add*) **prefer** *3*
proof *safe* **case** *goal1* **have** $\bigwedge k.\ k \in d \Longrightarrow f\ integrable\text{-}on\ k \wedge g\ integrable\text{-}on\ k$
apply(*drule division-ofD(4)*[*OF goal1*]) **apply** *safe*
apply(*rule-tac*[!] *integrable-on-subinterval*[*of - UNIV*]) **using** *assms* **by** *auto*
hence $(\sum k{\in}d.\ norm\ (integral\ k\ (\lambda x.\ f\ x + g\ x))) \le (\sum k{\in}d.\ norm\ (integral\ k\ f)) + (\sum k{\in}d.\ norm\ (integral\ k\ g))$ **apply**−

unfolding *setsum-addf*[*THEN sym*] **apply**(*rule setsum-mono*)
apply(*subst integral-add*) **prefer** *3* **apply**(*rule norm-triangle-ineq*) **by** *auto*
also have ... ≤ *B1* + *B2* **using** *B*(*1*)[*OF goal1*] *B*(*2*)[*OF goal1*] **by** *auto*
finally show *?case* **.**
qed(*insert assms*,*auto*) **qed**

lemma *absolutely-integrable-sub*[*intro*]: **fixes** *f g*::*real^'n* ⇒ *real^'m*
assumes *f absolutely-integrable-on s g absolutely-integrable-on s*
shows (λ*x*. *f*(*x*) − *g*(*x*)) *absolutely-integrable-on s*
using *absolutely-integrable-add*[*OF assms*(*1*) *absolutely-integrable-neg*[*OF assms*(*2*)]]
unfolding *algebra-simps* **.**

lemma *absolutely-integrable-linear*: **fixes** *f*::*real^'m* ⇒ *real^'n* **and** *h*::*real^'n* ⇒ *real^'p*
assumes *f absolutely-integrable-on s bounded-linear h*
shows (*h o f*) *absolutely-integrable-on s*
proof− **{ presume** *as*:⋀*f*::*real^'m* ⇒ *real^'n*. ⋀*h*::*real^'n* ⇒ *real^'p*.
f absolutely-integrable-on UNIV ⟹ *bounded-linear h* ⟹
(*h o f*) *absolutely-integrable-on UNIV* **note** *a* = *absolutely-integrable-restrict-univ*[*THEN sym*]
show *?thesis* **apply**(*subst a*) **using** *as*[*OF assms*[*unfolded a*[*of f*] *a*[*of g*]]]
unfolding *o-def if-distrib linear-simps*[*OF assms*(*2*)] **. }**
fix *f*::*real^'m* ⇒ *real^'n* **and** *h*::*real^'n* ⇒ *real^'p*
assume *assms*:*f absolutely-integrable-on UNIV bounded-linear h*
from *absolutely-integrable-bounded-variation*[*OF assms*(*1*)] **guess** *B* **. note** *B*=*this*
from *bounded-linear.pos-bounded*[*OF assms*(*2*)] **guess** *b* **.. note** *b*=*conjunctD2*[*OF this*]
show (*h o f*) *absolutely-integrable-on UNIV*
apply(*rule bounded-variation-absolutely-integrable*[*of - B* ∗ *b*])
apply(*rule integrable-linear*[*OF - assms*(*2*)])
proof *safe* **case** *goal2*
have (∑ *k*∈*d*. *norm* (*integral k* (*h* ∘ *f*))) ≤ *setsum* (λ*k*. *norm*(*integral k f*)) *d* ∗ *b*
unfolding *setsum-left-distrib* **apply**(*rule setsum-mono*)
proof− **case** *goal1* **from** *division-ofD*(*4*)[*OF goal2 this*]
guess *u v* **apply**−**by**(*erule exE*)+ **note** *uv*=*this*
have ∗:*f integrable-on k* **unfolding** *uv* **apply**(*rule integrable-on-subinterval*[*of - UNIV*])
using *assms* **by** *auto* **note** *this*[*unfolded has-integral-integral*]
note *has-integral-linear*[*OF this assms*(*2*)] *integrable-linear*[*OF* ∗ *assms*(*2*)]
note ∗ = *has-integral-unique*[*OF this*(*2*)[*unfolded has-integral-integral*] *this*(*1*)]
show *?case* **unfolding** ∗ **using** *b* **by** *auto*
qed also have ... ≤ *B* ∗ *b* **apply**(*rule mult-right-mono*) **using** *B goal2 b* **by** *auto*
finally show *?case* **.**
qed(*insert assms*,*auto*) **qed**

lemma *absolutely-integrable-setsum*: **fixes** *f*::*'a* ⇒ *real^'n* ⇒ *real^'m*
assumes *finite t* ⋀*a*. *a* ∈ *t* ⟹ (*f a*) *absolutely-integrable-on s*

shows $(\lambda x.\ setsum\ (\lambda a.\ f\ a\ x)\ t)$ *absolutely-integrable-on s*
using *assms*(*1*,*2*) **apply** *induct* **defer apply**(*subst setsum.insert*) **apply** *assumption*+ **by**(*rule*,*auto*)

lemma *absolutely-integrable-vector-abs*:
assumes *f absolutely-integrable-on s*
shows $(\lambda x.\ (\chi\ i.\ abs(f\ x\$i))::real\hat{}'n)$ *absolutely-integrable-on s*
proof− **have** $*:\bigwedge x.\ ((\chi\ i.\ abs(f\ x\$i))::real\hat{}'n) = (setsum\ (\lambda i.$
$(((\lambda y.\ (\chi\ j.\ if\ j = i\ then\ y\$1\ else\ 0))\ o$
$(vec1\ o\ ((\lambda x.\ (norm((\chi\ j.\ if\ j = i\ then\ x\$i\ else\ 0)::real\hat{}'n)))\ o\ f)))\ x))\ UNIV)$
unfolding *Cart-eq setsum-component Cart-lambda-beta*
proof case *goal1* **have** $*:\bigwedge i\ xa.\ ((if\ i = xa\ then\ f\ x\ \$\ xa\ else\ 0) \cdot (if\ i = xa\ then\ f\ x\ \$\ xa\ else\ 0)) =$
$(if\ i = xa\ then\ (f\ x\ \$\ xa) * (f\ x\ \$\ xa)\ else\ 0)$ **by** *auto*
have $|f\ x\ \$\ i| = (setsum\ (\lambda k.\ if\ k = i\ then\ abs\ ((f\ x)\$i)\ else\ 0)\ UNIV)$
unfolding *setsum-delta*[*OF finite-UNIV*] **by** *auto*
also have $... = (\sum xa{\in}UNIV.\ ((\lambda y.\ \chi\ j.\ if\ j = xa\ then\ dest\text{-}vec1\ y\ else\ 0) \circ$
$(\lambda x.\ vec1\ (norm\ (\chi\ j.\ if\ j = xa\ then\ x\ \$\ xa\ else\ 0))) \circ f)\ x\ \$\ i)$
unfolding *norm-eq-sqrt-inner inner-vector-def Cart-lambda-beta o-def* $*$
apply(*rule setsum-cong2*) **unfolding** *setsum-delta*[*OF finite-UNIV*] **by** *auto*
finally show *?case* **unfolding** *o-def* **. qed**
show *?thesis* **unfolding** $*$ **apply**(*rule absolutely-integrable-setsum*) **apply**(*rule finite-UNIV*)
apply(*rule absolutely-integrable-linear*)
unfolding *absolutely-integrable-on-trans* **unfolding** *o-def* **apply**(*rule absolutely-integrable-norm*)
apply(*rule absolutely-integrable-linear*[*OF assms*,*unfolded o-def*]) **unfolding** *linear-linear*
apply(*rule-tac*[!] *linearI*) **unfolding** *Cart-eq* **by** *auto*
qed

lemma *absolutely-integrable-max*: **fixes** $f::real\hat{}'m \Rightarrow real\hat{}'n$
assumes *f absolutely-integrable-on s g absolutely-integrable-on s*
shows $(\lambda x.\ (\chi\ i.\ max\ (f(x)\$i)\ (g(x)\$i))::real\hat{}'n)$ *absolutely-integrable-on s*
proof− **have** $*:\bigwedge x.\ (1\ /\ 2) *_R ((\chi\ i.\ |(f\ x - g\ x)\ \$\ i|) + (f\ x + g\ x)) = (\chi\ i.$
$max\ (f(x)\$i)\ (g(x)\$i))$
unfolding *Cart-eq* **by** *auto*
note *absolutely-integrable-sub*[*OF assms*] *absolutely-integrable-add*[*OF assms*]
note *absolutely-integrable-vector-abs*[*OF this*(*1*)] *this*(*2*)
note *absolutely-integrable-add*[*OF this*] **note** *absolutely-integrable-cmul*[*OF this*,*of 1/2*]
thus *?thesis* **unfolding** $*$ **. qed**

lemma *absolutely-integrable-max-real*: **fixes** $f::real\hat{}'m \Rightarrow real$
assumes *f absolutely-integrable-on s g absolutely-integrable-on s*
shows $(\lambda x.\ max\ (f\ x)\ (g\ x))$ *absolutely-integrable-on s*
proof− **have** $*:(\lambda x.\ \chi\ i.\ max\ ((vec1 \circ f)\ x\ \$\ i)\ ((vec1 \circ g)\ x\ \$\ i)) = vec1\ o\ (\lambda x.$
$max\ (f\ x)\ (g\ x))$
apply *rule* **unfolding** *Cart-eq* **by** *auto* **note** *absolutely-integrable-max*[*of vec1 o f s vec1 o g*]

note *this*[*unfolded absolutely-integrable-on-trans*,*OF assms*]
thus *?thesis* **unfolding** $*$ **by** *auto* **qed**

lemma *absolutely-integrable-min*: **fixes** f::*real^'m* $\Rightarrow$ *real^'n*
assumes f *absolutely-integrable-on s g absolutely-integrable-on s*
shows $(\lambda x.\ (\chi\ i.\ min\ (f(x)\$i)\ (g(x)\$i))$::*real^'n*$)$ *absolutely-integrable-on s*
proof− **have** $*$:$\bigwedge x.\ (1\ /\ 2) *_R ((f\ x + g\ x) - (\chi\ i.\ |(f\ x - g\ x)\ \$\ i|)) = (\chi\ i.\ min\ (f(x)\$i)\ (g(x)\$i))$
unfolding *Cart-eq* **by** *auto*
note *absolutely-integrable-add*[*OF assms*] *absolutely-integrable-sub*[*OF assms*]
note *this*(*1*) *absolutely-integrable-vector-abs*[*OF this*(*2*)]
note *absolutely-integrable-sub*[*OF this*] **note** *absolutely-integrable-cmul*[*OF this*,*of 1/2*]
thus *?thesis* **unfolding** $*$ **. qed**

lemma *absolutely-integrable-min-real*: **fixes** f::*real^'m* $\Rightarrow$ *real*
assumes f *absolutely-integrable-on s g absolutely-integrable-on s*
shows $(\lambda x.\ min\ (f\ x)\ (g\ x))$ *absolutely-integrable-on s*
proof− **have** $*$:$(\lambda x.\ \chi\ i.\ min\ ((vec1 \circ f)\ x\ \$\ i)\ ((vec1 \circ g)\ x\ \$\ i)) = vec1\ o\ (\lambda x.\ min\ (f\ x)\ (g\ x))$
apply *rule* **unfolding** *Cart-eq* **by** *auto* **note** *absolutely-integrable-min*[*of vec1 o f s vec1 o g*]
note *this*[*unfolded absolutely-integrable-on-trans*,*OF assms*]
thus *?thesis* **unfolding** $*$ **by** *auto* **qed**

lemma *absolutely-integrable-abs-eq*: **fixes** f::*real^'n* $\Rightarrow$ *real^'m*
shows f *absolutely-integrable-on s* $\longleftrightarrow$ f *integrable-on s* $\wedge$
$(\lambda x.\ (\chi\ i.\ abs(f\ x\$i))$::*real^'m*$)$ *integrable-on s* (**is** *?l* = *?r*)
proof assume *?l* **thus** *?r* **apply**−**apply** *rule* **defer**
apply(*drule absolutely-integrable-vector-abs*) **by** *auto*
next assume *?r* **{ presume** *lem*:$\bigwedge f$::*real^'n* $\Rightarrow$ *real^'m*. f *integrable-on UNIV* $\Longrightarrow$
$(\lambda x.\ (\chi\ i.\ abs(f(x)\$i))$::*real^'m*$)$ *integrable-on UNIV* $\Longrightarrow$ f *absolutely-integrable-on UNIV*
have $*$:$\bigwedge x.\ (\chi\ i.\ |(if\ x \in s\ then\ f\ x\ else\ 0)\ \$\ i|) = (if\ x \in s\ then\ (\chi\ i.\ |f\ x\ \$\ i|)\ else\ 0)$
unfolding *Cart-eq* **by** *auto*
show *?l* **apply**(*subst absolutely-integrable-restrict-univ*[*THEN sym*]) **apply**(*rule lem*)
unfolding *integrable-restrict-univ* $*$ **using** ‹*?r*› **by** *auto* **}**
fix f::*real^'n* $\Rightarrow$ *real^'m* **assume** *assms*:f *integrable-on UNIV*
$(\lambda x.\ (\chi\ i.\ abs(f(x)\$i))$::*real^'m*$)$ *integrable-on UNIV*
let *?B* = *setsum* $(\lambda i.\ integral\ UNIV\ (\lambda x.\ (\chi\ j.\ abs(f\ x\$j))$ ::*real^'m*$)\ \$\ i)$ *UNIV*
show f *absolutely-integrable-on UNIV*
apply(*rule bounded-variation-absolutely-integrable*[*OF assms*(*1*), **where** *B*=*?B*],*safe*)
proof− **case** *goal1* **note** *d*=*this* **and** *d'*=*division-ofD*[*OF this*]
have $(\sum k \in d.\ norm\ (integral\ k\ f)) \leq$
$(\sum k \in d.\ setsum\ (op\ \$\ (integral\ k\ (\lambda x.\ \chi\ j.\ |f\ x\ \$\ j|)))\ UNIV)$ **apply**(*rule setsum-mono*)

apply(*rule order-trans*[*OF norm-le-l1*],*rule setsum-mono*)
proof− **fix** *k* **and** *i*::′*m* **assume** *k*∈*d*
from *d*′(*4*)[*OF this*] **guess** *a b* **apply**−**by**(*erule exE*)+ **note** *ab*=*this*
show |*integral k f* $ *i*| ≤ *integral k* (λ*x*. χ *j*. |*f x* $ *j*|) $ *i* **apply**(*rule abs-leI*)
unfolding *vector-uminus-component*[*THEN sym*] **defer apply**(*subst integral-neg*[*THEN sym*])
defer apply(*rule-tac*[*1*−*2*] *integral-component-le*) **apply**(*rule integrable-neg*)
using *integrable-on-subinterval*[*OF assms*(*1*),*of a b*]
integrable-on-subinterval[*OF assms*(*2*),*of a b*] **unfolding** *ab* **by** *auto*
qed also have ... ≤ *setsum* (*op* $ (*integral UNIV* (λ*x*. χ *j*. |*f x* $ *j*|))) *UNIV*
apply(*subst setsum-commute*,*rule setsum-mono*)
proof− **case** *goal1* **have** ∗:(λ*x*. χ *j*. |*f x* $ *j*|) *integrable-on* ⋃ *d*
using *integrable-on-subdivision*[*OF d assms*(*2*)] **by** *auto*
have (∑ *i*∈*d*. *integral i* (λ*x*. χ *j*. |*f x* $ *j*|) $ *j*)
= *integral* (⋃ *d*) (λ*x*. (χ *j*. *abs*(*f x*$*j*)) ::*real*ˆ′*m*) $ *j*
unfolding *setsum-component*[*THEN sym*]
apply(*subst integral-combine-division-topdown*[*THEN sym*,*OF* ∗ *d*]) **by** *auto*
also have ... ≤ *integral UNIV* (λ*x*. χ *j*. |*f x* $ *j*|) $ *j*
apply(*rule integral-subset-component-le*) **using** *assms* ∗ **by** *auto*
finally show *?case* **.**
qed finally show *?case* **. qed qed**

lemma *nonnegative-absolutely-integrable*: **fixes** *f*::*real*ˆ′*n* ⇒ *real*ˆ′*m*
assumes ∀ *x*∈*s*. ∀ *i*. *0* ≤ *f*(*x*)$*i f integrable-on s*
shows *f absolutely-integrable-on s*
unfolding *absolutely-integrable-abs-eq* **apply** *rule* **defer**
apply(*rule integrable-eq*[*of - f*]) **unfolding** *Cart-eq* **using** *assms* **by** *auto*

lemma *absolutely-integrable-integrable-bound*: **fixes** *f*::*real*ˆ′*n* ⇒ *real*ˆ′*m*
assumes ∀ *x*∈*s*. *norm*(*f x*) ≤ *g x f integrable-on s g integrable-on s*
shows *f absolutely-integrable-on s*
proof− { **presume** ∗:⋀*f*::*real*ˆ′*n* ⇒ *real*ˆ′*m*. ⋀ *g*. ∀ *x*. *norm*(*f x*) ≤ *g x* ⟹ *f integrable-on UNIV*
⟹ *g integrable-on UNIV* ⟹ *f absolutely-integrable-on UNIV*
show *?thesis* **apply**(*subst absolutely-integrable-restrict-univ*[*THEN sym*])
apply(*rule* ∗[*of - λx. if x*∈*s then g x else 0*])
using *assms* **unfolding** *integrable-restrict-univ* **by** *auto* }
fix *g* **and** *f* :: *real*ˆ′*n* ⇒ *real*ˆ′*m*
assume *assms*:∀ *x*. *norm*(*f x*) ≤ *g x f integrable-on UNIV g integrable-on UNIV*
show *f absolutely-integrable-on UNIV*
apply(*rule bounded-variation-absolutely-integrable*[*OF assms*(*2*),**where** *B*=*integral UNIV g*])
proof *safe* **case** *goal1* **note** *d*=*this* **and** *d′*=*division-ofD*[*OF this*]
have (∑ *k*∈*d*. *norm* (*integral k f*)) ≤ (∑ *k*∈*d*. *integral k g*)
apply(*rule setsum-mono*) **apply**(*rule integral-norm-bound-integral*) **apply**(*drule-tac*[!] *d*′(*4*),*safe*)
apply(*rule-tac*[*1*−*2*] *integrable-on-subinterval*) **using** *assms* **by** *auto*
also have ... = *integral* (⋃ *d*) *g* **apply**(*rule integral-combine-division-bottomup*[*THEN sym*])

apply(*rule d*,*safe*) **apply**(*drule d'*(*4*),*safe*)
apply(*rule integrable-on-subinterval*[*OF assms*(*3*)]) **by** *auto*
also have ... ≤ *integral UNIV g* **apply**(*rule integral-subset-le*) **defer**
apply(*rule integrable-on-subdivision*[*OF d*,*of - UNIV*]) **prefer** *4*
apply(*rule*,*rule-tac y*=*norm* (*f x*) **in** *order-trans*) **using** *assms* **by** *auto*
finally show *?case* **. qed qed**

lemma *absolutely-integrable-integrable-bound-real*: **fixes** *f*::*real^'n* ⇒ *real*
assumes ∀ *x*∈*s*. *norm*(*f x*) ≤ *g x f integrable-on s g integrable-on s*
shows *f absolutely-integrable-on s*
apply(*subst absolutely-integrable-on-trans*[*THEN sym*])
apply(*rule absolutely-integrable-integrable-bound*[**where** *g*=*g*])
using *assms* **unfolding** *o-def* **by** *auto*

lemma *absolutely-integrable-absolutely-integrable-bound*:
fixes *f*::*real^'n* ⇒ *real^'m* **and** *g*::*real^'n* ⇒ *real^'p*
assumes ∀ *x*∈*s*. *norm*(*f x*) ≤ *norm*(*g x*) *f integrable-on s g absolutely-integrable-on s*
shows *f absolutely-integrable-on s*
apply(*rule absolutely-integrable-integrable-bound*[*of s f* λ*x*. *norm* (*g x*)])
using *assms* **by** *auto*

lemma *absolutely-integrable-inf-real*:
assumes *finite k k* ≠ {}
∀ *i*∈*k*. (λ*x*. (*fs x i*)::*real*) *absolutely-integrable-on s*
shows (λ*x*. (*Inf* ((*fs x*) ' *k*))) *absolutely-integrable-on s* **using** *assms*
proof *induct* **case** (*insert a k*) **let** *?P* = (λ*x*. *if fs x ' k* = {} *then fs x a*
else min (*fs x a*) (*Inf* (*fs x ' k*))) *absolutely-integrable-on s*
show *?case* **unfolding** *image-insert*
apply(*subst Inf-insert-finite*) **apply**(*rule finite-imageI*[*OF insert*(*1*)])
proof(*cases k*={}) **case** *True*
thus *?P* **apply**(*subst if-P*) **defer apply**(*rule insert*(*5*)[*rule-format*]) **by** *auto*
next case *False* **thus** *?P* **apply**(*subst if-not-P*) **defer**
apply(*rule absolutely-integrable-min-real*)
defer apply(*rule insert*(*3*)[*OF False*]) **using** *insert*(*5*) **by** *auto*
qed qed *auto*

lemma *absolutely-integrable-sup-real*:
assumes *finite k k* ≠ {}
∀ *i*∈*k*. (λ*x*. (*fs x i*)::*real*) *absolutely-integrable-on s*
shows (λ*x*. (*Sup* ((*fs x*) ' *k*))) *absolutely-integrable-on s* **using** *assms*
proof *induct* **case** (*insert a k*) **let** *?P* = (λ*x*. *if fs x ' k* = {} *then fs x a*
else max (*fs x a*) (*Sup* (*fs x ' k*))) *absolutely-integrable-on s*
show *?case* **unfolding** *image-insert*
apply(*subst Sup-insert-finite*) **apply**(*rule finite-imageI*[*OF insert*(*1*)])
proof(*cases k*={}) **case** *True*
thus *?P* **apply**(*subst if-P*) **defer apply**(*rule insert*(*5*)[*rule-format*]) **by** *auto*
next case *False* **thus** *?P* **apply**(*subst if-not-P*) **defer**
apply(*rule absolutely-integrable-max-real*)

defer apply(*rule insert*(*3*)[*OF False*]) **using** *insert*(*5*) **by** *auto*
qed qed *auto*

23.68 Dominated convergence.

lemma *dominated-convergence*: **fixes** *f*::*nat* ⇒ *real*^'*n* ⇒ *real*
assumes ⋀*k*. (*f k*) *integrable-on s h integrable-on s*
⋀*k*. ∀ *x* ∈ *s*. *norm*(*f k x*) ≤ (*h x*)
∀ *x* ∈ *s*. ((λ*k*. *f k x*) −−−> *g x*) *sequentially*
shows *g integrable-on s* ((λ*k*. *integral s* (*f k*)) −−−> *integral s g*) *sequentially*
proof− **have** ⋀*m*. (λ*x*. *Inf* {*f j x* |*j*. *m* ≤ *j*}) *integrable-on s* ∧
((λ*k*. *integral s* (λ*x*. *Inf* {*f j x* |*j*. *j* ∈ {*m*..*m* + *k*}})) −−−>
integral s (λ*x*. *Inf* {*f j x* |*j*. *m* ≤ *j*}))*sequentially*
proof(*rule monotone-convergence-decreasing-real*,*safe*) **fix** *m*::*nat*
show *bounded* {*integral s* (λ*x*. *Inf* {*f j x* |*j*. *j* ∈ {*m*..*m* + *k*}}) |*k*. *True*}
unfolding *bounded-iff* **apply**(*rule-tac x*=*integral s h* **in** *exI*)
proof *safe* **fix** *k*::*nat*
show *norm* (*integral s* (λ*x*. *Inf* {*f j x* |*j*. *j* ∈ {*m*..*m* + *k*}})) ≤ *integral s h*
apply(*rule integral-norm-bound-integral*) **unfolding** *simple-image*
apply(*rule absolutely-integrable-onD*) **apply**(*rule absolutely-integrable-inf-real*)
prefer *5* **unfolding** *real-norm-def* **apply**(*rule*) **apply**(*rule Inf-abs-ge*)
prefer *5* **apply** *rule* **apply**(*rule-tac g*=*h* **in** *absolutely-integrable-integrable-bound-real*)
using *assms* **unfolding** *real-norm-def* **by** *auto*
qed fix *k*::*nat* **show** (λ*x*. *Inf* {*f j x* |*j*. *j* ∈ {*m*..*m* + *k*}}) *integrable-on s*
unfolding *simple-image* **apply**(*rule absolutely-integrable-onD*)
apply(*rule absolutely-integrable-inf-real*) **prefer** *3*
using *absolutely-integrable-integrable-bound-real*[*OF assms*(*3*,*1*,*2*)] **by** *auto*
fix *x* **assume** *x*:*x*∈*s* **show** *Inf* {*f j x* |*j*. *j* ∈ {*m*..*m* + *Suc k*}}
≤ *Inf* {*f j x* |*j*. *j* ∈ {*m*..*m* + *k*}} **apply**(*rule Inf-ge*) **unfolding** *setge-def*
defer apply *rule* **apply**(*subst Inf-finite-le-iff*) **prefer** *3*
apply(*rule-tac x*=*xa* **in** *bexI*) **by** *auto*
let *?S* = {*f j x*| *j*. *m* ≤ *j*} **def** *i* ≡ *Inf ?S*
show ((λ*k*. *Inf* {*f j x* |*j*. *j* ∈ {*m*..*m* + *k*}}) −−−> *i*) *sequentially*
unfolding *Lim-sequentially*
proof *safe* **case** *goal1* **note** *e*=*this* **have** *i*:*isGlb UNIV ?S i* **unfolding** *i-def*
apply(*rule Inf*)
defer apply(*rule-tac x*=− *h x* − *1* **in** *exI*) **unfolding** *setge-def*
proof *safe* **case** *goal1* **thus** *?case* **using** *assms*(*3*)[*rule-format*,*OF x*, *of j*]
by *auto*
qed *auto*

have ∃ *y*∈*?S*. ¬ *y* ≥ *i* + *e*
proof(*rule ccontr*) **case** *goal1* **hence** *i* ≥ *i* + *e* **apply**−
apply(*rule isGlb-le-isLb*[*OF i*]) **apply**(*rule isLbI*) **unfolding** *setge-def* **by**
fastsimp+
thus *False* **using** *e* **by** *auto*
qed then guess *y* **.. note** *y*=*this*[*unfolded not-le*]
from *this*(*1*)[*unfolded mem-Collect-eq*] **guess** *N* **.. note** *N*=*conjunctD2*[*OF*
this]

```
      show ?case apply(rule-tac x=N in exI)
      proof safe case goal1
        have *:⋀y ix. y < i + e ⟶ i ≤ ix ⟶ ix ≤ y ⟶ abs(ix − i) < e by
arith
       show ?case unfolding dist-real-def apply(rule *[rule-format,OF y(2)])
         unfolding i-def apply(rule real-le-inf-subset) prefer 3
         apply(rule,rule isGlbD1[OF i]) prefer 3 apply(subst Inf-finite-le-iff)
         prefer 3 apply(rule-tac x=y in bexI) using N goal1 by auto
      qed qed qed note dec1 = conjunctD2[OF this]

  have ⋀m. (λx. Sup {f j x |j. m ≤ j}) integrable-on s ∧
    ((λk. integral s (λx. Sup {f j x |j. j ∈ {m..m + k}})) −−−>
    integral s (λx. Sup {f j x |j. m ≤ j})) sequentially
  proof(rule monotone-convergence-increasing-real,safe) fix m::nat
    show bounded {integral s (λx. Sup {f j x |j. j ∈ {m..m + k}}) |k. True}
      unfolding bounded-iff apply(rule-tac x=integral s h in exI)
    proof safe fix k::nat
      show norm (integral s (λx. Sup {f j x |j. j ∈ {m..m + k}})) ≤ integral s h
        apply(rule integral-norm-bound-integral) unfolding simple-image
      apply(rule absolutely-integrable-onD) apply(rule absolutely-integrable-sup-real)
        prefer 5 unfolding real-norm-def apply(rule) apply(rule Sup-abs-le)
      prefer 5 apply rule apply(rule-tac g=h in absolutely-integrable-integrable-bound-real)
        using assms unfolding real-norm-def by auto
    qed fix k::nat show (λx. Sup {f j x |j. j ∈ {m..m + k}}) integrable-on s
      unfolding simple-image apply(rule absolutely-integrable-onD)
      apply(rule absolutely-integrable-sup-real) prefer 3
      using absolutely-integrable-integrable-bound-real[OF assms(3,1,2)] by auto
    fix x assume x:x∈s show Sup {f j x |j. j ∈ {m..m + Suc k}}
     ≥ Sup {f j x |j. j ∈ {m..m + k}} apply(rule Sup-le) unfolding setle-def
     defer apply rule apply(subst Sup-finite-ge-iff) prefer 3 apply(rule-tac x=y
in bexI) by auto
    let ?S = {f j x| j.  m ≤ j} def i ≡ Sup ?S
    show ((λk. Sup {f j x |j. j ∈ {m..m + k}}) −−−> i) sequentially
      unfolding Lim-sequentially
    proof safe case goal1 note e=this have i:isLub UNIV ?S i unfolding i-def
apply(rule Sup)
       defer apply(rule-tac x=h x in exI) unfolding setle-def
      proof safe case goal1 thus ?case using assms(3)[rule-format,OF x, of j]
by auto
      qed auto

      have ∃y∈?S. ¬ y ≤ i − e
      proof(rule ccontr) case goal1 hence i ≤ i − e apply−
          apply(rule isLub-le-isUb[OF i]) apply(rule isUbI) unfolding setle-def
by fastsimp+
        thus False using e by auto
      qed then guess y .. note y=this[unfolded not-le]
      from this(1)[unfolded mem-Collect-eq] guess N .. note N=conjunctD2[OF
```

this]

show *?case* **apply**(*rule-tac x=N* **in** *exI*)
proof *safe* **case** *goal1*
have *∗*:⋀*y ix. i − e < y ⟶ ix ≤ i ⟶ y ≤ ix ⟶ abs*(*ix − i*) *< e* **by** *arith*
show *?case* **unfolding** *dist-real-def* **apply**(*rule ∗*[*rule-format*,*OF y*(*2*)])
unfolding *i-def* **apply**(*rule real-ge-sup-subset*) **prefer** *3*
apply(*rule*,*rule isLubD1*[*OF i*]) **prefer** *3* **apply**(*subst Sup-finite-ge-iff*)
prefer *3* **apply**(*rule-tac x=y* **in** *bexI*) **using** *N goal1* **by** *auto*
qed qed qed note *inc1 = conjunctD2*[*OF this*]

have *g integrable-on s ∧* ((*λk. integral s* (*λx. Inf* {*f j x* |*j. k ≤ j*})) *−−−>* *integral s g*) *sequentially*
apply(*rule monotone-convergence-increasing-real*,*safe*) **apply** *fact*
proof− **show** *bounded* {*integral s* (*λx. Inf* {*f j x* |*j. k ≤ j*}) |*k. True*}
unfolding *bounded-iff* **apply**(*rule-tac x=integral s h* **in** *exI*)
proof *safe* **fix** *k*::*nat*
show *norm* (*integral s* (*λx. Inf* {*f j x* |*j. k ≤ j*})) *≤ integral s h*
apply(*rule integral-norm-bound-integral*) **apply** *fact+*
unfolding *real-norm-def* **apply**(*rule*) **apply**(*rule Inf-abs-ge*) **using** *assms*(*3*) **by** *auto*
qed fix *k*::*nat* **and** *x* **assume** *x*:*x*∈*s*

have *∗*:⋀*x y*::*real. x ≥ − y ⟹ − x ≤ y* **by** *auto*
show *Inf* {*f j x* |*j. k ≤ j*} *≤ Inf* {*f j x* |*j. Suc k ≤ j*} **apply**−
apply(*rule real-le-inf-subset*) **prefer** *3* **unfolding** *setge-def*
apply(*rule-tac x=− h x* **in** *exI*) **apply** *safe* **apply**(*rule ∗*)
using *assms*(*3*)[*rule-format*,*OF x*] **unfolding** *real-norm-def abs-le-iff* **by** *auto*
show ((*λk. Inf* {*f j x* |*j. k ≤ j*}) *−−−> g x*) *sequentially* **unfolding** *Lim-sequentially*
proof *safe* **case** *goal1* **hence** *0<e/2* **by** *auto*
from *assms*(*4*)[*unfolded Lim-sequentially*,*rule-format*,*OF x this*] **guess** *N* **..**
note *N=this*
show *?case* **apply**(*rule-tac x=N* **in** *exI*,*safe*) **unfolding** *dist-real-def*
apply(*rule le-less-trans*[*of - e/2*]) **apply**(*rule Inf-asclose*) **apply** *safe*
defer apply(*rule less-imp-le*) **using** *N goal1* **unfolding** *dist-real-def* **by** *auto*
qed qed note *inc2 = conjunctD2*[*OF this*]

have *g integrable-on s ∧* ((*λk. integral s* (*λx. Sup* {*f j x* |*j. k ≤ j*})) *−−−>* *integral s g*) *sequentially*
apply(*rule monotone-convergence-decreasing-real*,*safe*) **apply** *fact*
proof− **show** *bounded* {*integral s* (*λx. Sup* {*f j x* |*j. k ≤ j*}) |*k. True*}
unfolding *bounded-iff* **apply**(*rule-tac x=integral s h* **in** *exI*)
proof *safe* **fix** *k*::*nat*
show *norm* (*integral s* (*λx. Sup* {*f j x* |*j. k ≤ j*})) *≤ integral s h*
apply(*rule integral-norm-bound-integral*) **apply** *fact+*
unfolding *real-norm-def* **apply**(*rule*) **apply**(*rule Sup-abs-le*) **using** *assms*(*3*) **by** *auto*

qed fix *k*::*nat* **and** *x* **assume** *x*:*x*∈*s*

show *Sup* {*f j x* |*j*. *k* ≤ *j*} ≥ *Sup* {*f j x* |*j*. *Suc k* ≤ *j*} **apply**−
apply(*rule real-ge-sup-subset*) **prefer** *3* **unfolding** *setle-def*
apply(*rule-tac x*=*h x* **in** *exI*) **apply** *safe*
using *assms*(*3*)[*rule-format*,*OF x*] **unfolding** *real-norm-def abs-le-iff* **by** *auto*
show ((λ*k*. *Sup* {*f j x* |*j*. *k* ≤ *j*}) −−−> *g x*) *sequentially* **unfolding** *Lim-sequentially*
proof *safe* **case** *goal1* **hence** *0*<*e*/*2* **by** *auto*
from *assms*(*4*)[*unfolded Lim-sequentially*,*rule-format*,*OF x this*] **guess** *N* **..**
note *N*=*this*
show *?case* **apply**(*rule-tac x*=*N* **in** *exI*,*safe*) **unfolding** *dist-real-def*
apply(*rule le-less-trans*[*of - e*/*2*]) **apply**(*rule Sup-asclose*) **apply** *safe*
defer apply(*rule less-imp-le*) **using** *N goal1* **unfolding** *dist-real-def* **by** *auto*
qed qed note *dec2* = *conjunctD2*[*OF this*]

show *g integrable-on s* **by** *fact*
show ((λ*k*. *integral s* (*f k*)) −−−> *integral s g*) *sequentially* **unfolding** *Lim-sequentially*
proof *safe* **case** *goal1*
from *inc2*(*2*)[*unfolded Lim-sequentially*,*rule-format*,*OF goal1*] **guess** *N1* **..**
note *N1*=*this*[*unfolded dist-real-def*]
from *dec2*(*2*)[*unfolded Lim-sequentially*,*rule-format*,*OF goal1*] **guess** *N2* **..**
note *N2*=*this*[*unfolded dist-real-def*]
show *?case* **apply**(*rule-tac x*=*N1*+*N2* **in** *exI*,*safe*)
proof− **fix** *n* **assume** *n*:*n* ≥ *N1* + *N2*
have ∗:⋀*i0 i i1 g*. |*i0* − *g*| < *e* ⟶ |*i1* − *g*| < *e* ⟶ *i0* ≤ *i* ⟶ *i* ≤ *i1* ⟶ |*i* − *g*| < *e* **by** *arith*
show *dist* (*integral s* (*f n*)) (*integral s g*) < *e* **unfolding** *dist-real-def*
apply(*rule* ∗[*rule-format*,*OF N1*[*rule-format*] *N2*[*rule-format*], *of n n*])
proof− **show** *integral s* (λ*x*. *Inf* {*f j x* |*j*. *n* ≤ *j*}) ≤ *integral s* (*f n*)
proof(*rule integral-le*[*OF dec1*(*1*) *assms*(*1*)],*safe*)
fix *x* **assume** *x*:*x* ∈ *s* **have** ∗:⋀*x y*::*real*. *x* ≥ − *y* ⟹ − *x* ≤ *y* **by** *auto*
show *Inf* {*f j x* |*j*. *n* ≤ *j*} ≤ *f n x* **apply**(*rule Inf-lower*[**where** *z*=− *h x*]) **defer**
apply(*rule* ∗) **using** *assms*(*3*)[*rule-format*,*OF x*] **unfolding** *real-norm-def abs-le-iff* **by** *auto*
qed show *integral s* (*f n*) ≤ *integral s* (λ*x*. *Sup* {*f j x* |*j*. *n* ≤ *j*})
proof(*rule integral-le*[*OF assms*(*1*) *inc1*(*1*)],*safe*)
fix *x* **assume** *x*:*x* ∈ *s*
show *f n x* ≤ *Sup* {*f j x* |*j*. *n* ≤ *j*} **apply**(*rule Sup-upper*[**where** *z*=*h x*]) **defer**
using *assms*(*3*)[*rule-format*,*OF x*] **unfolding** *real-norm-def abs-le-iff* **by** *auto*
qed qed(*insert n*,*auto*) **qed qed qed**

declare [[*smt-certificates*=]]
declare [[*smt-fixed*=*false*]]

end

24 Real-Integration: Integration on real intervals

theory *Real-Integration*
imports *Integration*
begin

We follow John Harrison in formalizing the Gauge integral.

definition *Integral* :: *real set* ⇒ (*real* ⇒ *real*) ⇒ *real* ⇒ *bool* **where**
Integral s f k = (*f o dest-vec1 has-integral k*) (*vec1 ' s*)

lemmas *integral-unfold* = *Integral-def split-conv o-def vec1-interval*

lemma *Integral-unique*:
[| *Integral*{*a..b*} *f k1*; *Integral*{*a..b*} *f k2* |] ==> *k1* = *k2*
unfolding *integral-unfold* **apply**(*rule has-integral-unique*) **by** *assumption*+

lemma *Integral-zero* [*simp*]: *Integral*{*a..a*} *f 0*
unfolding *integral-unfold* **by** *auto*

lemma *Integral-eq-diff-bounds*: **assumes** *a* ≤ *b* **shows** *Integral*{*a..b*} (%*x*. *1*) (*b* − *a*)
unfolding *integral-unfold* **using** *has-integral-const*[*of 1*::*real vec1 a vec1 b*]
unfolding *content-1'*[*OF assms*] **by** *auto*

lemma *Integral-mult-const*: **assumes** *a* ≤ *b* **shows** *Integral*{*a..b*} (%*x*. *c*) (*c*∗(*b* − *a*))
unfolding *integral-unfold* **using** *has-integral-const*[*of c*::*real vec1 a vec1 b*]
unfolding *content-1'*[*OF assms*] **by**(*auto simp add*:*field-simps*)

lemma *Integral-mult*: **assumes** *Integral*{*a..b*} *f k* **shows** *Integral*{*a..b*} (%*x*. *c* ∗ *f x*) (*c* ∗ *k*)
using *assms* **unfolding** *integral-unfold* **apply**(*drule-tac has-integral-cmul*[**where** *c*=*c*]) **by** *auto*

lemma *Integral-add*:
assumes *Integral* {*a..b*} *f x1 Integral* {*b..c*} *f x2* *a* ≤ *b* **and** *b* ≤ *c*
shows *Integral* {*a..c*} *f* (*x1* + *x2*)
using *assms* **unfolding** *integral-unfold* **apply**−
apply(*rule has-integral-combine*[*of vec1 a vec1 b vec1 c*]) **by** *auto*

lemma *FTC1*: **assumes** *a* ≤ *b* ∀*x*. *a* ≤ *x* & *x* ≤ *b* −−> *DERIV f x* :> *f'*(*x*)
shows *Integral*{*a..b*} *f'* (*f*(*b*) − *f*(*a*))
proof−**note** *fundamental-theorem-of-calculus*[*OF assms*(*1*), *off o dest-vec1 f' o dest-vec1*]
note ∗ = *this*[*unfolded o-def vec1-dest-vec1*]

have $**$:$\bigwedge x.\ (\lambda xa{::}real.\ xa * f'\ x) =\ op * (f'\ x)$ **apply**(*rule ext*) **by**(*auto simp add*:*field-simps*)
show *?thesis* **unfolding** *integral-unfold* **apply**(*rule* $*$)
using *assms*(*2*) **unfolding** *DERIV-conv-has-derivative has-vector-derivative-def*
apply *safe* **apply**(*rule has-derivative-at-within*) **by**(*auto simp add*:$**$) **qed**

lemma *Integral-subst*: [| *Integral*{*a*..*b*} *f k1*; *k2*=*k1* |] ==> *Integral*{*a*..*b*} *f k2*
by *simp*

24.1 Additivity Theorem of Gauge Integral

Bartle/Sherbert: Theorem 10.1.5 p. 278

lemma *Integral-add-fun*: [| *Integral*{*a*..*b*} *f k1*; *Integral*{*a*..*b*} *g k2* |] ==> *Integral*{*a*..*b*} (%*x*. *f x* + *g x*) (*k1* + *k2*)
unfolding *integral-unfold* **apply**(*rule has-integral-add*) **by** *assumption*+

lemma *norm-vec1*'[*simp*]:*norm* (*vec1 x*) = *norm x*
using *norm-vector-1*[*of vec1 x*] **by** *auto*

lemma *Integral-le*: **assumes** $a \le b$ $\forall x.\ a \le x$ & $x \le b$ --> $f(x) \le g(x)$ *Integral*{*a*..*b*} *f k1 Integral*{*a*..*b*} *g k2* **shows** *k1* ≤ *k2*
proof− **note** *assms*(*3*−*4*)[*unfolded integral-unfold*] **note** *has-integral-vec1*[*OF this*(*1*)] *has-integral-vec1*[*OF this*(*2*)]
note *has-integral-component-le*[*OF this*,*of 1*] **thus** *?thesis* **using** *assms*(*2*) **by** *auto* **qed**

lemma *monotonic-anti-derivative*:
fixes *f g* :: *real* => *real* **shows**
[| $a \le b$; $\forall c.\ a \le c$ & $c \le b$ --> $f'\ c \le g'\ c$;
$\forall x.\ DERIV\ f\ x :> f'\ x$; $\forall x.\ DERIV\ g\ x :> g'\ x$ |]
==> $f\ b - f\ a \le g\ b - g\ a$
apply (*rule Integral-le*, *assumption*)
apply (*auto intro*: *FTC1*)
done

end

25 Path-Connected: Continuous paths and path-connected sets

theory *Path-Connected*
imports *Convex-Euclidean-Space*
begin

25.1 Paths.

definition

path :: (*real* ⇒ *'a*::*topological-space*) ⇒ *bool*
where *path g* ⟷ *continuous-on* {*0* .. *1*} *g*

definition
pathstart :: (*real* ⇒ *'a*::*topological-space*) ⇒ *'a*
where *pathstart g* = *g 0*

definition
pathfinish :: (*real* ⇒ *'a*::*topological-space*) ⇒ *'a*
where *pathfinish g* = *g 1*

definition
path-image :: (*real* ⇒ *'a*::*topological-space*) ⇒ *'a set*
where *path-image g* = *g* ‘ {*0* .. *1*}

definition
reversepath :: (*real* ⇒ *'a*::*topological-space*) ⇒ (*real* ⇒ *'a*)
where *reversepath g* = (λ*x*. *g*(*1* − *x*))

definition
joinpaths :: (*real* ⇒ *'a*::*topological-space*) ⇒ (*real* ⇒ *'a*) ⇒ (*real* ⇒ *'a*)
(**infixr** +++ *75*)
where *g1* +++ *g2* = (λ*x*. *if x* ≤ *1*/*2 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*))

definition
simple-path :: (*real* ⇒ *'a*::*topological-space*) ⇒ *bool*
where *simple-path g* ⟷
(∀ *x*∈{*0*..*1*}. ∀ *y*∈{*0*..*1*}. *g x* = *g y* ⟶ *x* = *y* ∨ *x* = *0* ∧ *y* = *1* ∨ *x* = *1* ∧ *y* = *0*)

definition
injective-path :: (*real* ⇒ *'a*::*topological-space*) ⇒ *bool*
where *injective-path g* ⟷ (∀ *x*∈{*0*..*1*}. ∀ *y*∈{*0*..*1*}. *g x* = *g y* ⟶ *x* = *y*)

25.2 Some lemmas about these concepts.

lemma *injective-imp-simple-path*:
injective-path g ⟹ *simple-path g*
unfolding *injective-path-def simple-path-def* **by** *auto*

lemma *path-image-nonempty*: *path-image g* ≠ {}
unfolding *path-image-def image-is-empty interval-eq-empty* **by** *auto*

lemma *pathstart-in-path-image*[*intro*]: (*pathstart g*) ∈ *path-image g*
unfolding *pathstart-def path-image-def* **by** *auto*

lemma *pathfinish-in-path-image*[*intro*]: (*pathfinish g*) ∈ *path-image g*
unfolding *pathfinish-def path-image-def* **by** *auto*

lemma *connected-path-image*[*intro*]: *path g* ⟹ *connected*(*path-image g*)
unfolding *path-def path-image-def*
apply (*erule connected-continuous-image*)
by(*rule convex-connected*, *rule convex-real-interval*)

lemma *compact-path-image*[*intro*]: *path g* ⟹ *compact*(*path-image g*)
unfolding *path-def path-image-def*
by (*erule compact-continuous-image*, *rule compact-real-interval*)

lemma *reversepath-reversepath*[*simp*]: *reversepath*(*reversepath g*) = *g*
unfolding *reversepath-def* **by** *auto*

lemma *pathstart-reversepath*[*simp*]: *pathstart*(*reversepath g*) = *pathfinish g*
unfolding *pathstart-def reversepath-def pathfinish-def* **by** *auto*

lemma *pathfinish-reversepath*[*simp*]: *pathfinish*(*reversepath g*) = *pathstart g*
unfolding *pathstart-def reversepath-def pathfinish-def* **by** *auto*

lemma *pathstart-join*[*simp*]: *pathstart*(*g1* +++ *g2*) = *pathstart g1*
unfolding *pathstart-def joinpaths-def pathfinish-def* **by** *auto*

lemma *pathfinish-join*[*simp*]:*pathfinish*(*g1* +++ *g2*) = *pathfinish g2*
unfolding *pathstart-def joinpaths-def pathfinish-def* **by** *auto*

lemma *path-image-reversepath*[*simp*]: *path-image*(*reversepath g*) = *path-image g*
proof−
have ∗:⋀*g*. *path-image*(*reversepath g*) ⊆ *path-image g*
unfolding *path-image-def subset-eq reversepath-def Ball-def image-iff* **apply**(*rule*,*rule*,*erule bexE*)
apply(*rule-tac x=1 − xa* **in** *bexI*) **by** *auto*
show *?thesis* **using** ∗[*of g*] ∗[*of reversepath g*] **unfolding** *reversepath-reversepath*
by *auto* **qed**

lemma *path-reversepath*[*simp*]: *path*(*reversepath g*) ⟷ *path g* **proof**−
have ∗:⋀*g*. *path g* ⟹ *path*(*reversepath g*) **unfolding** *path-def reversepath-def*
apply(*rule continuous-on-compose*[*unfolded o-def*, *of* - λ*x*. *1 − x*])
apply(*rule continuous-on-sub*, *rule continuous-on-const*, *rule continuous-on-id*)
apply(*rule continuous-on-subset*[*of* {*0..1*}], *assumption*) **by** *auto*
show *?thesis* **using** ∗[*of reversepath g*] ∗[*of g*] **unfolding** *reversepath-reversepath*
by (*rule iffI*) **qed**

lemmas *reversepath-simps* = *path-reversepath path-image-reversepath pathstart-reversepath pathfinish-reversepath*

lemma *path-join*[*simp*]: **assumes** *pathfinish g1* = *pathstart g2* **shows** *path* (*g1* +++ *g2*) ⟷ *path g1* ∧ *path g2*
unfolding *path-def pathfinish-def pathstart-def* **apply** *rule* **defer apply**(*erule conjE*) **proof**−
assume *as*:*continuous-on* {*0..1*} (*g1* +++ *g2*)

have $*$:*g1* = $(\lambda x.\ g1\ (2 *_R x)) \circ (\lambda x.\ (1/2) *_R x)$
g2 = $(\lambda x.\ g2\ (2 *_R x - 1)) \circ (\lambda x.\ (1/2) *_R (x + 1))$
unfolding *o-def* **by** (*auto simp add*: *add-divide-distrib*)
have *op* $*_R$ *(1 / 2)* ‘ $\{0$::*real*..*1*$\} \subseteq \{0..1\}$ $(\lambda x.\ (1\ /\ 2) *_R (x + 1))$ ‘ $\{(0$::*real*$)..1\} \subseteq \{0..1\}$
by *auto*
thus *continuous-on* $\{0..1\}$ *g1* $\wedge$ *continuous-on* $\{0..1\}$ *g2* **apply** −**apply** *rule*
apply(*subst* $*$) **defer apply**(*subst* $*$) **apply** (*rule-tac*[!] *continuous-on-compose*)
apply (*rule continuous-on-cmul*, *rule continuous-on-add*, *rule continuous-on-id*, *rule continuous-on-const*) **defer**
apply (*rule continuous-on-cmul*, *rule continuous-on-id*) **apply**(*rule-tac*[!] *continuous-on-eq*[*of* - *g1* +++ *g2*]) **defer prefer** *3*
apply(*rule-tac*[*1*−*2*] *continuous-on-subset*[*of* $\{0$ .. *1*$\}$]) **apply**(*rule as*, *assumption*, *rule as*, *assumption*)
apply(*rule*) **defer apply** *rule* **proof**−
fix *x* **assume** *x* ∈ *op* $*_R$ *(1 / 2)* ‘ $\{0$::*real*..*1*$\}$
hence *x* ≤ *1 / 2* **unfolding** *image-iff* **by** *auto*
thus *(g1 +++ g2) x = g1 (2* $*_R$ *x)* **unfolding** *joinpaths-def* **by** *auto* **next**
fix *x* **assume** *x* ∈ $(\lambda x.\ (1\ /\ 2) *_R (x + 1))$ ‘ $\{0$::*real*..*1*$\}$
hence *x* ≥ *1 / 2* **unfolding** *image-iff* **by** *auto*
thus *(g1 +++ g2) x = g2 (2* $*_R$ *x* − *1)* **proof**(*cases x = 1 / 2*)
case *True* **hence** *x = (1/2)* $*_R$ *1* **unfolding** *Cart-eq* **by** *auto*
thus *?thesis* **unfolding** *joinpaths-def* **using** *assms*[*unfolded pathstart-def pathfinish-def*] **by** (*auto simp add*: *mult-ac*)
qed (*auto simp add*:*le-less joinpaths-def*) **qed**
next assume *as*:*continuous-on* $\{0..1\}$ *g1 continuous-on* $\{0..1\}$ *g2*
have $*$:$\{0$ .. *1*::*real*$\} = \{0..$ *(1/2)*$*_R$ *1*$\} \cup \{$*(1/2)* $*_R$ *1* .. *1*$\}$ **by** *auto*
have $**$:*op* $*_R$ *2* ‘ $\{0..$*(1 / 2)* $*_R$ *1*$\} = \{0..1$::*real*$\}$ **apply**(*rule set-ext*, *rule*) **unfolding** *image-iff*
defer apply(*rule-tac x*=*(1/2)*$*_R$ *x* **in** *bexI*) **by** *auto*
have $***$:$(\lambda x.\ 2 *_R x - 1)$ ‘ $\{(1\ /\ 2) *_R 1..1\} = \{0..1$::*real*$\}$
apply (*auto simp add*: *image-def*)
apply (*rule-tac x*=*(x + 1) / 2* **in** *bexI*)
apply (*auto simp add*: *add-divide-distrib*)
done
show *continuous-on* $\{0..1\}$ *(g1 +++ g2)* **unfolding** $*$ **apply**(*rule continuous-on-union*) **apply** (*rule closed-real-atLeastAtMost*)+ **proof**−
show *continuous-on* $\{0..$*(1 / 2)* $*_R$ *1*$\}$ *(g1 +++ g2)* **apply**(*rule continuous-on-eq*[*of* - $\lambda x.$ *g1 (2* $*_R$ *x)*]) **defer**
unfolding *o-def*[*THEN sym*] **apply**(*rule continuous-on-compose*) **apply**(*rule continuous-on-cmul*, *rule continuous-on-id*)
unfolding $**$ **apply**(*rule as(1)*) **unfolding** *joinpaths-def* **by** *auto* **next**
show *continuous-on* $\{$*(1/2)*$*_R$*1..1*$\}$ *(g1 +++ g2)* **apply**(*rule continuous-on-eq*[*of* - *g2* ∘ $(\lambda x.\ 2 *_R x - 1)$]) **defer**
apply(*rule continuous-on-compose*) **apply**(*rule continuous-on-sub*, *rule continuous-on-cmul*, *rule continuous-on-id*, *rule continuous-on-const*)
unfolding $***$ *o-def joinpaths-def* **apply**(*rule as(2)*) **using** *assms*[*unfolded pathstart-def pathfinish-def*]
by (*auto simp add*: *mult-ac*) **qed qed**

lemma *path-image-join-subset*: *path-image*(*g1 +++ g2*) ⊆ (*path-image g1* ∪ *path-image g2*) **proof**
fix *x* **assume** *x* ∈ *path-image* (*g1 +++ g2*)
then obtain *y* **where** *y*:*y*∈{*0..1*} *x* = (*if y* ≤ *1 / 2 then g1* (*2* ∗$_R$ *y*) *else g2* (*2* ∗$_R$ *y* − *1*))
unfolding *path-image-def image-iff joinpaths-def* **by** *auto*
thus *x* ∈ *path-image g1* ∪ *path-image g2* **apply**(*cases y* ≤ *1/2*)
apply(*rule-tac UnI1*) **defer apply**(*rule-tac UnI2*) **unfolding** *y*(*2*) *path-image-def* **using** *y*(*1*)
by(*auto intro*!: *imageI*) **qed**

lemma *subset-path-image-join*:
assumes *path-image g1* ⊆ *s path-image g2* ⊆ *s* **shows** *path-image*(*g1 +++ g2*) ⊆ *s*
using *path-image-join-subset*[*of g1 g2*] **and** *assms* **by** *auto*

lemma *path-image-join*:
assumes *path g1 path g2 pathfinish g1* = *pathstart g2*
shows *path-image*(*g1 +++ g2*) = (*path-image g1*) ∪ (*path-image g2*)
apply(*rule*, *rule path-image-join-subset*, *rule*) **unfolding** *Un-iff* **proof**(*erule disjE*)
fix *x* **assume** *x* ∈ *path-image g1*
then obtain *y* **where** *y*:*y*∈{*0..1*} *x* = *g1 y* **unfolding** *path-image-def image-iff* **by** *auto*
thus *x* ∈ *path-image* (*g1 +++ g2*) **unfolding** *joinpaths-def path-image-def image-iff*
apply(*rule-tac x*=(*1/2*) ∗$_R$ *y* **in** *bexI*) **by** *auto* **next**
fix *x* **assume** *x* ∈ *path-image g2*
then obtain *y* **where** *y*:*y*∈{*0..1*} *x* = *g2 y* **unfolding** *path-image-def image-iff* **by** *auto*
then show *x* ∈ *path-image* (*g1 +++ g2*) **unfolding** *joinpaths-def path-image-def image-iff*
apply(*rule-tac x*=(*1/2*) ∗$_R$ (*y* + *1*) **in** *bexI*) **using** *assms*(*3*)[*unfolded pathfinish-def pathstart-def*]
by (*auto simp add*: *add-divide-distrib*) **qed**

lemma *not-in-path-image-join*:
assumes *x* ∉ *path-image g1 x* ∉ *path-image g2* **shows** *x* ∉ *path-image*(*g1 +++ g2*)
using *assms* **and** *path-image-join-subset*[*of g1 g2*] **by** *auto*

lemma *simple-path-reversepath*: **assumes** *simple-path g* **shows** *simple-path* (*reversepath g*)
using *assms* **unfolding** *simple-path-def reversepath-def* **apply**− **apply**(*rule ballI*)+
apply(*erule-tac x*=*1*−*x* **in** *ballE*, *erule-tac x*=*1*−*y* **in** *ballE*)
by *auto*

lemma *simple-path-join-loop*:

assumes *injective-path g1 injective-path g2 pathfinish g2 = pathstart g1*
(path-image g1 ∩ path-image g2) ⊆ {pathstart g1,pathstart g2}
shows *simple-path(g1 +++ g2)*
unfolding *simple-path-def* **proof**((*rule ballI*)+, *rule impI*) **let** *?g = g1 +++ g2*
note *inj = assms(1,2)[unfolded injective-path-def, rule-format]*
fix *x y*::*real* **assume** *xy*:*x ∈ {0..1} y ∈ {0..1} ?g x = ?g y*
show *x = y ∨ x = 0 ∧ y = 1 ∨ x = 1 ∧ y = 0* **proof**(*case-tac x ≤ 1/2*,*case-tac*[!] *y ≤ 1/2*, *unfold not-le*)
assume *as*:*x ≤ 1 / 2 y ≤ 1 / 2*
hence *g1 (2 *R x) = g1 (2 *R y)* **using** *xy(3)* **unfolding** *joinpaths-def* **by** *auto*
moreover have *2 *R x ∈ {0..1} 2 *R y ∈ {0..1}* **using** *xy(1,2) as*
by *auto*
ultimately show *?thesis* **using** *inj(1)[of 2*R x 2*R y]* **by** *auto*
next assume *as*:*x > 1 / 2 y > 1 / 2*
hence *g2 (2 *R x − 1) = g2 (2 *R y − 1)* **using** *xy(3)* **unfolding** *joinpaths-def* **by** *auto*
moreover have *2 *R x − 1 ∈ {0..1} 2 *R y − 1 ∈ {0..1}* **using** *xy(1,2) as* **by** *auto*
ultimately show *?thesis* **using** *inj(2)[of 2*R x − 1 2*R y − 1]* **by** *auto*
next assume *as*:*x ≤ 1 / 2 y > 1 / 2*
hence *?g x ∈ path-image g1 ?g y ∈ path-image g2* **unfolding** *path-image-def joinpaths-def*
using *xy(1,2)* **by** *auto*
moreover have *?g y ≠ pathstart g2* **using** *as(2)* **unfolding** *pathstart-def joinpaths-def*
using *inj(2)[of 2 *R y − 1 0]* **and** *xy(2)*
by (*auto simp add*: *field-simps*)
ultimately have *∗:?g x = pathstart g1* **using** *assms(4)* **unfolding** *xy(3)* **by** *auto*
hence *x = 0* **unfolding** *pathstart-def joinpaths-def* **using** *as(1)* **and** *xy(1)*
using *inj(1)[of 2 *R x 0]* **by** *auto*
moreover have *y = 1* **using** *∗* **unfolding** *xy(3) assms(3)[THEN sym]*
unfolding *joinpaths-def pathfinish-def* **using** *as(2)* **and** *xy(2)*
using *inj(2)[of 2 *R y − 1 1]* **by** *auto*
ultimately show *?thesis* **by** *auto*
next assume *as*:*x > 1 / 2 y ≤ 1 / 2*
hence *?g x ∈ path-image g2 ?g y ∈ path-image g1* **unfolding** *path-image-def joinpaths-def*
using *xy(1,2)* **by** *auto*
moreover have *?g x ≠ pathstart g2* **using** *as(1)* **unfolding** *pathstart-def joinpaths-def*
using *inj(2)[of 2 *R x − 1 0]* **and** *xy(1)*
by (*auto simp add*: *field-simps*)
ultimately have *∗:?g y = pathstart g1* **using** *assms(4)* **unfolding** *xy(3)* **by** *auto*
hence *y = 0* **unfolding** *pathstart-def joinpaths-def* **using** *as(2)* **and** *xy(2)*
using *inj(1)[of 2 *R y 0]* **by** *auto*
moreover have *x = 1* **using** *∗* **unfolding** *xy(3)[THEN sym] assms(3)[THEN*

sym]
unfolding *joinpaths-def pathfinish-def* **using** *as*(*1*) **and** *xy*(*1*)
using *inj*(*2*)[*of 2* $*_R$ *x − 1 1*] **by** *auto*
ultimately show *?thesis* **by** *auto* **qed qed**

lemma *injective-path-join*:
assumes *injective-path g1 injective-path g2 pathfinish g1 = pathstart g2*
(*path-image g1* ∩ *path-image g2*) ⊆ {*pathstart g2*}
shows *injective-path*(*g1 +++ g2*)
unfolding *injective-path-def* **proof**(*rule*,*rule*,*rule*) **let** *?g = g1 +++ g2*
note *inj = assms*(*1*,*2*)[*unfolded injective-path-def*, *rule-format*]
fix *x y* **assume** *xy*:*x* ∈ {*0..1*} *y* ∈ {*0..1*} (*g1 +++ g2*) *x* = (*g1 +++ g2*) *y*
show *x = y* **proof**(*cases x* ≤ *1/2*, *case-tac*[!] *y* ≤ *1/2*, *unfold not-le*)
assume *x* ≤ *1 / 2 y* ≤ *1 / 2* **thus** *?thesis* **using** *inj*(*1*)[*of 2*$*_R$ *x 2*$*_R$ *y*] **and** *xy*
unfolding *joinpaths-def* **by** *auto*
next assume *x > 1 / 2 y > 1 / 2* **thus** *?thesis* **using** *inj*(*2*)[*of 2*$*_R$ *x − 1 2*$*_R$ *y − 1*] **and** *xy*
unfolding *joinpaths-def* **by** *auto*
next assume *as*:*x* ≤ *1 / 2 y > 1 / 2*
hence *?g x* ∈ *path-image g1 ?g y* ∈ *path-image g2* **unfolding** *path-image-def joinpaths-def*
using *xy*(*1*,*2*) **by** *auto*
hence *?g x = pathfinish g1 ?g y = pathstart g2* **using** *assms*(*4*) **unfolding** *assms*(*3*) *xy*(*3*) **by** *auto*
thus *?thesis* **using** *as* **and** *inj*(*1*)[*of 2* $*_R$ *x 1*] *inj*(*2*)[*of 2* $*_R$ *y − 1 0*] **and** *xy*(*1*,*2*)
unfolding *pathstart-def pathfinish-def joinpaths-def*
by *auto*
next assume *as*:*x > 1 / 2 y* ≤ *1 / 2*
hence *?g x* ∈ *path-image g2 ?g y* ∈ *path-image g1* **unfolding** *path-image-def joinpaths-def*
using *xy*(*1*,*2*) **by** *auto*
hence *?g x = pathstart g2 ?g y = pathfinish g1* **using** *assms*(*4*) **unfolding** *assms*(*3*) *xy*(*3*) **by** *auto*
thus *?thesis* **using** *as* **and** *inj*(*2*)[*of 2* $*_R$ *x − 1 0*] *inj*(*1*)[*of 2* $*_R$ *y 1*] **and** *xy*(*1*,*2*)
unfolding *pathstart-def pathfinish-def joinpaths-def*
by *auto* **qed qed**

lemmas *join-paths-simps = path-join path-image-join pathstart-join pathfinish-join*

25.3 Reparametrizing a closed curve to start at some chosen point.

definition *shiftpath a* (*f*::*real* ⇒ *'a*::*topological-space*) =
(λ*x*. *if* (*a* + *x*) ≤ *1 then f*(*a* + *x*) *else f*(*a* + *x* − *1*))

lemma *pathstart-shiftpath*: *a* ≤ *1* ⟹ *pathstart*(*shiftpath a g*) = *g a*

unfolding *pathstart-def shiftpath-def* **by** *auto*

lemma *pathfinish-shiftpath*: **assumes** $0 \leq a$ *pathfinish* $g =$ *pathstart* g
shows *pathfinish*(*shiftpath* a g) $=$ g a
using *assms* **unfolding** *pathstart-def pathfinish-def shiftpath-def*
by *auto*

lemma *endpoints-shiftpath*:
assumes *pathfinish* $g =$ *pathstart* g $a \in \{0 \,..\, 1\}$
shows *pathfinish*(*shiftpath* a g) $= g\ a$ *pathstart*(*shiftpath* a g) $= g\ a$
using *assms* **by**(*auto intro*!:*pathfinish-shiftpath pathstart-shiftpath*)

lemma *closed-shiftpath*:
assumes *pathfinish* $g =$ *pathstart* g $a \in \{0..1\}$
shows *pathfinish*(*shiftpath* a g) $=$ *pathstart*(*shiftpath* a g)
using *endpoints-shiftpath*[*OF assms*] **by** *auto*

lemma *path-shiftpath*:
assumes *path* g *pathfinish* $g =$ *pathstart* g $a \in \{0..1\}$
shows *path*(*shiftpath* a g) **proof**−
have $*$:$\{0 \,..\, 1\} = \{0 \,..\, 1-a\} \cup \{1-a \,..\, 1\}$ **using** *assms*(*3*) **by** *auto*
have $**$:$\bigwedge x.\ x + a = 1 \Longrightarrow g\ (x + a - 1) = g\ (x + a)$
using *assms*(*2*)[*unfolded pathfinish-def pathstart-def*] **by** *auto*
show *?thesis* **unfolding** *path-def shiftpath-def* $*$ **apply**(*rule continuous-on-union*)
apply(*rule closed-real-atLeastAtMost*)+ **apply**(*rule continuous-on-eq*[*of - g* $\circ$ $(\lambda x.\ a + x)$]) **prefer** *3*
apply(*rule continuous-on-eq*[*of - g* $\circ$ $(\lambda x.\ a - 1 + x)$]) **defer prefer** *3*
apply(*rule continuous-on-intros*)+ **prefer** *2* **apply**(*rule continuous-on-intros*)+
apply(*rule-tac*[*1−2*] *continuous-on-subset*[*OF assms*(*1*)[*unfolded path-def*]])
using *assms*(*3*) **and** $**$ **by**(*auto*, *auto simp add*: *field-simps*) **qed**

lemma *shiftpath-shiftpath*: **assumes** *pathfinish* $g =$ *pathstart* g $a \in \{0..1\}$ $x \in \{0..1\}$
shows *shiftpath* $(1 - a)$ (*shiftpath* a g) $x = g\ x$
using *assms* **unfolding** *pathfinish-def pathstart-def shiftpath-def* **by** *auto*

lemma *path-image-shiftpath*:
assumes $a \in \{0..1\}$ *pathfinish* $g =$ *pathstart* g
shows *path-image*(*shiftpath* a g) $=$ *path-image* g **proof**−
{ fix x **assume** *as*:$g\ 1 = g\ 0$ $x \in \{0..1$::*real*$\}$ $\forall\, y \in \{0..1\} \cap \{x.\ \neg\ a + x \leq 1\}.\ g\ x \neq g\ (a + y - 1)$
hence $\exists\, y \in \{0..1\} \cap \{x.\ a + x \leq 1\}.\ g\ x = g\ (a + y)$ **proof**(*cases* $a \leq x$)
case *False* **thus** *?thesis* **apply**(*rule-tac* x=$1 + x - a$ **in** *bexI*)
using *as*(*1*,*2*) **and** *as*(*3*)[*THEN bspec*[**where** x=$1 + x - a$]] **and** *assms*(*1*)
by(*auto simp add*: *field-simps atomize-not*) **next**
case *True* **thus** *?thesis* **using** *as*(*1−2*) **and** *assms*(*1*) **apply**(*rule-tac* x=$x - a$ **in** *bexI*)
by(*auto simp add*: *field-simps*) **qed** **}**
thus *?thesis* **using** *assms* **unfolding** *shiftpath-def path-image-def pathfinish-def*

pathstart-def
by(*auto simp add*: *image-iff*) **qed**

25.4 Special case of straight-line paths.

definition
linepath :: *'a*::*real-normed-vector* ⇒ *'a* ⇒ *real* ⇒ *'a* **where**
linepath a b = (λ*x*. (*1* − *x*) *∗R a* + *x ∗R b*)

lemma *pathstart-linepath*[*simp*]: *pathstart*(*linepath a b*) = *a*
unfolding *pathstart-def linepath-def* **by** *auto*

lemma *pathfinish-linepath*[*simp*]: *pathfinish*(*linepath a b*) = *b*
unfolding *pathfinish-def linepath-def* **by** *auto*

lemma *continuous-linepath-at*[*intro*]: *continuous* (*at x*) (*linepath a b*)
unfolding *linepath-def* **by** (*intro continuous-intros*)

lemma *continuous-on-linepath*[*intro*]: *continuous-on s* (*linepath a b*)
using *continuous-linepath-at* **by**(*auto intro*!: *continuous-at-imp-continuous-on*)

lemma *path-linepath*[*intro*]: *path*(*linepath a b*)
unfolding *path-def* **by**(*rule continuous-on-linepath*)

lemma *path-image-linepath*[*simp*]: *path-image*(*linepath a b*) = (*closed-segment a b*)
unfolding *path-image-def segment linepath-def* **apply** (*rule set-ext*, *rule*) **defer**
unfolding *mem-Collect-eq image-iff* **apply**(*erule exE*) **apply**(*rule-tac x*=*u ∗R 1* **in** *bexI*)
by *auto*

lemma *reversepath-linepath*[*simp*]: *reversepath*(*linepath a b*) = *linepath b a*
unfolding *reversepath-def linepath-def* **by**(*rule ext*, *auto*)

lemma *injective-path-linepath*:
assumes *a* ≠ *b* **shows** *injective-path*(*linepath a b*)
proof −
{ fix *x y* :: *real*
assume *x ∗R b* + *y ∗R a* = *x ∗R a* + *y ∗R b*
hence (*x* − *y*) *∗R a* = (*x* − *y*) *∗R b* **by** (*simp add*: *algebra-simps*)
with *assms* **have** *x* = *y* **by** *simp* **}**
thus *?thesis* **unfolding** *injective-path-def linepath-def* **by**(*auto simp add*: *algebra-simps*)
qed

lemma *simple-path-linepath*[*intro*]: *a* ≠ *b* ⟹ *simple-path*(*linepath a b*) **by**(*auto intro*!: *injective-imp-simple-path injective-path-linepath*)

25.5 Bounding a point away from a path.

lemma *not-on-path-ball*:

fixes $g :: real \Rightarrow 'a::heine\text{-}borel$
assumes $path\ g\ z \notin path\text{-}image\ g$
shows $\exists e{>}0.\ ball\ z\ e \cap (path\text{-}image\ g) = \{\}$ **proof**−
obtain a **where** $a{\in}path\text{-}image\ g\ \forall y{\in}path\text{-}image\ g.\ dist\ z\ a \leq dist\ z\ y$
using *distance-attains-inf*[*OF - path-image-nonempty, of g z*]
using *compact-path-image*[*THEN compact-imp-closed, OF assms*(*1*)] **by** *auto*
thus *?thesis* **apply**(*rule-tac x=dist z a* **in** *exI*) **using** *assms*(*2*) **by**(*auto intro*!: *dist-pos-lt*) **qed**

lemma *not-on-path-cball*:
fixes $g :: real \Rightarrow 'a::heine\text{-}borel$
assumes $path\ g\ z \notin path\text{-}image\ g$
shows $\exists e{>}0.\ cball\ z\ e \cap (path\text{-}image\ g) = \{\}$ **proof**−
obtain e **where** $ball\ z\ e \cap path\text{-}image\ g = \{\}\ e{>}0$ **using** *not-on-path-ball*[*OF assms*] **by** *auto*
moreover have $cball\ z\ (e/2) \subseteq ball\ z\ e$ **using** ‹$e{>}0$› **by** *auto*
ultimately show *?thesis* **apply**(*rule-tac x=e/2* **in** *exI*) **by** *auto* **qed**

25.6 Path component, considered as a "joinability" relation (from Tom Hales).

definition $path\text{-}component\ s\ x\ y \longleftrightarrow (\exists g.\ path\ g \wedge path\text{-}image\ g \subseteq s \wedge pathstart\ g = x \wedge pathfinish\ g = y)$

lemmas *path-defs = path-def pathstart-def pathfinish-def path-image-def path-component-def*

lemma *path-component-mem*: **assumes** *path-component s x y* **shows** $x \in s\ y \in s$
using *assms* **unfolding** *path-defs* **by** *auto*

lemma *path-component-refl*: **assumes** $x \in s$ **shows** *path-component s x x*
unfolding *path-defs* **apply**(*rule-tac x=λu. x* **in** *exI*) **using** *assms*
by(*auto intro*!:*continuous-on-intros*)

lemma *path-component-refl-eq*: *path-component s x x* $\longleftrightarrow x \in s$
by(*auto intro*!: *path-component-mem path-component-refl*)

lemma *path-component-sym*: *path-component s x y* $\Longrightarrow$ *path-component s y x*
using *assms* **unfolding** *path-component-def* **apply**(*erule exE*) **apply**(*rule-tac x=reversepath g* **in** *exI*)
by *auto*

lemma *path-component-trans*: **assumes** *path-component s x y path-component s y z* **shows** *path-component s x z*
using *assms* **unfolding** *path-component-def* **apply**− **apply**(*erule exE*)+ **apply**(*rule-tac x=g +++ ga* **in** *exI*) **by**(*auto simp add*: *path-image-join*)

lemma *path-component-of-subset*: $s \subseteq t \Longrightarrow$ *path-component s x y* $\Longrightarrow$ *path-component t x y*

unfolding *path-component-def* **by** *auto*

25.7 Can also consider it as a set, as the name suggests.

lemma *path-component-set*: *path-component s x* = { *y*. (∃ *g*. *path g* ∧ *path-image g* ⊆ *s* ∧ *pathstart g* = *x* ∧ *pathfinish g* = *y*)}
apply(*rule set-ext*) **unfolding** *mem-Collect-eq* **unfolding** *mem-def path-component-def* **by** *auto*

lemma *mem-path-component-set*:*x* ∈ *path-component s y* ⟷ *path-component s y x* **unfolding** *mem-def* **by** *auto*

lemma *path-component-subset*: (*path-component s x*) ⊆ *s*
apply(*rule*, *rule path-component-mem*(*2*)) **by**(*auto simp add*:*mem-def*)

lemma *path-component-eq-empty*: *path-component s x* = {} ⟷ *x* ∉ *s*
apply *rule* **apply**(*drule equals0D*[*of* - *x*]) **defer apply**(*rule equals0I*) **unfolding** *mem-path-component-set*
apply(*drule path-component-mem*(*1*)) **using** *path-component-refl* **by** *auto*

25.8 Path connectedness of a space.

definition *path-connected s* ⟷ (∀ *x*∈*s*. ∀ *y*∈*s*. ∃ *g*. *path g* ∧ (*path-image g*) ⊆ *s* ∧ *pathstart g* = *x* ∧ *pathfinish g* = *y*)

lemma *path-connected-component*: *path-connected s* ⟷ (∀ *x*∈*s*. ∀ *y*∈*s*. *path-component s x y*)
unfolding *path-connected-def path-component-def* **by** *auto*

lemma *path-connected-component-set*: *path-connected s* ⟷ (∀ *x*∈*s*. *path-component s x* = *s*)
unfolding *path-connected-component* **apply**(*rule*, *rule*, *rule*, *rule path-component-subset*)

unfolding *subset-eq mem-path-component-set Ball-def mem-def* **by** *auto*

25.9 Some useful lemmas about path-connectedness.

lemma *convex-imp-path-connected*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *convex s* **shows** *path-connected s*
unfolding *path-connected-def* **apply**(*rule*,*rule*,*rule-tac x*=*linepath x y* **in** *exI*)
unfolding *path-image-linepath* **using** *assms*[*unfolded convex-contains-segment*]
by *auto*

lemma *path-connected-imp-connected*: **assumes** *path-connected s* **shows** *connected s*
unfolding *connected-def not-ex* **apply**(*rule*,*rule*,*rule ccontr*) **unfolding** *not-not* **apply**(*erule conjE*)+ **proof**−
fix *e1 e2* **assume** *as*:*open e1 open e2 s* ⊆ *e1* ∪ *e2 e1* ∩ *e2* ∩ *s* = {} *e1* ∩ *s* ≠ {} *e2* ∩ *s* ≠ {}

then obtain *x1 x2* **where** *obt*:*x1*∈*e1*∩*s x2*∈*e2*∩*s* **by** *auto*
then obtain *g* **where** *g*:*path g path-image g* ⊆ *s pathstart g* = *x1 pathfinish g* = *x2*
using *assms*[*unfolded path-connected-def*,*rule-format*,*of x1 x2*] **by** *auto*
have ∗:*connected* {*0..1*::*real*} **by**(*auto intro*!: *convex-connected convex-real-interval*)
have {*0..1*} ⊆ {*x* ∈ {*0..1*}. *g x* ∈ *e1*} ∪ {*x* ∈ {*0..1*}. *g x* ∈ *e2*} **using** *as*(*3*) *g*(*2*)[*unfolded path-defs*] **by** *blast*
moreover have {*x* ∈ {*0..1*}. *g x* ∈ *e1*} ∩ {*x* ∈ {*0..1*}. *g x* ∈ *e2*} = {} **using** *as*(*4*) *g*(*2*)[*unfolded path-defs*] **unfolding** *subset-eq* **by** *auto*
moreover have {*x* ∈ {*0..1*}. *g x* ∈ *e1*} ≠ {} ∧ {*x* ∈ {*0..1*}. *g x* ∈ *e2*} ≠ {} **using** *g*(*3*,*4*)[*unfolded path-defs*] **using** *obt*
by (*simp add*: *ex-in-conv* [*symmetric*], *metis zero-le-one order-refl*)
ultimately show *False* **using** ∗[*unfolded connected-local not-ex*,*rule-format*, *of* {*x*∈{*0..1*}. *g x* ∈ *e1*} {*x*∈{*0..1*}. *g x* ∈ *e2*}]
using *continuous-open-in-preimage*[*OF g*(*1*)[*unfolded path-def*] *as*(*1*)]
using *continuous-open-in-preimage*[*OF g*(*1*)[*unfolded path-def*] *as*(*2*)] **by** *auto*
qed

lemma *open-path-component*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *open s* **shows** *open*(*path-component s x*)
unfolding *open-contains-ball* **proof**
fix *y* **assume** *as*:*y* ∈ *path-component s x*
hence *y*∈*s* **apply**− **apply**(*rule path-component-mem*(*2*)) **unfolding** *mem-def* **by** *auto*
then obtain *e* **where** *e*:*e*>*0 ball y e* ⊆ *s* **using** *assms*[*unfolded open-contains-ball*] **by** *auto*
show ∃ *e*>*0*. *ball y e* ⊆ *path-component s x* **apply**(*rule-tac x*=*e* **in** *exI*) **apply**(*rule*,*rule* ‹*e*>*0*›,*rule*) **unfolding** *mem-ball mem-path-component-set* **proof**−
fix *z* **assume** *dist y z* < *e* **thus** *path-component s x z* **apply**(*rule-tac path-component-trans*[*of* - - *y*]) **defer**
apply(*rule path-component-of-subset*[*OF e*(*2*)]) **apply**(*rule convex-imp-path-connected*[*OF convex-ball*, *unfolded path-connected-component*, *rule-format*]) **using** ‹*e*>*0*›
using *as*[*unfolded mem-def*] **by** *auto* **qed qed**

lemma *open-non-path-component*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *open s* **shows** *open*(*s* − *path-component s x*)
unfolding *open-contains-ball* **proof**
fix *y* **assume** *as*:*y*∈*s* − *path-component s x*
then obtain *e* **where** *e*:*e*>*0 ball y e* ⊆ *s* **using** *assms*[*unfolded open-contains-ball*] **by** *auto*
show ∃ *e*>*0*. *ball y e* ⊆ *s* − *path-component s x* **apply**(*rule-tac x*=*e* **in** *exI*) **apply**(*rule*,*rule* ‹*e*>*0*›,*rule*,*rule*) **defer proof**(*rule ccontr*)
fix *z* **assume** *z*∈*ball y e* ¬ *z* ∉ *path-component s x*
hence *y* ∈ *path-component s x* **unfolding** *not-not mem-path-component-set* **using** ‹*e*>*0*›
apply− **apply**(*rule path-component-trans*,*assumption*) **apply**(*rule path-component-of-subset*[*OF e*(*2*)])

apply(*rule convex-imp-path-connected*[*OF convex-ball*, *unfolded path-connected-component*, *rule-format*]) **by** *auto*
thus *False* **using** *as* **by** *auto* **qed**(*insert e*(*2*), *auto*) **qed**

lemma *connected-open-path-connected*:
fixes *s* :: '*a*::*real-normed-vector set*
assumes *open s connected s* **shows** *path-connected s*
unfolding *path-connected-component-set* **proof**(*rule*,*rule*,*rule path-component-subset*, *rule*)
fix *x y* **assume** *x* ∈ *s y* ∈ *s* **show** *y* ∈ *path-component s x* **proof**(*rule ccontr*)
assume *y* ∉ *path-component s x* **moreover**
have *path-component s x* ∩ *s* ≠ {} **using** ‹*x*∈*s*› *path-component-eq-empty path-component-subset*[*of s x*] **by** *auto*
ultimately show *False* **using** ‹*y*∈*s*› *open-non-path-component*[*OF assms*(*1*)] *open-path-component*[*OF assms*(*1*)]
using *assms*(*2*)[*unfolded connected-def not-ex*, *rule-format*, *ofpath-component s x s* − *path-component s x*] **by** *auto*
qed qed

lemma *path-connected-continuous-image*:
assumes *continuous-on s f path-connected s* **shows** *path-connected* (*f* ‘ *s*)
unfolding *path-connected-def* **proof**(*rule*,*rule*)
fix *x*′ *y*′ **assume** *x*′ ∈ *f* ‘ *s y*′ ∈ *f* ‘ *s*
then obtain *x y* **where** *xy*:*x*∈*s y*∈*s x*′ = *f x y*′ = *f y* **by** *auto*
guess *g* **using** *assms*(*2*)[*unfolded path-connected-def*,*rule-format*,*OF xy*(*1*,*2*)] **..**
thus ∃ *g*. *path g* ∧ *path-image g* ⊆ *f* ‘ *s* ∧ *pathstart g* = *x*′ ∧ *pathfinish g* = *y*′
unfolding *xy* **apply**(*rule-tac x*=*f* ∘ *g* **in** *exI*) **unfolding** *path-defs*
using *assms*(*1*) **by**(*auto intro*!: *continuous-on-compose continuous-on-subset*[*of* - - *g* ‘ {*0*..*1*}]) **qed**

lemma *homeomorphic-path-connectedness*:
s homeomorphic t ⟹ (*path-connected s* ⟷ *path-connected t*)
unfolding *homeomorphic-def homeomorphism-def* **apply**(*erule exE*|*erule conjE*)+
apply *rule*
apply(*drule-tac f*=*f* **in** *path-connected-continuous-image*) **prefer** *3*
apply(*drule-tac f*=*g* **in** *path-connected-continuous-image*) **by** *auto*

lemma *path-connected-empty*: *path-connected* {}
unfolding *path-connected-def* **by** *auto*

lemma *path-connected-singleton*: *path-connected* {*a*}
unfolding *path-connected-def pathstart-def pathfinish-def path-image-def*
apply (*clarify*, *rule-tac x*=λ*x*. *a* **in** *exI*, *simp add*: *image-constant-conv*)
apply (*simp add*: *path-def continuous-on-const*)
done

lemma *path-connected-Un*: **assumes** *path-connected s path-connected t s* ∩ *t* ≠ {}
shows *path-connected* (*s* ∪ *t*) **unfolding** *path-connected-component* **proof**(*rule*,*rule*)
fix *x y* **assume** *as*:*x* ∈ *s* ∪ *t y* ∈ *s* ∪ *t*

from *assms*(*3*) **obtain** *z* **where** $z \in s \cap t$ **by** *auto*
thus *path-component* $(s \cup t)$ *x y* **using** *as* **using** *assms*(*1−2*)[*unfolded path-connected-component*]
apply−
apply(*erule-tac*[!] *UnE*)+ **apply**(*rule-tac*[*2−3*] *path-component-trans*[*of - - z*])
by(*auto simp add*:*path-component-of-subset* [*OF Un-upper1*] *path-component-of-subset*[*OF Un-upper2*]) **qed**

25.10 sphere is path-connected.

lemma *path-connected-punctured-universe*:
assumes $2 \leq CARD('n::finite)$ **shows** *path-connected*((*UNIV*::(*real*^'*n*) *set*) − $\{a\}$) **proof**−
obtain ψ **where** ψ:*bij-betw* ψ $\{1..CARD('n)\}$ (*UNIV*::'*n set*) **using** *ex-bij-betw-nat-finite-1*[*OF finite-UNIV*] **by** *auto*
let *?U* = *UNIV*::(*real*^'*n*) *set* **let** *?u* = *?U* − $\{0\}$
let *?basis* = $\lambda k.$ *basis* (ψ *k*)
let *?A* = $\lambda k.$ $\{x::real\hat{}'n.\ \exists i{\in}\{1..k\}.$ *inner* (*basis* (ψ *i*)) *x* ≠ *0*$\}$
have $\forall k{\in}\{2..CARD('n)\}.$ *path-connected* (*?A k*) **proof**
have ∗:$\bigwedge k.$ *?A* (*Suc k*) = $\{x.$ *inner* (*?basis* (*Suc k*)) *x* < *0*$\}$ ∪ $\{x.$ *inner* (*?basis* (*Suc k*)) *x* > *0*$\}$ ∪ *?A k* **apply**(*rule set-ext*,*rule*) **defer**
apply(*erule UnE*)+ **unfolding** *mem-Collect-eq* **apply**(*rule-tac*[*1−2*] *x*=*Suc k* **in** *bexI*)
by(*auto elim*!: *ballE simp add*: *not-less le-Suc-eq*)
fix *k* **assume** $k \in \{2..CARD('n)\}$ **thus** *path-connected* (*?A k*) **proof**(*induct k*)
case (*Suc k*) **show** *?case* **proof**(*cases k* = *1*)
case *False* **from** *Suc* **have** *d*:$k \in \{1..CARD('n)\}$ *Suc k* $\in \{1..CARD('n)\}$ **by** *auto*
hence ψ *k* ≠ ψ (*Suc k*) **using** ψ[*unfolded bij-betw-def inj-on-def*, *THEN conjunct1*, *THEN bspec*[**where** *x*=*k*]] **by** *auto*
hence ∗∗:*?basis k* + *?basis* (*Suc k*) ∈ $\{x.$ *0* < *inner* (*?basis* (*Suc k*)) *x*$\}$ ∩ (*?A k*)
?basis k − *?basis* (*Suc k*) ∈ $\{x.$ *0* > *inner* (*?basis* (*Suc k*)) *x*$\}$ ∩ ($\{x.$ *0* < *inner* (*?basis* (*Suc k*)) *x*$\}$ ∪ (*?A k*)) **using** *d*
by(*auto simp add*: *inner-basis intro*!:*bexI*[**where** *x*=*k*])
show *?thesis* **unfolding** ∗ *Un-assoc* **apply**(*rule path-connected-Un*) **defer**
apply(*rule path-connected-Un*)
prefer *5* **apply**(*rule-tac*[*1−2*] *convex-imp-path-connected*, *rule convex-halfspace-lt*, *rule convex-halfspace-gt*)
apply(*rule Suc*(*1*)) **using** *d* ∗∗ *False* **by** *auto*
next case *True* **hence** *d*:*1*∈$\{1..CARD('n)\}$ *2*∈$\{1..CARD('n)\}$ **using** *Suc*(*2*) **by** *auto*
have ∗∗∗:*Suc 1* = *2* **by** *auto*
have ∗∗:$\bigwedge s\ t\ P\ Q.$ $s \cup t \cup \{x.\ P\ x \vee Q\ x\} = (s \cup \{x.\ P\ x\}) \cup (t \cup \{x.\ Q\ x\})$ **by** *auto*
have *nequals0I*:$\bigwedge x\ A.$ *x*∈*A* ⟹ *A* ≠ {} **by** *auto*
have ψ *2* ≠ ψ (*Suc 0*) **using** ψ[*unfolded bij-betw-def inj-on-def*, *THEN conjunct1*, *THEN bspec*[**where** *x*=*2*]] **using** *assms* **by** *auto*
thus *?thesis* **unfolding** ∗ *True* **unfolding** ∗∗ *neq-iff bex-disj-distrib* **apply**

–
apply(*rule path-connected-Un*, *rule-tac*[*1*−*2*] *path-connected-Un*) **defer** *3*
apply(*rule-tac*[*1*−*4*] *convex-imp-path-connected*)
apply(*rule-tac*[*5*] *x*= *?basis 1* + *?basis 2* **in** *nequals0I*)
apply(*rule-tac*[*6*] *x*=− *?basis 1* + *?basis 2* **in** *nequals0I*)
apply(*rule-tac*[*7*] *x*=− *?basis 1* − *?basis 2* **in** *nequals0I*)
using *d* **unfolding** ∗∗∗ **by**(*auto intro*!: *convex-halfspace-gt convex-halfspace-lt*, *auto simp add*: *inner-basis*)
qed qed *auto* **qed note** *lem* = *this*

have ∗∗∗:⋀*x*::*real*^*'n*. (∃ *i*∈{*1*..*CARD*(*'n*)}. *inner* (*basis* (*ψ i*)) *x* ≠ *0*) ⟷ (∃ *i*. *inner* (*basis i*) *x* ≠ *0*)
apply *rule* **apply**(*erule bexE*) **apply**(*rule-tac x*=*ψ i* **in** *exI*) **defer apply**(*erule exE*) **proof**−
fix *x*::*real*^*'n* **and** *i* **assume** *as*:*inner* (*basis i*) *x* ≠ *0*
have *i*∈*ψ* ‘ {*1*..*CARD*(*'n*)} **using** *ψ*[*unfolded bij-betw-def*, *THEN conjunct2*] **by** *auto*
then obtain *j* **where** *j*∈{*1*..*CARD*(*'n*)} *ψ j* = *i* **by** *auto*
thus ∃ *i*∈{*1*..*CARD*(*'n*)}. *inner* (*basis* (*ψ i*)) *x* ≠ *0* **apply**(*rule-tac x*=*j* **in** *bexI*) **using** *as* **by** *auto* **qed** *auto*
have ∗:*?U* − {*a*} = (λ*x*. *x* + *a*) ‘ {*x*. *x* ≠ *0*} **apply**(*rule set-ext*) **unfolding** *image-iff*
apply *rule* **apply**(*rule-tac x*=*x* − *a* **in** *bexI*) **by** *auto*
have ∗∗:⋀*x*::*real*^*'n*. *x*≠*0* ⟷ (∃ *i*. *inner* (*basis i*) *x* ≠ *0*) **unfolding** *Cart-eq* **by**(*auto simp add*: *inner-basis*)
show *?thesis* **unfolding** ∗ **apply**(*rule path-connected-continuous-image*) **apply**(*rule continuous-on-intros*)+
unfolding ∗∗ **apply**(*rule lem*[*THEN bspec*[**where** *x*=*CARD*(*'n*)], *unfolded* ∗∗∗]) **using** *assms* **by** *auto* **qed**

lemma *path-connected-sphere*: **assumes** *2* ≤ *CARD*(*'n*::*finite*) **shows** *path-connected* {*x*::*real*^*'n*. *norm*(*x* − *a*) = *r*} **proof**(*cases r*≤*0*)
case *True* **thus** *?thesis* **proof**(*cases r*=*0*)
case *False* **hence** {*x*::*real*^*'n*. *norm*(*x* − *a*) = *r*} = {} **using** *True* **by** *auto*
thus *?thesis* **using** *path-connected-empty* **by** *auto*
qed(*auto intro*!:*path-connected-singleton*) **next**
case *False* **hence** ∗:{*x*::*real*^*'n*. *norm*(*x* − *a*) = *r*} = (λ*x*. *a* + *r* ∗*R* *x*) ‘ {*x*. *norm x* = *1*} **unfolding** *not-le* **apply** −**apply**(*rule set-ext*,*rule*)
unfolding *image-iff* **apply**(*rule-tac x*=(*1*/*r*) ∗*R* (*x* − *a*) **in** *bexI*) **unfolding** *mem-Collect-eq norm-scaleR* **by** (*auto simp add*: *scaleR-right-diff-distrib*)
have ∗∗:{*x*::*real*^*'n*. *norm x* = *1*} = (λ*x*. (*1*/*norm x*) ∗*R* *x*) ‘ (*UNIV* − {*0*}) **apply**(*rule set-ext*,*rule*)
unfolding *image-iff* **apply**(*rule-tac x*=*x* **in** *bexI*) **unfolding** *mem-Collect-eq* **by**(*auto split*:*split-if-asm*)
have *continuous-on* (*UNIV* − {*0*}) (λ*x*::*real*^*'n*. *1* / *norm x*) **unfolding** *o-def continuous-on-eq-continuous-within*
apply(*rule*, *rule continuous-at-within-inv*[*unfolded o-def inverse-eq-divide*]) **apply**(*rule continuous-at-within*)
apply(*rule continuous-at-norm*[*unfolded o-def*]) **by** *auto*

thus *?thesis* **unfolding** $*$ $**$ **using** *path-connected-punctured-universe*[*OF assms*]
by(*auto intro*!: *path-connected-continuous-image continuous-on-intros*) **qed**

lemma *connected-sphere*: $2 \leq CARD('n) \Longrightarrow$ *connected* $\{x::real\hat{}'n.\ norm(x - a) = r\}$
using *path-connected-sphere path-connected-imp-connected* **by** *auto*

end

26 Fashoda: Fashoda meet theorem.

theory *Fashoda*
imports *Brouwer-Fixpoint Vec1 Path-Connected*
begin

26.1 Fashoda meet theorem.

lemma *infnorm-2*: *infnorm* $(x::real\hat{}2) = max\ (abs(x\$1))\ (abs(x\$2))$
unfolding *infnorm-def UNIV-2* **apply**(*rule Sup-eq*) **by** *auto*

lemma *infnorm-eq-1-2*: *infnorm* $(x::real\hat{}2) = 1 \longleftrightarrow$
$(abs(x\$1) \leq 1 \wedge abs(x\$2) \leq 1 \wedge (x\$1 = -1 \vee x\$1 = 1 \vee x\$2 = -1 \vee x\$2 = 1))$
unfolding *infnorm-2* **by** *auto*

lemma *infnorm-eq-1-imp*: **assumes** *infnorm* $(x::real\hat{}2) = 1$ **shows** $abs(x\$1) \leq 1\ abs(x\$2) \leq 1$
using *assms* **unfolding** *infnorm-eq-1-2* **by** *auto*

lemma *fashoda-unit*: **fixes** $f\ g::real \Rightarrow real\hat{}2$
assumes $f\ `\ \{-1..1\} \subseteq \{-1..1\}\ g\ `\ \{-1..1\} \subseteq \{-1..1\}$
continuous-on $\{-1..1\}\ f$ *continuous-on* $\{-1..1\}\ g$
$f\ (-1)\$1 = -1\ f\ 1\$1 = 1\ g\ (-1)\ \$2 = -1\ g\ 1\ \$2 = 1$
shows $\exists s{\in}\{-1..1\}.\ \exists t{\in}\{-1..1\}.\ f\ s = g\ t$ **proof**(*rule ccontr*)
case *goal1* **note** *as* = *this*[*unfolded bex-simps*,*rule-format*]
def *sqprojection* $\equiv \lambda z::real\hat{}2.\ (inverse\ (infnorm\ z)) *_R z$
def *negatex* $\equiv \lambda x::real\hat{}2.\ (vector\ [-(x\$1),\ x\$2])::real\hat{}2$
have *lem1*:$\forall\ z::real\hat{}2.\ infnorm(negatex\ z) = infnorm\ z$
unfolding *negatex-def infnorm-2 vector-2* **by** *auto*
have *lem2*:$\forall\ z.\ z{\neq}0 \longrightarrow infnorm(sqprojection\ z) = 1$ **unfolding** *sqprojection-def*
unfolding *infnorm-mul*[*unfolded smult-conv-scaleR*] **unfolding** *abs-inverse real-abs-infnorm*
unfolding *infnorm-eq-0*[*THEN sym*] **by** *auto*
let *?F* $= (\lambda w::real\hat{}2.\ (f \circ (\lambda x.\ x\$1))\ w - (g \circ (\lambda x.\ x\$2))\ w)$
have $*:\bigwedge i.\ (\lambda x::real\hat{}2.\ x\ \$\ i)\ `\ \{-1..1\} = \{-1..1::real\}$
apply(*rule set-ext*) **unfolding** *image-iff Bex-def mem-interval* **apply** *rule* **defer**
apply(*rule-tac x*=*vec x* **in** *exI*) **by** *auto*

{ fix *x* **assume** *x* ∈ (λ*w*. (*f* ∘ (λ*x*. *x* $ *1*)) *w* − (*g* ∘ (λ*x*. *x* $ *2*)) *w*) ‘ {− *1..1*::*real^2*}
then guess *w* **unfolding** *image-iff* **.. note** *w* = *this*
hence *x* ≠ *0* **using** *as*[*of w$1 w$2*] **unfolding** *mem-interval* **by** *auto*} **note** *x0*=*this*
have *21*:⋀*i*::*2*. *i*≠*1* ⟹ *i*=*2* **using** *UNIV-2* **by** *auto*
have *1*:{− *1*<..<*1*::*real^2*} ≠ {} **unfolding** *interval-eq-empty* **by** *auto*
have *2*:*continuous-on* {− *1..1*} (*negatex* ∘ *sqprojection* ∘ *?F*) **apply**(*rule continuous-on-intros continuous-on-component continuous-on-vec1*)+
prefer *2* **apply**(*rule continuous-on-intros continuous-on-component continuous-on-vec1*)+
unfolding ∗
apply(*rule assms*)+ **apply**(*rule continuous-on-compose*,*subst sqprojection-def*)
apply(*rule continuous-on-mul*) **apply**(*rule continuous-at-imp-continuous-on*,*rule*)
apply(*rule continuous-at-inv*[*unfolded o-def*])
apply(*rule continuous-at-infnorm*) **unfolding** *infnorm-eq-0* **defer apply**(*rule continuous-on-id*) **apply**(*rule linear-continuous-on*) **proof**−
show *bounded-linear negatex* **apply**(*rule bounded-linearI'*) **unfolding** *Cart-eq* **proof**(*rule-tac*[!] *allI*) **fix** *i*::*2* **and** *x y*::*real^2* **and** *c*::*real*
show *negatex* (*x* + *y*) $ *i* = (*negatex x* + *negatex y*) $ *i negatex* (*c* ∗*R x*) $ *i* = (*c* ∗*R negatex x*) $ *i*
apply−**apply**(*case-tac*[!] *i*≠*1*) **prefer** *3* **apply**(*drule-tac*[*1*−*2*] *21*)
unfolding *negatex-def* **by**(*auto simp add*:*vector-2*) **qed qed**(*insert x0*, *auto*)
have *3*:(*negatex* ∘ *sqprojection* ∘ *?F*) ‘ {− *1..1*} ⊆ {− *1..1*} **unfolding** *subset-eq* **apply** *rule* **proof**−
case *goal1* **then guess** *y* **unfolding** *image-iff* **.. note** *y*=*this* **have** *?F y* ≠ *0* **apply**(*rule x0*) **using** *y*(*1*) **by** *auto*
hence ∗:*infnorm* (*sqprojection* (*?F y*)) = *1* **unfolding** *y o-def* **apply**− **by**(*rule lem2*[*rule-format*])
have *infnorm x* = *1* **unfolding** ∗[*THEN sym*] *y o-def* **by**(*rule lem1*[*rule-format*])
thus *x*∈{− *1..1*} **unfolding** *mem-interval infnorm-2* **apply**− **apply** *rule*
proof−**case** *goal1* **thus** *?case* **apply**(*cases i*=*1*) **defer apply**(*drule 21*) **by** *auto* **qed qed**
guess *x* **apply**(*rule brouwer-weak*[*of* {− *1..1*::*real^2*} *negatex* ∘ *sqprojection* ∘ *?F*])
apply(*rule compact-interval convex-interval*)+ **unfolding** *interior-closed-interval*
apply(*rule 1 2 3*)+ **. note** *x*=*this*
have *?F x* ≠ *0* **apply**(*rule x0*) **using** *x*(*1*) **by** *auto*
hence ∗:*infnorm* (*sqprojection* (*?F x*)) = *1* **unfolding** *o-def* **by**(*rule lem2*[*rule-format*])
have *nx*:*infnorm x* = *1* **apply**(*subst x*(*2*)[*THEN sym*]) **unfolding** ∗[*THEN sym*] *o-def* **by**(*rule lem1*[*rule-format*])
have ∀ *x i*. *x* ≠ *0* ⟶ (*0* < (*sqprojection x*)$*i* ⟷ *0* < *x*$*i*) ∀ *x i*. *x* ≠ *0* ⟶ ((*sqprojection x*)$*i* < *0* ⟷ *x*$*i* < *0*)
apply− **apply**(*rule-tac*[!] *allI impI*)+ **proof**− **fix** *x*::*real^2* **and** *i*::*2* **assume** *x*:*x*≠*0*
have *inverse* (*infnorm x*) > *0* **using** *x*[*unfolded infnorm-pos-lt*[*THEN sym*]] **by** *auto*
thus (*0* < *sqprojection x* $ *i*) = (*0* < *x* $ *i*) (*sqprojection x* $ *i* < *0*) = (*x* $ *i* < *0*)

unfolding *sqprojection-def vector-component-simps Cart-nth.scaleR real-scaleR-def*
unfolding *zero-less-mult-iff mult-less-0-iff* **by**(*auto simp add:field-simps*) **qed**
note *lem3* = *this*[*rule-format*]
have *x1*:*x* $ *1* ∈ {− *1..1*::*real*} *x* $ *2* ∈ {− *1..1*::*real*} **using** *x*(*1*) **unfolding** *mem-interval* **by** *auto*
hence *nz*:*f* (*x* $ *1*) − *g* (*x* $ *2*) ≠ *0* **unfolding** *right-minus-eq* **apply**−**apply**(*rule as*) **by** *auto*
have *x* $ *1* = −*1* ∨ *x* $ *1* = *1* ∨ *x* $ *2* = −*1* ∨ *x* $ *2* = *1* **using** *nx* **unfolding** *infnorm-eq-1-2* **by** *auto*
thus *False* **proof**− **fix** *P Q R S*
presume *P* ∨ *Q* ∨ *R* ∨ *S* *P*⟹*False* *Q*⟹*False* *R*⟹*False* *S*⟹*False* **thus** *False* **by** *auto*
next assume *as*:*x*$*1* = *1*
hence ∗:*f* (*x* $ *1*) $ *1* = *1* **using** *assms*(*6*) **by** *auto*
have *sqprojection* (*f* (*x*$*1*) − *g* (*x*$*2*)) $ *1* < *0*
using *x*(*2*)[*unfolded o-def Cart-eq*,*THEN spec*[**where** *x*=*1*]]
unfolding *as negatex-def vector-2* **by** *auto* **moreover**
from *x1* **have** *g* (*x* $ *2*) ∈ {− *1..1*} **apply**−**apply**(*rule assms*(*2*)[*unfolded subset-eq*,*rule-format*]) **by** *auto*
ultimately show *False* **unfolding** *lem3*[*OF nz*] *vector-component-simps* ∗ *mem-interval*
apply(*erule-tac x*=*1* **in** *allE*) **by** *auto*
next assume *as*:*x*$*1* = −*1*
hence ∗:*f* (*x* $ *1*) $ *1* = − *1* **using** *assms*(*5*) **by** *auto*
have *sqprojection* (*f* (*x*$*1*) − *g* (*x*$*2*)) $ *1* > *0*
using *x*(*2*)[*unfolded o-def Cart-eq*,*THEN spec*[**where** *x*=*1*]]
unfolding *as negatex-def vector-2* **by** *auto* **moreover**
from *x1* **have** *g* (*x* $ *2*) ∈ {− *1..1*} **apply**−**apply**(*rule assms*(*2*)[*unfolded subset-eq*,*rule-format*]) **by** *auto*
ultimately show *False* **unfolding** *lem3*[*OF nz*] *vector-component-simps* ∗ *mem-interval*
apply(*erule-tac x*=*1* **in** *allE*) **by** *auto*
next assume *as*:*x*$*2* = *1*
hence ∗:*g* (*x* $ *2*) $ *2* = *1* **using** *assms*(*8*) **by** *auto*
have *sqprojection* (*f* (*x*$*1*) − *g* (*x*$*2*)) $ *2* > *0*
using *x*(*2*)[*unfolded o-def Cart-eq*,*THEN spec*[**where** *x*=*2*]]
unfolding *as negatex-def vector-2* **by** *auto* **moreover**
from *x1* **have** *f* (*x* $ *1*) ∈ {− *1..1*} **apply**−**apply**(*rule assms*(*1*)[*unfolded subset-eq*,*rule-format*]) **by** *auto*
ultimately show *False* **unfolding** *lem3*[*OF nz*] *vector-component-simps* ∗ *mem-interval*
apply(*erule-tac x*=*2* **in** *allE*) **by** *auto*
next assume *as*:*x*$*2* = −*1*
hence ∗:*g* (*x* $ *2*) $ *2* = − *1* **using** *assms*(*7*) **by** *auto*
have *sqprojection* (*f* (*x*$*1*) − *g* (*x*$*2*)) $ *2* < *0*
using *x*(*2*)[*unfolded o-def Cart-eq*,*THEN spec*[**where** *x*=*2*]]
unfolding *as negatex-def vector-2* **by** *auto* **moreover**
from *x1* **have** *f* (*x* $ *1*) ∈ {− *1..1*} **apply**−**apply**(*rule assms*(*1*)[*unfolded subset-eq*,*rule-format*]) **by** *auto*

ultimately show *False* **unfolding** *lem3*[*OF nz*] *vector-component-simps* ∗ *mem-interval*
apply(*erule-tac x=2* **in** *allE*) **by** *auto* **qed**(*auto*) **qed**

lemma *fashoda-unit-path*: **fixes** *f* ::*real* ⇒ *real^2* **and** *g* ::*real* ⇒ *real^2*
assumes *path f path g path-image f* ⊆ {− *1..1*} *path-image g* ⊆ {− *1..1*}
(*pathstart f*)$*1* = −*1* (*pathfinish f*)$*1* = *1* (*pathstart g*)$*2* = −*1* (*pathfinish g*)$*2* = *1*
obtains *z* **where** *z* ∈ *path-image f z* ∈ *path-image g* **proof**−
note *assms*=*assms*[*unfolded path-def pathstart-def pathfinish-def path-image-def*]
def *iscale* ≡ λ*z*::*real*. *inverse 2* ∗*R* (*z* + *1*)
have *isc*:*iscale* ‘ {− *1..1*} ⊆ {*0..1*} **unfolding** *iscale-def* **by**(*auto*)
have ∃ *s*∈{− *1..1*}. ∃ *t*∈{− *1..1*}. (*f* ∘ *iscale*) *s* = (*g* ∘ *iscale*) *t* **proof**(*rule fashoda-unit*)
show (*f* ∘ *iscale*) ‘ {− *1..1*} ⊆ {− *1..1*} (*g* ∘ *iscale*) ‘ {− *1..1*} ⊆ {− *1..1*}
using *isc* **and** *assms*(*3*−*4*) **unfolding** *image-compose* **by** *auto*
have ∗:*continuous-on* {− *1..1*} *iscale* **unfolding** *iscale-def* **by**(*rule continuous-on-intros*)+
show *continuous-on* {− *1..1*} (*f* ∘ *iscale*) *continuous-on* {− *1..1*} (*g* ∘ *iscale*)
apply−**apply**(*rule-tac*[!] *continuous-on-compose*[*OF* ∗]) **apply**(*rule-tac*[!] *continuous-on-subset*[*OF - isc*])
by(*rule assms*)+ **have** ∗:(*1 / 2*) ∗*R* (*1* + (*1*::*real^1*)) = *1* **unfolding** *Cart-eq* **by** *auto*
show (*f* ∘ *iscale*) (− *1*) $ *1* = − *1* (*f* ∘ *iscale*) *1* $ *1* = *1* (*g* ∘ *iscale*) (− *1*) $ *2* = −*1* (*g* ∘ *iscale*) *1* $ *2* = *1*
unfolding *o-def iscale-def* **using** *assms* **by**(*auto simp add*:∗) **qed**
then guess *s* **..** **from** *this*(*2*) **guess** *t* **..** **note** *st*=*this*
show *thesis* **apply**(*rule-tac z=f* (*iscale s*) **in** *that*)
using *st* ‹*s*∈{− *1..1*}› **unfolding** *o-def path-image-def image-iff* **apply**−
apply(*rule-tac x=iscale s* **in** *bexI*) **prefer** *3* **apply**(*rule-tac x=iscale t* **in** *bexI*)
using *isc*[*unfolded subset-eq*, *rule-format*] **by** *auto* **qed**

lemma *fashoda*: **fixes** *b*::*real^2*
assumes *path f path g path-image f* ⊆ {*a..b*} *path-image g* ⊆ {*a..b*}
(*pathstart f*)$*1* = *a*$*1* (*pathfinish f*)$*1* = *b*$*1*
(*pathstart g*)$*2* = *a*$*2* (*pathfinish g*)$*2* = *b*$*2*
obtains *z* **where** *z* ∈ *path-image f z* ∈ *path-image g* **proof**−
fix *P Q S* **presume** *P* ∨ *Q* ∨ *S P* ⟹ *thesis Q* ⟹ *thesis S* ⟹ *thesis* **thus** *thesis* **by** *auto*
next have {*a..b*} ≠ {} **using** *assms*(*3*) **using** *path-image-nonempty* **by** *auto*
hence *a* ≤ *b* **unfolding** *interval-eq-empty vector-le-def* **by**(*auto simp add*: *not-less*)
thus *a*$*1* = *b*$*1* ∨ *a*$*2* = *b*$*2* ∨ (*a*$*1* < *b*$*1* ∧ *a*$*2* < *b*$*2*) **unfolding** *vector-le-def forall-2* **by** *auto*
next assume *as*:*a*$*1* = *b*$*1* **have** ∃ *z*∈*path-image g*. *z*$*2* = (*pathstart f*)$*2* **apply**(*rule connected-ivt-component*)
apply(*rule connected-path-image assms*)+**apply**(*rule pathstart-in-path-image*,*rule pathfinish-in-path-image*)
unfolding *assms* **using** *assms*(*3*)[*unfolded path-image-def subset-eq*,*rule-format*,*of f 0*]
unfolding *pathstart-def* **by**(*auto simp add*: *vector-le-def*) **then guess** *z* **..** **note**

z=this
have *z* ∈ {*a..b*} **using** *z(1) assms(4)* **unfolding** *path-image-def* **by** *blast*
hence *z* = *f 0* **unfolding** *Cart-eq forall-2* **unfolding** *z(2) pathstart-def*
using *assms(3)*[*unfolded path-image-def subset-eq mem-interval*,*rule-format*,*of f 0 1*]
unfolding *mem-interval* **apply**(*erule-tac x=1* **in** *allE*) **using** *as* **by** *auto*
thus *thesis* **apply**−**apply**(*rule that*[*OF - z(1)*]) **unfolding** *path-image-def* **by** *auto*
next assume *as*:*a*$*2* = *b*$*2* **have** ∃*z*∈*path-image f*. *z*$*1* = (*pathstart g*)$*1* **apply**(*rule connected-ivt-component*)
apply(*rule connected-path-image assms*)+**apply**(*rule pathstart-in-path-image*,*rule pathfinish-in-path-image*)
unfolding *assms* **using** *assms(4)*[*unfolded path-image-def subset-eq*,*rule-format*,*of g 0*]
unfolding *pathstart-def* **by**(*auto simp add*: *vector-le-def*) **then guess** *z* **.. note** *z=this*
have *z* ∈ {*a..b*} **using** *z(1) assms(3)* **unfolding** *path-image-def* **by** *blast*
hence *z* = *g 0* **unfolding** *Cart-eq forall-2* **unfolding** *z(2) pathstart-def*
using *assms(4)*[*unfolded path-image-def subset-eq mem-interval*,*rule-format*,*of g 0 2*]
unfolding *mem-interval* **apply**(*erule-tac x=2* **in** *allE*) **using** *as* **by** *auto*
thus *thesis* **apply**−**apply**(*rule that*[*OF z(1)*]) **unfolding** *path-image-def* **by** *auto*
next assume *as*:*a* $ *1* < *b* $ *1* ∧ *a* $ *2* < *b* $ *2*
have *int-nem*:{− *1..1*::*real*^*2*} ≠ {} **unfolding** *interval-eq-empty* **by** *auto*
guess *z* **apply**(*rule fashoda-unit-path*[*of interval-bij* (*a*,*b*) (− *1*,*1*) ∘ *f interval-bij* (*a*,*b*) (− *1*,*1*) ∘ *g*])
unfolding *path-def path-image-def pathstart-def pathfinish-def*
apply(*rule-tac*[*1*−*2*] *continuous-on-compose*) **apply**(*rule assms*[*unfolded path-def*] *continuous-on-interval-bij*)+
unfolding *subset-eq* **apply**(*rule-tac*[*1*−*2*] *ballI*)
proof− **fix** *x* **assume** *x* ∈ (*interval-bij* (*a*, *b*) (− *1*, *1*) ∘ *f*) ‘ {*0..1*}
then guess *y* **unfolding** *image-iff* **.. note** *y=this*
show *x* ∈ {− *1..1*} **unfolding** *y o-def* **apply**(*rule in-interval-interval-bij*)
using *y(1)* **using** *assms(3)*[*unfolded path-image-def subset-eq*] *int-nem* **by** *auto*
next fix *x* **assume** *x* ∈ (*interval-bij* (*a*, *b*) (− *1*, *1*) ∘ *g*) ‘ {*0..1*}
then guess *y* **unfolding** *image-iff* **.. note** *y=this*
show *x* ∈ {− *1..1*} **unfolding** *y o-def* **apply**(*rule in-interval-interval-bij*)
using *y(1)* **using** *assms(4)*[*unfolded path-image-def subset-eq*] *int-nem* **by** *auto*
next show (*interval-bij* (*a*, *b*) (− *1*, *1*) ∘ *f*) *0* $ *1* = −*1*
(*interval-bij* (*a*, *b*) (− *1*, *1*) ∘ *f*) *1* $ *1* = *1*
(*interval-bij* (*a*, *b*) (− *1*, *1*) ∘ *g*) *0* $ *2* = −*1*
(*interval-bij* (*a*, *b*) (− *1*, *1*) ∘ *g*) *1* $ *2* = *1* **unfolding** *interval-bij-def Cart-lambda-beta vector-component-simps o-def split-conv*
unfolding *assms*[*unfolded pathstart-def pathfinish-def*] **using** *as* **by** *auto* **qed**
note *z=this*
from *z(1)* **guess** *zf* **unfolding** *image-iff* **.. note** *zf=this*

from *z(2)* **guess** *zg* **unfolding** *image-iff* **..** **note** *zg=this*
have *∗*:*∀ i. (− 1) $ i < (1::real^2) $ i ∧ a $ i < b $ i* **unfolding** *forall-2* **using** *as* **by** *auto*
show *thesis* **apply**(*rule-tac z=interval-bij (− 1,1) (a,b) z* **in** *that*)
apply(*subst zf*) **defer** **apply**(*subst zg*) **unfolding** *o-def interval-bij-bij*[*OF ∗*] *path-image-def*
using *zf(1) zg(1)* **by** *auto* **qed**

26.2 Some slightly ad hoc lemmas I use below

lemma *segment-vertical*: **fixes** *a::real^2* **assumes** *a$1 = b$1*
shows *x ∈ closed-segment a b ⟷ (x$1 = a$1 ∧ x$1 = b$1 ∧*
(a$2 ≤ x$2 ∧ x$2 ≤ b$2 ∨ b$2 ≤ x$2 ∧ x$2 ≤ a$2)) (**is** *- = ?R*)
proof−
let *?L = ∃ u. (x $ 1 = (1 − u) ∗ a $ 1 + u ∗ b $ 1 ∧ x $ 2 = (1 − u) ∗ a $ 2 + u ∗ b $ 2) ∧ 0 ≤ u ∧ u ≤ 1*
{ presume *?L ⟹ ?R ?R ⟹ ?L* **thus** *?thesis* **unfolding** *closed-segment-def mem-Collect-eq*
unfolding *Cart-eq forall-2 smult-conv-scaleR*[*THEN sym*] *vector-component-simps* **by** *blast* **}**
{ assume *?L* **then guess** *u* **apply**−**apply**(*erule exE*)**apply**(*erule conjE*)+ **.** **note** *u=this*
{ fix *b a* **assume** *b + u ∗ a > a + u ∗ b*
hence *(1 − u) ∗ b > (1 − u) ∗ a* **by**(*auto simp add:field-simps*)
hence *b ≥ a* **apply**(*drule-tac mult-less-imp-less-left*) **using** *u* **by** *auto*
hence *u ∗ a ≤ u ∗ b* **apply**−**apply**(*rule mult-left-mono*[*OF - u(3)*])
using *u(3−4)* **by**(*auto simp add:field-simps*) **}** **note** *∗ = this*
{ fix *a b* **assume** *u ∗ b > u ∗ a* **hence** *(1 − u) ∗ a ≤ (1 − u) ∗ b* **apply**−**apply**(*rule mult-left-mono*)
apply(*drule mult-less-imp-less-left*) **using** *u* **by** *auto*
hence *a + u ∗ b ≤ b + u ∗ a* **by**(*auto simp add:field-simps*) **}** **note** *∗∗ = this*
thus *?R* **unfolding** *u assms* **using** *u* **by**(*auto simp add:field-simps not-le intro:∗ ∗∗*) **}**
{ assume *?R* **thus** *?L* **proof**(*cases x$2 = b$2*)
case *True* **thus** *?L* **apply**(*rule-tac x=(x$2 − a$2) / (b$2 − a$2)* **in** *exI*) **unfolding** *assms True*
using ‹*?R*› **by**(*auto simp add:field-simps*)
next case *False* **thus** *?L* **apply**(*rule-tac x=1 − (x$2 − b$2) / (a$2 − b$2)* **in** *exI*) **unfolding** *assms* **using** ‹*?R*›
by(*auto simp add:field-simps*)
qed **}** **qed**

lemma *segment-horizontal*: **fixes** *a::real^2* **assumes** *a$2 = b$2*
shows *x ∈ closed-segment a b ⟷ (x$2 = a$2 ∧ x$2 = b$2 ∧*
(a$1 ≤ x$1 ∧ x$1 ≤ b$1 ∨ b$1 ≤ x$1 ∧ x$1 ≤ a$1)) (**is** *- = ?R*)
proof−
let *?L = ∃ u. (x $ 1 = (1 − u) ∗ a $ 1 + u ∗ b $ 1 ∧ x $ 2 = (1 − u) ∗ a $ 2 + u ∗ b $ 2) ∧ 0 ≤ u ∧ u ≤ 1*

{ presume *?L* $\Longrightarrow$ *?R* *?R* $\Longrightarrow$ *?L* **thus** *?thesis* **unfolding** *closed-segment-def mem-Collect-eq*
unfolding *Cart-eq forall-2 smult-conv-scaleR*[*THEN sym*] *vector-component-simps* **by** *blast* **}**
{ assume *?L* **then guess** *u* **apply−apply**(*erule exE*)**apply**(*erule conjE*)+ **.**
note *u=this*
{ fix *b a* **assume** $b + u * a > a + u * b$
hence $(1 - u) * b > (1 - u) * a$ **by**(*auto simp add:field-simps*)
hence $b \geq a$ **apply**(*drule-tac mult-less-imp-less-left*) **using** *u* **by** *auto*
hence $u * a \leq u * b$ **apply−apply**(*rule mult-left-mono*[*OF - u*(*3*)])
using *u*(*3−4*) **by**(*auto simp add:field-simps*) **}** **note** $* = this$
{ fix *a b* **assume** $u * b > u * a$ **hence** $(1 - u) * a \leq (1 - u) * b$ **apply−apply**(*rule mult-left-mono*)
apply(*drule mult-less-imp-less-left*) **using** *u* **by** *auto*
hence $a + u * b \leq b + u * a$ **by**(*auto simp add:field-simps*) **}** **note** $** = this$
thus *?R* **unfolding** *u assms* **using** *u* **by**(*auto simp add:field-simps not-le intro*:$* **$) **}**
{ assume *?R* **thus** *?L* **proof**(*cases* $x\$1 = b\$1$)
case *True* **thus** *?L* **apply**(*rule-tac* $x=(x\$1 - a\$1) / (b\$1 - a\$1)$ **in** *exI*) **unfolding** *assms True*
using ‹*?R*› **by**(*auto simp add:field-simps*)
next case *False* **thus** *?L* **apply**(*rule-tac* $x=1 - (x\$1 - b\$1) / (a\$1 - b\$1)$ **in** *exI*) **unfolding** *assms* **using** ‹*?R*›
by(*auto simp add:field-simps*)
qed } qed

26.3 useful Fashoda corollary pointed out to me by Tom Hales.

lemma *fashoda-interlace*: **fixes** *a*::*real^2*
assumes *path f path g*
path-image f $\subseteq$ {*a..b*} *path-image g* $\subseteq$ {*a..b*}
(*pathstart f*)$2 = a$2 (*pathfinish f*)$2 = a$2
(*pathstart g*)$2 = a$2 (*pathfinish g*)$2 = a$2
(*pathstart f*)$1 < (*pathstart g*)$1 (*pathstart g*)$1 < (*pathfinish f*)$1
(*pathfinish f*)$1 < (*pathfinish g*)$1
obtains *z* **where** $z \in$ *path-image f* $z \in$ *path-image g*
proof−
have {*a..b*} $\neq$ {} **using** *path-image-nonempty* **using** *assms*(*3*) **by** *auto*
note *ab=this*[*unfolded interval-eq-empty not-ex forall-2 not-less*]
have *pathstart f* $\in$ {*a..b*} *pathfinish f* $\in$ {*a..b*} *pathstart g* $\in$ {*a..b*} *pathfinish g* $\in$ {*a..b*}
using *pathstart-in-path-image pathfinish-in-path-image* **using** *assms*(*3−4*) **by** *auto*
note *startfin = this*[*unfolded mem-interval forall-2*]
let *?P1 = linepath* (*vector*[*a*$1 − 2, *a*$2 − 2]) (*vector*[(*pathstart f*)$1,*a*$2 − 2]) +++
linepath(*vector*[(*pathstart f*)$1,*a*$2 − 2])(*pathstart f*) +++ *f* +++

linepath(*pathfinish f*)(*vector*[(*pathfinish f*)$*1*,*a*$*2 − 2*]) +++
linepath(*vector*[(*pathfinish f*)$*1*,*a*$*2 − 2*])(*vector*[*b*$*1 + 2*,*a*$*2 − 2*])
let *?P2 = linepath*(*vector*[(*pathstart g*)$*1*, (*pathstart g*)$*2 − 3*])(*pathstart g*) +++ *g* +++
linepath(*pathfinish g*)(*vector*[(*pathfinish g*)$*1*,*a*$*2 − 1*]) +++
linepath(*vector*[(*pathfinish g*)$*1*,*a*$*2 − 1*])(*vector*[*b*$*1 + 1*,*a*$*2 − 1*]) +++
linepath(*vector*[*b*$*1 + 1*,*a*$*2 − 1*])(*vector*[*b*$*1 + 1*,*b*$*2 + 3*])
let *?a = vector*[*a*$*1 − 2*, *a*$*2 − 3*]
let *?b = vector*[*b*$*1 + 2*, *b*$*2 + 3*]
have *P1P2*:*path-image ?P1 = path-image* (*linepath* (*vector*[*a*$*1 − 2*, *a*$*2 − 2*]) (*vector*[(*pathstart f*)$*1*,*a*$*2 − 2*])) ∪
path-image (*linepath*(*vector*[(*pathstart f*)$*1*,*a*$*2 − 2*])(*pathstart f*)) ∪ *path-image f* ∪
path-image (*linepath*(*pathfinish f*)(*vector*[(*pathfinish f*)$*1*,*a*$*2 − 2*])) ∪
path-image (*linepath*(*vector*[(*pathfinish f*)$*1*,*a*$*2 − 2*])(*vector*[*b*$*1 + 2*,*a*$*2 − 2*]))
path-image ?P2 = path-image(*linepath*(*vector*[(*pathstart g*)$*1*, (*pathstart g*)$*2 − 3*])(*pathstart g*)) ∪ *path-image g* ∪
path-image(*linepath*(*pathfinish g*)(*vector*[(*pathfinish g*)$*1*,*a*$*2 − 1*])) ∪
path-image(*linepath*(*vector*[(*pathfinish g*)$*1*,*a*$*2 − 1*])(*vector*[*b*$*1 + 1*,*a*$*2 − 1*])) ∪
path-image(*linepath*(*vector*[*b*$*1 + 1*,*a*$*2 − 1*])(*vector*[*b*$*1 + 1*,*b*$*2 + 3*]))
using *assms*(*1−2*)
by(*auto simp add*: *path-image-join path-linepath*)
have *abab*: {*a*..*b*} ⊆ {*?a*..*?b*} **by**(*auto simp add*:*vector-le-def forall-2 vector-2*)
guess *z* **apply**(*rule fashoda*[*of ?P1 ?P2 ?a ?b*])
unfolding *pathstart-join pathfinish-join pathstart-linepath pathfinish-linepath vector-2* **proof**−
show *path ?P1 path ?P2* **using** *assms* **by** *auto*
have *path-image ?P1* ⊆ {*?a .. ?b*} **unfolding** *P1P2 path-image-linepath* **apply**(*rule Un-least*)+ **defer** *3*
apply(*rule-tac*[*1−4*] *convex-interval*(*1*)[*unfolded convex-contains-segment*,*rule-format*])
unfolding *mem-interval forall-2 vector-2* **using** *ab startfin abab assms*(*3*)
using *assms*(*9−*) **unfolding** *assms* **by**(*auto simp add*:*field-simps*)
thus *path-image ?P1* ⊆ {*?a .. ?b*} **.**
have *path-image ?P2* ⊆ {*?a .. ?b*} **unfolding** *P1P2 path-image-linepath* **apply**(*rule Un-least*)+ **defer** *2*
apply(*rule-tac*[*1−4*] *convex-interval*(*1*)[*unfolded convex-contains-segment*,*rule-format*])
unfolding *mem-interval forall-2 vector-2* **using** *ab startfin abab assms*(*4*)
using *assms*(*9−*) **unfolding** *assms* **by**(*auto simp add*:*field-simps*)
thus *path-image ?P2* ⊆ {*?a .. ?b*} **.**
show *a* $ *1 − 2 = a* $ *1 − 2 b* $ *1 + 2 = b* $ *1 + 2 pathstart g* $ *2 − 3 = a* $ *2 − 3 b* $ *2 + 3 = b* $ *2 + 3*
by(*auto simp add*: *assms*)
qed note *z*=*this*[*unfolded P1P2 path-image-linepath*]
show *thesis* **apply**(*rule that*[*of z*]) **proof**−
have (*z* ∈ *closed-segment* (*vector* [*a* $ *1 − 2*, *a* $ *2 − 2*]) (*vector* [*pathstart f* $ *1*, *a* $ *2 − 2*]) ∨
z ∈ *closed-segment* (*vector* [*pathstart f* $ *1*, *a* $ *2 − 2*]) (*pathstart f*)) ∨

z ∈ *closed-segment* (*pathfinish f*) (*vector* [*pathfinish f* $ *1*, *a* $ *2* − *2*]) ∨
z ∈ *closed-segment* (*vector* [*pathfinish f* $ *1*, *a* $ *2* − *2*]) (*vector* [*b* $ *1* + *2*, *a* $ *2* − *2*]) ⟹
(((*z* ∈ *closed-segment* (*vector* [*pathstart g* $ *1*, *pathstart g* $ *2* − *3*]) (*pathstart g*)) ∨
z ∈ *closed-segment* (*pathfinish g*) (*vector* [*pathfinish g* $ *1*, *a* $ *2* − *1*])) ∨
z ∈ *closed-segment* (*vector* [*pathfinish g* $ *1*, *a* $ *2* − *1*]) (*vector* [*b* $ *1* + *1*, *a* $ *2* − *1*])) ∨
z ∈ *closed-segment* (*vector* [*b* $ *1* + *1*, *a* $ *2* − *1*]) (*vector* [*b* $ *1* + *1*, *b* $ *2* + *3*]) ⟹ *False*
apply(*simp only*: *segment-vertical segment-horizontal vector-2*) **proof**− **case** *goal1* **note** *as*=*this*
have *pathfinish f* ∈ {*a*..*b*} **using** *assms*(*3*) *pathfinish-in-path-image*[*of f*] **by** *auto*
hence *1* + *b* $ *1* ≤ *pathfinish f* $ *1* ⟹ *False* **unfolding** *mem-interval forall-2* **by** *auto*
hence *z*$*1* ≠ *pathfinish f*$*1* **using** *as*(*2*) **using** *assms ab* **by**(*auto simp add*:*field-simps*)
moreover have *pathstart f* ∈ {*a*..*b*} **using** *assms*(*3*) *pathstart-in-path-image*[*of f*] **by** *auto*
hence *1* + *b* $ *1* ≤ *pathstart f* $ *1* ⟹ *False* **unfolding** *mem-interval forall-2* **by** *auto*
hence *z*$*1* ≠ *pathstart f*$*1* **using** *as*(*2*) **using** *assms ab* **by**(*auto simp add*:*field-simps*)
ultimately have ∗:*z*$*2* = *a*$*2* − *2* **using** *goal1*(*1*) **by** *auto*
have *z*$*1* ≠ *pathfinish g*$*1* **using** *as*(*2*) **using** *assms ab* **by**(*auto simp add*:*field-simps* ∗)
moreover have *pathstart g* ∈ {*a*..*b*} **using** *assms*(*4*) *pathstart-in-path-image*[*of g*] **by** *auto*
note *this*[*unfolded mem-interval forall-2*]
hence *z*$*1* ≠ *pathstart g*$*1* **using** *as*(*1*) **using** *assms ab* **by**(*auto simp add*:*field-simps* ∗)
ultimately have *a* $ *2* − *1* ≤ *z* $ *2* ∧ *z* $ *2* ≤ *b* $ *2* + *3* ∨ *b* $ *2* + *3* ≤ *z* $ *2* ∧ *z* $ *2* ≤ *a* $ *2* − *1*
using *as*(*2*) **unfolding** ∗ *assms* **by**(*auto simp add*:*field-simps*)
thus *False* **unfolding** ∗ **using** *ab* **by** *auto*
qed hence *z* ∈ *path-image f* ∨ *z* ∈ *path-image g* **using** *z* **unfolding** *Un-iff* **by** *blast*
hence *z*′:*z*∈{*a*..*b*} **using** *assms*(*3*−*4*) **by** *auto*
have *a* $ *2* = *z* $ *2* ⟹ (*z* $ *1* = *pathstart f* $ *1* ∨ *z* $ *1* = *pathfinish f* $ *1*) ⟹ (*z* = *pathstart f* ∨ *z* = *pathfinish f*)
unfolding *Cart-eq forall-2 assms* **by** *auto*
with *z*′ **show** *z*∈*path-image f* **using** *z*(*1*) **unfolding** *Un-iff mem-interval forall-2* **apply**−
apply(*simp only*: *segment-vertical segment-horizontal vector-2*) **unfolding** *assms* **by** *auto*
have *a* $ *2* = *z* $ *2* ⟹ (*z* $ *1* = *pathstart g* $ *1* ∨ *z* $ *1* = *pathfinish g* $ *1*) ⟹ (*z* = *pathstart g* ∨ *z* = *pathfinish g*)
unfolding *Cart-eq forall-2 assms* **by** *auto*

with z' **show** $z \in$*path-image g* **using** $z(2)$ **unfolding** *Un-iff mem-interval forall-2* **apply**−
apply(*simp only*: *segment-vertical segment-horizontal vector-2*) **unfolding** *assms* **by** *auto*
qed qed

end

theory *Multivariate-Analysis*
imports *Determinants Integration Real-Integration Fashoda*
begin

end