Some results of number theory

Jeremy Avigad
David Gray
Adam Kramer
Thomas M Rasmussen

June 21, 2010

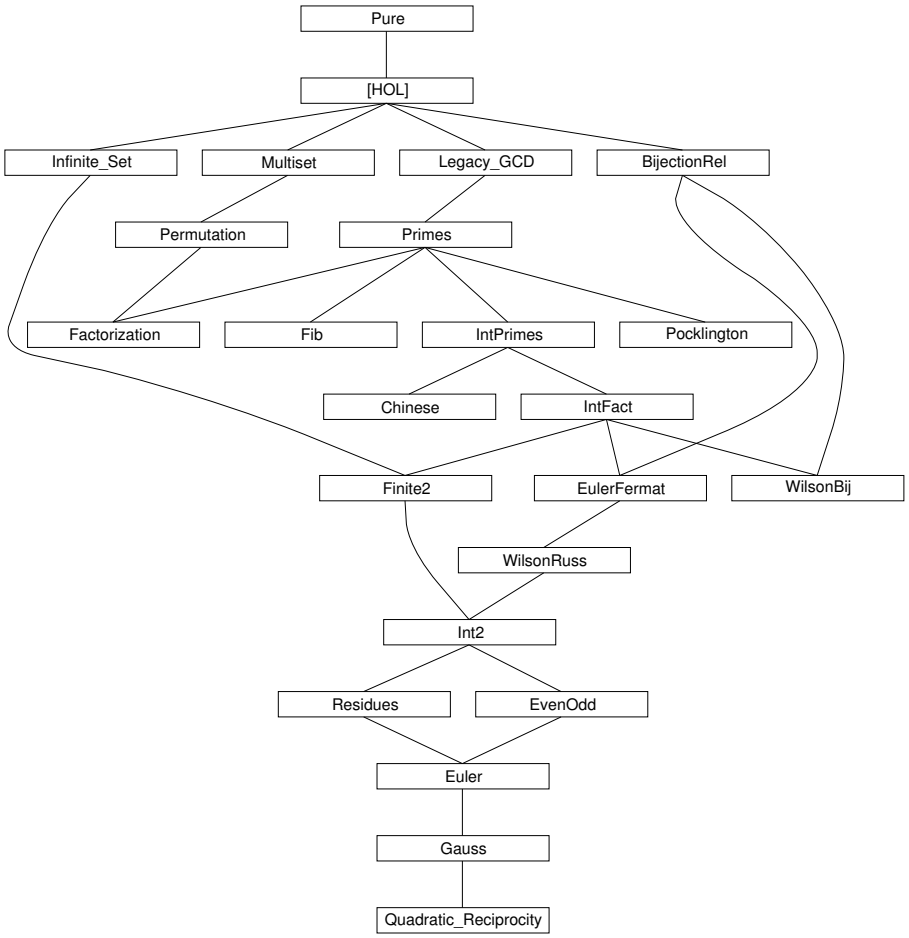
Abstract

This is a collection of formalized proofs of many results of number theory. The proofs of the Chinese Remainder Theorem and Wilson's Theorem are due to Rasmussen. The proof of Gauss's law of quadratic reciprocity is due to Avigad, Gray and Kramer. Proofs can be found in most introductory number theory textbooks; Goldman's *The Queen of Mathematics: a Historically Motivated Guide to Number Theory* provides some historical context.

Avigad, Gray and Kramer have also provided library theories dealing with finite sets and finite sums, divisibility and congruences, parity and residues. The authors are engaged in redesigning and polishing these theories for more serious use. For the latest information in this respect, please see the web page http://www.andrew.cmu.edu/~avigad/isabelle. Other theories contain proofs of Euler's criteria, Gauss' lemma, and the law of quadratic reciprocity. The formalization follows Eisenstein's proof, which is the one most commonly found in introductory textbooks; in particular, it follows the presentation in Niven and Zuckerman, *The Theory of Numbers*.

To avoid having to count roots of polynomials, however, we relied on a trick previously used by David Russinoff in formalizing quadratic reciprocity for the Boyer-Moore theorem prover; see Russinoff, David, "A mechanical proof of quadratic reciprocity," *Journal of Automated Reasoning* 8:3-21, 1992. We are grateful to Larry Paulson for calling our attention to this reference.

Contents


Pure
[HOL]
Infinite_Set
Multiset
Legacy_GCD
BijectionRel
Permutation
Primes
Factorization
Fib
IntPrimes
Pocklington
Chinese
IntFact
Finite2
EulerFermat
WilsonBij
WilsonRuss
Int2
Residues
EvenOdd
Euler
Gauss
Quadratic_Reciprocity

1 The Greatest Common Divisor

theory *Legacy-GCD*
imports *Main*
begin

See [**?**].

1.1 Specification of GCD on nats

definition
is-gcd :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where** — *gcd* as a relation
[*code del*]: *is-gcd m n p* ⟷ *p dvd m* ∧ *p dvd n* ∧
(∀ *d*. *d dvd m* ⟶ *d dvd n* ⟶ *d dvd p*)

Uniqueness

lemma *is-gcd-unique*: *is-gcd a b m* ⟹ *is-gcd a b n* ⟹ *m* = *n*
⟨*proof*⟩

Connection to divides relation

lemma *is-gcd-dvd*: *is-gcd a b m* ⟹ *k dvd a* ⟹ *k dvd b* ⟹ *k dvd m*
⟨*proof*⟩

Commutativity

lemma *is-gcd-commute*: *is-gcd m n k* = *is-gcd n m k*
⟨*proof*⟩

1.2 GCD on nat by Euclid's algorithm

fun
gcd :: *nat* => *nat* => *nat*
where
gcd m n = (*if n* = *0 then m else gcd n* (*m mod n*))
lemma *gcd-induct* [*case-names 0 rec*]:
fixes *m n* :: *nat*
assumes ⋀*m*. *P m 0*
and ⋀*m n*. *0* < *n* ⟹ *P n* (*m mod n*) ⟹ *P m n*
shows *P m n*
⟨*proof*⟩

lemma *gcd-0* [*simp*, *algebra*]: *gcd m 0* = *m*
⟨*proof*⟩

lemma *gcd-0-left* [*simp*,*algebra*]: *gcd 0 m* = *m*
⟨*proof*⟩

lemma *gcd-non-0*: *n* > *0* ⟹ *gcd m n* = *gcd n* (*m mod n*)
⟨*proof*⟩

lemma *gcd-1* [*simp*, *algebra*]: *gcd m* (*Suc 0*) = *Suc 0*
⟨*proof*⟩

lemma *nat-gcd-1-right* [*simp*, *algebra*]: *gcd m 1* = *1*
⟨*proof*⟩

declare *gcd.simps* [*simp del*]

gcd m n divides *m* and *n*. The conjunctions don't seem provable separately.

lemma *gcd-dvd1* [*iff*, *algebra*]: *gcd m n dvd m*
and *gcd-dvd2* [*iff*, *algebra*]: *gcd m n dvd n*
⟨*proof*⟩

Maximality: for all *m*, *n*, *k* naturals, if *k* divides *m* and *k* divides *n* then *k* divides *gcd m n*.

lemma *gcd-greatest*: *k dvd m* ⟹ *k dvd n* ⟹ *k dvd gcd m n*
⟨*proof*⟩

Function gcd yields the Greatest Common Divisor.

lemma *is-gcd*: *is-gcd m n* (*gcd m n*)
⟨*proof*⟩

1.3 Derived laws for GCD

lemma *gcd-greatest-iff* [*iff*, *algebra*]: *k dvd gcd m n* ⟷ *k dvd m* ∧ *k dvd n*
⟨*proof*⟩

lemma *gcd-zero*[*algebra*]: *gcd m n* = *0* ⟷ *m* = *0* ∧ *n* = *0*
⟨*proof*⟩

lemma *gcd-commute*: *gcd m n* = *gcd n m*
⟨*proof*⟩

lemma *gcd-assoc*: *gcd* (*gcd k m*) *n* = *gcd k* (*gcd m n*)
⟨*proof*⟩

lemma *gcd-1-left* [*simp*, *algebra*]: *gcd* (*Suc 0*) *m* = *Suc 0*
⟨*proof*⟩

lemma *nat-gcd-1-left* [*simp*, *algebra*]: *gcd 1 m* = *1*
⟨*proof*⟩

Multiplication laws

lemma *gcd-mult-distrib2*: *k* ∗ *gcd m n* = *gcd* (*k* ∗ *m*) (*k* ∗ *n*)
— [**?**, page 27]

⟨*proof*⟩

lemma *gcd-mult* [*simp*, *algebra*]: *gcd k* (*k* ∗ *n*) = *k*
⟨*proof*⟩

lemma *gcd-self* [*simp*, *algebra*]: *gcd k k* = *k*
⟨*proof*⟩

lemma *relprime-dvd-mult*: *gcd k n* = *1* ==> *k dvd m* ∗ *n* ==> *k dvd m*
⟨*proof*⟩

lemma *relprime-dvd-mult-iff*: *gcd k n* = *1* ==> (*k dvd m* ∗ *n*) = (*k dvd m*)
⟨*proof*⟩

lemma *gcd-mult-cancel*: *gcd k n* = *1* ==> *gcd* (*k* ∗ *m*) *n* = *gcd m n*
⟨*proof*⟩

Addition laws

lemma *gcd-add1* [*simp*, *algebra*]: *gcd* (*m* + *n*) *n* = *gcd m n*
⟨*proof*⟩

lemma *gcd-add2* [*simp*, *algebra*]: *gcd m* (*m* + *n*) = *gcd m n*
⟨*proof*⟩

lemma *gcd-add2′* [*simp*, *algebra*]: *gcd m* (*n* + *m*) = *gcd m n*
⟨*proof*⟩

lemma *gcd-add-mult*[*algebra*]: *gcd m* (*k* ∗ *m* + *n*) = *gcd m n*
⟨*proof*⟩

lemma *gcd-dvd-prod*: *gcd m n dvd m* ∗ *n*
⟨*proof*⟩

Division by gcd yields rrelatively primes.

lemma *div-gcd-relprime*:
assumes *nz*: *a* ≠ *0* ∨ *b* ≠ *0*
shows *gcd* (*a div gcd a b*) (*b div gcd a b*) = *1*
⟨*proof*⟩

lemma *gcd-unique*: *d dvd a*∧*d dvd b* ∧ (∀ *e*. *e dvd a* ∧ *e dvd b* ⟶ *e dvd d*) ⟷ *d* = *gcd a b*
⟨*proof*⟩

lemma *gcd-eq*: **assumes** *H*: ∀ *d*. *d dvd x* ∧ *d dvd y* ⟷ *d dvd u* ∧ *d dvd v*
shows *gcd x y* = *gcd u v*
⟨*proof*⟩

lemma *ind-euclid*:
assumes *c*: $\forall a\ b.\ P\ (a::nat)\ b \longleftrightarrow P\ b\ a$ **and** *z*: $\forall a.\ P\ a\ 0$
and *add*: $\forall a\ b.\ P\ a\ b \longrightarrow P\ a\ (a + b)$
shows $P\ a\ b$
⟨*proof*⟩

lemma *bezout-lemma*:
assumes *ex*: $\exists (d::nat)\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ b \wedge (a * x = b * y + d \vee b * x = a * y + d)$
shows $\exists d\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ a + b \wedge (a * x = (a + b) * y + d \vee (a + b) * x = a * y + d)$
⟨*proof*⟩

lemma *bezout-add*: $\exists (d::nat)\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ b \wedge (a * x = b * y + d \vee b * x = a * y + d)$
⟨*proof*⟩

lemma *bezout*: $\exists (d::nat)\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ b \wedge (a * x - b * y = d \vee b * x - a * y = d)$
⟨*proof*⟩

We can get a stronger version with a nonzeroness assumption.

lemma *divides-le*: $m\ dvd\ n ==> m <= n \vee n = (0::nat)$ ⟨*proof*⟩

lemma *bezout-add-strong*: **assumes** *nz*: $a \neq (0::nat)$
shows $\exists d\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ b \wedge a * x = b * y + d$
⟨*proof*⟩

lemma *bezout-gcd*: $\exists x\ y.\ a * x - b * y = gcd\ a\ b \vee b * x - a * y = gcd\ a\ b$
⟨*proof*⟩

lemma *bezout-gcd-strong*: **assumes** *a*: $a \neq 0$
shows $\exists x\ y.\ a * x = b * y + gcd\ a\ b$
⟨*proof*⟩

lemma *gcd-mult-distrib*: $gcd(a * c)\ (b * c) = c * gcd\ a\ b$
⟨*proof*⟩

lemma *gcd-bezout*: $(\exists x\ y.\ a * x - b * y = d \vee b * x - a * y = d) \longleftrightarrow gcd\ a\ b\ dvd\ d$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

lemma *gcd-bezout-sum*: **assumes** *H*:$a * x + b * y = d$ **shows** $gcd\ a\ b\ dvd\ d$
⟨*proof*⟩

lemma *gcd-mult'*: $gcd\ b\ (a * b) = b$
⟨*proof*⟩

lemma *gcd-add*: *gcd*(*a* + *b*) *b* = *gcd a b*
gcd(*b* + *a*) *b* = *gcd a b gcd a* (*a* + *b*) = *gcd a b gcd a* (*b* + *a*) = *gcd a b*
⟨*proof*⟩

lemma *gcd-sub*: *b* <= *a* ==> *gcd*(*a* − *b*) *b* = *gcd a b a* <= *b* ==> *gcd a* (*b* − *a*) = *gcd a b*
⟨*proof*⟩

1.4 LCM defined by GCD

definition
lcm :: *nat* ⇒ *nat* ⇒ *nat*
where
lcm-def: *lcm m n* = *m* ∗ *n div gcd m n*

lemma *prod-gcd-lcm*:
m ∗ *n* = *gcd m n* ∗ *lcm m n*
⟨*proof*⟩

lemma *lcm-0* [*simp*]: *lcm m 0* = *0*
⟨*proof*⟩

lemma *lcm-1* [*simp*]: *lcm m 1* = *m*
⟨*proof*⟩

lemma *lcm-0-left* [*simp*]: *lcm 0 n* = *0*
⟨*proof*⟩

lemma *lcm-1-left* [*simp*]: *lcm 1 m* = *m*
⟨*proof*⟩

lemma *dvd-pos*:
fixes *n m* :: *nat*
assumes *n* > *0* **and** *m dvd n*
shows *m* > *0*
⟨*proof*⟩

lemma *lcm-least*:
assumes *m dvd k* **and** *n dvd k*
shows *lcm m n dvd k*
⟨*proof*⟩

lemma *lcm-dvd1* [*iff*]:
m dvd lcm m n
⟨*proof*⟩

lemma *lcm-dvd2* [*iff*]:
n dvd lcm m n

⟨*proof*⟩

lemma *gcd-add1-eq*: $gcd\ (m + k)\ k = gcd\ (m + k)\ m$
⟨*proof*⟩

lemma *gcd-diff2*: $m \leq n \Longrightarrow gcd\ n\ (n - m) = gcd\ n\ m$
⟨*proof*⟩

1.5 GCD and LCM on integers

definition
$zgcd :: int \Rightarrow int \Rightarrow int$ **where**
$zgcd\ i\ j = int\ (gcd\ (nat\ (abs\ i))\ (nat\ (abs\ j)))$

lemma *zgcd-zdvd1* [*iff*,*simp*, *algebra*]: $zgcd\ i\ j\ dvd\ i$
⟨*proof*⟩

lemma *zgcd-zdvd2* [*iff*,*simp*, *algebra*]: $zgcd\ i\ j\ dvd\ j$
⟨*proof*⟩

lemma *zgcd-pos*: $zgcd\ i\ j \geq 0$
⟨*proof*⟩

lemma *zgcd0* [*simp*,*algebra*]: $(zgcd\ i\ j = 0) = (i = 0 \wedge j = 0)$
⟨*proof*⟩

lemma *zgcd-commute*: $zgcd\ i\ j = zgcd\ j\ i$
⟨*proof*⟩

lemma *zgcd-zminus* [*simp*, *algebra*]: $zgcd\ (-\ i)\ j = zgcd\ i\ j$
⟨*proof*⟩

lemma *zgcd-zminus2* [*simp*, *algebra*]: $zgcd\ i\ (-\ j) = zgcd\ i\ j$
⟨*proof*⟩

lemma *zrelprime-dvd-mult*: $zgcd\ i\ j = 1 \Longrightarrow i\ dvd\ k * j \Longrightarrow i\ dvd\ k$
⟨*proof*⟩

lemma *int-nat-abs*: $int\ (nat\ (abs\ x)) = abs\ x$ ⟨*proof*⟩

lemma *zgcd-greatest*:
assumes $k\ dvd\ m$ **and** $k\ dvd\ n$
shows $k\ dvd\ zgcd\ m\ n$
⟨*proof*⟩

lemma *div-zgcd-relprime*:
assumes *nz*: $a \neq 0 \vee b \neq 0$
shows $zgcd\ (a\ div\ (zgcd\ a\ b))\ (b\ div\ (zgcd\ a\ b)) = 1$

⟨*proof*⟩

lemma *zgcd-0* [*simp*, *algebra*]: *zgcd m 0 = abs m*
⟨*proof*⟩

lemma *zgcd-0-left* [*simp*, *algebra*]: *zgcd 0 m = abs m*
⟨*proof*⟩

lemma *zgcd-non-0*: *0 < n ==> zgcd m n = zgcd n (m mod n)*
⟨*proof*⟩

lemma *zgcd-eq*: *zgcd m n = zgcd n (m mod n)*
⟨*proof*⟩

lemma *zgcd-1* [*simp*, *algebra*]: *zgcd m 1 = 1*
⟨*proof*⟩

lemma *zgcd-0-1-iff* [*simp*, *algebra*]: *zgcd 0 m = 1 ⟷ |m| = 1*
⟨*proof*⟩

lemma *zgcd-greatest-iff*[*algebra*]: *k dvd zgcd m n = (k dvd m ∧ k dvd n)*
⟨*proof*⟩

lemma *zgcd-1-left* [*simp*, *algebra*]: *zgcd 1 m = 1*
⟨*proof*⟩

lemma *zgcd-assoc*: *zgcd (zgcd k m) n = zgcd k (zgcd m n)*
⟨*proof*⟩

lemma *zgcd-left-commute*: *zgcd k (zgcd m n) = zgcd m (zgcd k n)*
⟨*proof*⟩

lemmas *zgcd-ac = zgcd-assoc zgcd-commute zgcd-left-commute*
— addition is an AC-operator

lemma *zgcd-zmult-distrib2*: *0 ≤ k ==> k * zgcd m n = zgcd (k * m) (k * n)*
⟨*proof*⟩

lemma *zgcd-zmult-distrib2-abs*: *zgcd (k * m) (k * n) = abs k * zgcd m n*
⟨*proof*⟩

lemma *zgcd-self* [*simp*]: *0 ≤ m ==> zgcd m m = m*
⟨*proof*⟩

lemma *zgcd-zmult-eq-self* [*simp*]: *0 ≤ k ==> zgcd k (k * n) = k*
⟨*proof*⟩

lemma *zgcd-zmult-eq-self2* [*simp*]: *0 ≤ k ==> zgcd (k * n) k = k*
⟨*proof*⟩

definition *zlcm i j = int (lcm(nat(abs i)) (nat(abs j)))*

lemma *dvd-zlcm-self1*[*simp*, *algebra*]: *i dvd zlcm i j*
⟨*proof*⟩

lemma *dvd-zlcm-self2*[*simp*, *algebra*]: *j dvd zlcm i j*
⟨*proof*⟩

lemma *dvd-imp-dvd-zlcm1*:
assumes *k dvd i* **shows** *k dvd (zlcm i j)*
⟨*proof*⟩

lemma *dvd-imp-dvd-zlcm2*:
assumes *k dvd j* **shows** *k dvd (zlcm i j)*
⟨*proof*⟩

lemma *zdvd-self-abs1*: *(d::int) dvd (abs d)*
⟨*proof*⟩

lemma *zdvd-self-abs2*: *(abs (d::int)) dvd d*
⟨*proof*⟩

lemma *lcm-pos*:
assumes *mpos*: *m > 0*
and *npos*: *n>0*
shows *lcm m n > 0*
⟨*proof*⟩

lemma *zlcm-pos*:
assumes *anz*: *a ≠ 0*
and *bnz*: *b ≠ 0*
shows *0 < zlcm a b*
⟨*proof*⟩

lemma *zgcd-code* [*code*]:
zgcd k l = |if l = 0 then k else zgcd l (|k| mod |l|)|
⟨*proof*⟩

end

2 Primality on nat

theory *Primes*
imports *Complex-Main Legacy-GCD*
begin

definition
coprime :: *nat* => *nat* => *bool* **where**
coprime m n ⟷ *gcd m n = 1*

definition
prime :: *nat* ⇒ *bool* **where**
[*code del*]: *prime p* ⟷ (*1 < p* ∧ (∀ *m*. *m dvd p* −−> *m = 1* ∨ *m = p*))

lemma *two-is-prime*: *prime 2*
⟨*proof*⟩

lemma *prime-imp-relprime*: *prime p* ==> ¬ *p dvd n* ==> *gcd p n = 1*
⟨*proof*⟩

This theorem leads immediately to a proof of the uniqueness of factorization. If p divides a product of primes then it is one of those primes.

lemma *prime-dvd-mult*: *prime p* ==> *p dvd m* ∗ *n* ==> *p dvd m* ∨ *p dvd n*
⟨*proof*⟩

lemma *prime-dvd-square*: *prime p* ==> *p dvd m*^*Suc* (*Suc 0*) ==> *p dvd m*
⟨*proof*⟩

lemma *prime-dvd-power-two*: *prime p* ==> *p dvd* m^2 ==> *p dvd m*
⟨*proof*⟩

lemma *exp-eq-1*:(*x*::*nat*)^*n = 1* ⟷ *x = 1* ∨ *n = 0*
⟨*proof*⟩

lemma *exp-mono-lt*: (*x*::*nat*) ^ (*Suc n*) < *y* ^ (*Suc n*) ⟷ *x < y*
⟨*proof*⟩

lemma *exp-mono-le*: (*x*::*nat*) ^ (*Suc n*) ≤ *y* ^ (*Suc n*) ⟷ *x* ≤ *y*
⟨*proof*⟩

lemma *exp-mono-eq*: (*x*::*nat*) ^ *Suc n = y* ^ *Suc n* ⟷ *x = y*
⟨*proof*⟩

lemma *even-square*: **assumes** *e*: *even* (*n*::*nat*) **shows** ∃ *x*. *n* ^ *2 = 4*∗*x*
⟨*proof*⟩

lemma *odd-square*: **assumes** *e*: *odd* (*n*::*nat*) **shows** ∃ *x*. *n* ^ *2* = *4*∗*x* + *1*
⟨*proof*⟩

lemma *diff-square*: (*x*::*nat*) ^*2* − *y* ^*2* = (*x*+*y*)∗(*x* − *y*)
⟨*proof*⟩

Elementary theory of divisibility

lemma *divides-ge*: (*a*::*nat*) *dvd b* ⟹ *b* = *0* ∨ *a* ≤ *b* ⟨*proof*⟩
lemma *divides-antisym*: (*x*::*nat*) *dvd y* ∧ *y dvd x* ⟷ *x* = *y*
⟨*proof*⟩

lemma *divides-add-revr*: **assumes** *da*: (*d*::*nat*) *dvd a* **and** *dab*:*d dvd* (*a* + *b*)
shows *d dvd b*
⟨*proof*⟩

declare *nat-mult-dvd-cancel-disj*[*presburger*]
lemma *nat-mult-dvd-cancel-disj* '[*presburger*]:
(*m*::*nat*)∗*k dvd n*∗*k* ⟷ *k* = *0* ∨ *m dvd n* ⟨*proof*⟩

lemma *divides-mul-l*: (*a*::*nat*) *dvd b* ==> (*c* ∗ *a*) *dvd* (*c* ∗ *b*)
⟨*proof*⟩

lemma *divides-mul-r*: (*a*::*nat*) *dvd b* ==> (*a* ∗ *c*) *dvd* (*b* ∗ *c*) ⟨*proof*⟩
lemma *divides-cases*: (*n*::*nat*) *dvd m* ==> *m* = *0* ∨ *m* = *n* ∨ *2* ∗ *n* <= *m*
⟨*proof*⟩

lemma *divides-div-not*: (*x*::*nat*) = (*q* ∗ *n*) + *r* ⟹ *0* < *r* ⟹ *r* < *n* ==> ~(*n*
dvd x)
⟨*proof*⟩
lemma *divides-exp*: (*x*::*nat*) *dvd y* ==> *x* ^ *n dvd y* ^ *n*
⟨*proof*⟩

lemma *divides-exp2*: *n* ≠ *0* ⟹ (*x*::*nat*) ^ *n dvd y* ⟹ *x dvd y*
⟨*proof*⟩

fun *fact* :: *nat* ⇒ *nat* **where**
fact 0 = *1*
| *fact* (*Suc n*) = *Suc n* ∗ *fact n*

lemma *fact-lt*: *0* < *fact n* ⟨*proof*⟩
lemma *fact-le*: *fact n* ≥ *1* ⟨*proof*⟩
lemma *fact-mono*: **assumes** *le*: *m* ≤ *n* **shows** *fact m* ≤ *fact n*
⟨*proof*⟩

lemma *divides-fact*: *1* <= *p* ⟹ *p* <= *n* ==> *p dvd fact n*
⟨*proof*⟩

declare *dvd-triv-left*[*presburger*]
declare *dvd-triv-right*[*presburger*]

lemma *divides-rexp*:
x dvd y $\Longrightarrow$ (*x*::*nat*) *dvd* (*y*^(*Suc n*)) ⟨*proof*⟩

Coprimality

lemma *coprime*: *coprime a b* $\longleftrightarrow$ ($\forall$ *d*. *d dvd a* $\wedge$ *d dvd b* $\longleftrightarrow$ *d* = *1*)
⟨*proof*⟩
lemma *coprime-commute*: *coprime a b* $\longleftrightarrow$ *coprime b a* ⟨*proof*⟩

lemma *coprime-bezout*: *coprime a b* $\longleftrightarrow$ ($\exists$ *x y*. *a* * *x* − *b* * *y* = *1* $\vee$ *b* * *x* − *a* * *y* = *1*)
⟨*proof*⟩

lemma *coprime-divprod*: *d dvd a* * *b* $\Longrightarrow$ *coprime d a* $\Longrightarrow$ *d dvd b*
⟨*proof*⟩

lemma *coprime-1*[*simp*]: *coprime a 1* ⟨*proof*⟩
lemma *coprime-1'*[*simp*]: *coprime 1 a* ⟨*proof*⟩
lemma *coprime-Suc0*[*simp*]: *coprime a* (*Suc 0*) ⟨*proof*⟩
lemma *coprime-Suc0'*[*simp*]: *coprime* (*Suc 0*) *a* ⟨*proof*⟩

lemma *gcd-coprime*:
assumes *z*: *gcd a b* ≠ *0* **and** *a*: *a* = *a'* * *gcd a b* **and** *b*: *b* = *b'* * *gcd a b*
shows *coprime a' b'*
⟨*proof*⟩
lemma *coprime-0*: *coprime d 0* $\longleftrightarrow$ *d* = *1* ⟨*proof*⟩
lemma *coprime-mul*: **assumes** *da*: *coprime d a* **and** *db*: *coprime d b*
shows *coprime d* (*a* * *b*)
⟨*proof*⟩
lemma *coprime-lmul2*: **assumes** *dab*: *coprime d* (*a* * *b*) **shows** *coprime d b*
⟨*proof*⟩

lemma *coprime-rmul2*: *coprime d* (*a* * *b*) $\Longrightarrow$ *coprime d a*
⟨*proof*⟩
lemma *coprime-mul-eq*: *coprime d* (*a* * *b*) $\longleftrightarrow$ *coprime d a* $\wedge$ *coprime d b*
⟨*proof*⟩

lemma *gcd-coprime-exists*:
assumes *nz*: *gcd a b* ≠ *0*
shows $\exists$ *a' b'*. *a* = *a'* * *gcd a b* $\wedge$ *b* = *b'* * *gcd a b* $\wedge$ *coprime a' b'*
⟨*proof*⟩

lemma *coprime-exp*: *coprime d a* ==> *coprime d* (*a*^*n*)
⟨*proof*⟩

lemma *coprime-exp-imp*: *coprime a b* ==> *coprime* (*a* ^*n*) (*b* ^*n*)
⟨*proof*⟩
lemma *coprime-refl*[*simp*]: *coprime n n* $\longleftrightarrow$ *n* = *1* ⟨*proof*⟩
lemma *coprime-plus1*[*simp*]: *coprime* (*n* + *1*) *n*
⟨*proof*⟩

lemma *coprime-minus1*: $n \neq 0 \Longrightarrow coprime\ (n - 1)\ n$
⟨*proof*⟩

lemma *bezout-gcd-pow*: $\exists x\ y.\ a\ \hat{}\ n * x - b\ \hat{}\ n * y = gcd\ a\ b\ \hat{}\ n \vee b\ \hat{}\ n * x - a\ \hat{}\ n * y = gcd\ a\ b\ \hat{}\ n$
⟨*proof*⟩

lemma *gcd-exp*: $gcd\ (a\hat{}n)\ (b\hat{}n) = gcd\ a\ b\hat{}n$
⟨*proof*⟩

lemma *coprime-exp2*: $coprime\ (a\ \hat{}\ Suc\ n)\ (b\hat{}\ Suc\ n) \longleftrightarrow coprime\ a\ b$
⟨*proof*⟩

lemma *division-decomp*: **assumes** *dc*: $(a::nat)\ dvd\ b * c$
shows $\exists b'\ c'.\ a = b' * c' \wedge b'\ dvd\ b \wedge c'\ dvd\ c$
⟨*proof*⟩

lemma *nat-power-eq-0-iff*: $(m::nat)\ \hat{}\ n = 0 \longleftrightarrow n \neq 0 \wedge m = 0$ ⟨*proof*⟩

lemma *divides-rev*: **assumes** *ab*: $(a::nat)\ \hat{}\ n\ dvd\ b\ \hat{}n$ **and** *n*:$n \neq 0$ **shows** *a dvd b*
⟨*proof*⟩

lemma *divides-mul*: **assumes** *mr*: *m dvd r* **and** *nr*: *n dvd r* **and** *mn*:*coprime m n*
shows $m * n\ dvd\ r$
⟨*proof*⟩

A binary form of the Chinese Remainder Theorem.

lemma *chinese-remainder*: **assumes** *ab*: *coprime a b* **and** *a*:$a \neq 0$ **and** *b*:$b \neq 0$
shows $\exists x\ q1\ q2.\ x = u + q1 * a \wedge x = v + q2 * b$
⟨*proof*⟩

Primality

A few useful theorems about primes

lemma *prime-0*[*simp*]: ~*prime 0* ⟨*proof*⟩
lemma *prime-1*[*simp*]: ~ *prime 1* ⟨*proof*⟩
lemma *prime-Suc0*[*simp*]: ~ *prime* (*Suc 0*) ⟨*proof*⟩

lemma *prime-ge-2*: *prime p* ==> $p \geq 2$ ⟨*proof*⟩
lemma *prime-factor*: **assumes** *n*: $n \neq 1$ **shows** $\exists\ p.\ prime\ p \wedge p\ dvd\ n$
⟨*proof*⟩

lemma *prime-factor-lt*: **assumes** *p*: *prime p* **and** *n*: $n \neq 0$ **and** *npm*:$n = p * m$
shows $m < n$
⟨*proof*⟩

lemma *euclid-bound*: $\exists p.\ prime\ p \wedge n < p \wedge\ p <= Suc\ (fact\ n)$

⟨*proof*⟩

lemma *euclid*: ∃ *p*. *prime p* ∧ *p* > *n* ⟨*proof*⟩

lemma *primes-infinite*: ¬ (*finite* {*p*. *prime p*})
⟨*proof*⟩

lemma *coprime-prime*: **assumes** *ab*: *coprime a b*
shows ~(*prime p* ∧ *p dvd a* ∧ *p dvd b*)
⟨*proof*⟩
lemma *coprime-prime-eq*: *coprime a b* ⟷ (∀ *p*. ~(*prime p* ∧ *p dvd a* ∧ *p dvd b*))

(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *prime-coprime*: **assumes** *p*: *prime p*
shows *n* = *1* ∨ *p dvd n* ∨ *coprime p n*
⟨*proof*⟩

lemma *prime-coprime-strong*: *prime p* ⟹ *p dvd n* ∨ *coprime p n*
⟨*proof*⟩

declare *coprime-0*[*simp*]

lemma *coprime-0'*[*simp*]: *coprime 0 d* ⟷ *d* = *1* ⟨*proof*⟩
lemma *coprime-bezout-strong*: **assumes** *ab*: *coprime a b* **and** *b*: *b* ≠ *1*
shows ∃ *x y*. *a* ∗ *x* = *b* ∗ *y* + *1*
⟨*proof*⟩

lemma *bezout-prime*: **assumes** *p*: *prime p* **and** *pa*: ¬ *p dvd a*
shows ∃ *x y*. *a*∗*x* = *p*∗*y* + *1*
⟨*proof*⟩
lemma *prime-divprod*: **assumes** *p*: *prime p* **and** *pab*: *p dvd a*∗*b*
shows *p dvd a* ∨ *p dvd b*
⟨*proof*⟩

lemma *prime-divprod-eq*: **assumes** *p*: *prime p*
shows *p dvd a*∗*b* ⟷ *p dvd a* ∨ *p dvd b*
⟨*proof*⟩

lemma *prime-divexp*: **assumes** *p*:*prime p* **and** *px*: *p dvd x*^*n*
shows *p dvd x*
⟨*proof*⟩

lemma *prime-divexp-n*: *prime p* ⟹ *p dvd x*^*n* ⟹ *p*^*n dvd x*^*n*
⟨*proof*⟩

lemma *coprime-prime-dvd-ex*: **assumes** *xy*: ¬*coprime x y*
shows ∃ *p*. *prime p* ∧ *p dvd x* ∧ *p dvd y*

⟨*proof*⟩
lemma *coprime-sos*: **assumes** *xy*: *coprime x y*
shows *coprime* (*x* ∗ *y*) (*x*^*2* + *y*^*2*)
⟨*proof*⟩

lemma *distinct-prime-coprime*: *prime p* ⟹ *prime q* ⟹ *p* ≠ *q* ⟹ *coprime p q*
⟨*proof*⟩

lemma *prime-coprime-lt*: **assumes** *p*: *prime p* **and** *x*: *0* < *x* **and** *xp*: *x* < *p*
shows *coprime x p*
⟨*proof*⟩

lemma *even-dvd*[*simp*]: *even* (*n*::*nat*) ⟷ *2 dvd n* ⟨*proof*⟩
lemma *prime-odd*: *prime p* ⟹ *p* = *2* ∨ *odd p* ⟨*proof*⟩

One property of coprimality is easier to prove via prime factors.

lemma *prime-divprod-pow*:
assumes *p*: *prime p* **and** *ab*: *coprime a b* **and** *pab*: *p*^*n dvd a* ∗ *b*
shows *p*^*n dvd a* ∨ *p*^*n dvd b*
⟨*proof*⟩

lemma *nat-mult-eq-one*: (*n*::*nat*) ∗ *m* = *1* ⟷ *n* = *1* ∧ *m* = *1* (**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

lemma *power-Suc0*[*simp*]: *Suc 0* ^ *n* = *Suc 0*
⟨*proof*⟩
lemma *coprime-pow*: **assumes** *ab*: *coprime a b* **and** *abcn*: *a* ∗ *b* = *c* ^*n*
shows ∃ *r s*. *a* = *r*^*n* ∧ *b* = *s* ^*n*
⟨*proof*⟩

More useful lemmas.

lemma *prime-product*:
assumes *prime* (*p* ∗ *q*)
shows *p* = *1* ∨ *q* = *1*
⟨*proof*⟩

lemma *prime-exp*: *prime* (*p*^*n*) ⟷ *prime p* ∧ *n* = *1*
⟨*proof*⟩

lemma *prime-power-mult*:
assumes *p*: *prime p* **and** *xy*: *x* ∗ *y* = *p* ^ *k*
shows ∃ *i j*. *x* = *p* ^*i* ∧ *y* = *p*^ *j*
⟨*proof*⟩

lemma *prime-power-exp*: **assumes** *p*: *prime p* **and** *n*:*n* ≠ *0*
and *xn*: *x*^*n* = *p*^*k* **shows** ∃ *i*. *x* = *p*^*i*
⟨*proof*⟩

lemma *divides-primepow*: **assumes** *p*: *prime p*
shows *d dvd p^k* $\longleftrightarrow$ $(\exists\ i.\ i \leq k \wedge d = p\ \hat{}\ i)$
⟨*proof*⟩

lemma *coprime-divisors*: *d dvd a* $\Longrightarrow$ *e dvd b* $\Longrightarrow$ *coprime a b* $\Longrightarrow$ *coprime d e*
⟨*proof*⟩

lemma *mult-inj-if-coprime-nat*:
inj-on f A $\Longrightarrow$ *inj-on g B* $\Longrightarrow$ *ALL a:A. ALL b:B. coprime* (*f a*) (*g b*)
$\Longrightarrow$ *inj-on* (%(*a*,*b*). *f a* $*$ *g b*::*nat*) (*A* $\times$ *B*)
⟨*proof*⟩

declare *power-Suc0*[*simp del*]
declare *even-dvd*[*simp del*]

end

3 The Fibonacci function

theory *Fib*
imports *Primes*
begin

Fibonacci numbers: proofs of laws taken from: R. L. Graham, D. E. Knuth, O. Patashnik. Concrete Mathematics. (Addison-Wesley, 1989)

fun *fib* :: *nat* $\Rightarrow$ *nat*
where
fib 0 = 0
| *fib* (*Suc 0*) = *1*
| *fib-2*: *fib* (*Suc* (*Suc n*)) = *fib n* + *fib* (*Suc n*)

The difficulty in these proofs is to ensure that the induction hypotheses are applied before the definition of *fib*. Towards this end, the *fib* equations are not declared to the Simplifier and are applied very selectively at first.

We disable *fib.fib-2fib-2* for simplification ...

declare *fib-2* [*simp del*]

...then prove a version that has a more restrictive pattern.

lemma *fib-Suc3*: *fib* (*Suc* (*Suc* (*Suc n*))) = *fib* (*Suc n*) + *fib* (*Suc* (*Suc n*))
⟨*proof*⟩

Concrete Mathematics, page 280

lemma *fib-add*: *fib* (*Suc* (*n* + *k*)) = *fib* (*Suc k*) $*$ *fib* (*Suc n*) + *fib k* $*$ *fib n*
⟨*proof*⟩

lemma *fib-Suc-neq-0*: *fib* (*Suc n*) ≠ *0*
⟨*proof*⟩

lemma *fib-Suc-gr-0*: *0* < *fib* (*Suc n*)
⟨*proof*⟩

lemma *fib-gr-0*: *0* < *n* ==> *0* < *fib n*
⟨*proof*⟩

Concrete Mathematics, page 278: Cassini's identity. The proof is much easier using integers, not natural numbers!

lemma *fib-Cassini-int*:
int (*fib* (*Suc* (*Suc n*)) * *fib n*) =
(*if n mod 2* = *0 then int* (*fib* (*Suc n*) * *fib* (*Suc n*)) − *1*
else int (*fib* (*Suc n*) * *fib* (*Suc n*)) + *1*)
⟨*proof*⟩

We now obtain a version for the natural numbers via the coercion function *int*.

theorem *fib-Cassini*:
fib (*Suc* (*Suc n*)) * *fib n* =
(*if n mod 2* = *0 then fib* (*Suc n*) * *fib* (*Suc n*) − *1*
else fib (*Suc n*) * *fib* (*Suc n*) + *1*)
⟨*proof*⟩

Toward Law 6.111 of Concrete Mathematics

lemma *gcd-fib-Suc-eq-1*: *gcd* (*fib n*) (*fib* (*Suc n*)) = *Suc 0*
⟨*proof*⟩

lemma *gcd-fib-add*: *gcd* (*fib m*) (*fib* (*n* + *m*)) = *gcd* (*fib m*) (*fib n*)
⟨*proof*⟩

lemma *gcd-fib-diff*: *m* ≤ *n* ==> *gcd* (*fib m*) (*fib* (*n* − *m*)) = *gcd* (*fib m*) (*fib n*)
⟨*proof*⟩

lemma *gcd-fib-mod*: *0* < *m* ==> *gcd* (*fib m*) (*fib* (*n mod m*)) = *gcd* (*fib m*) (*fib n*)
⟨*proof*⟩

lemma *fib-gcd*: *fib* (*gcd m n*) = *gcd* (*fib m*) (*fib n*) — Law 6.111
⟨*proof*⟩

theorem *fib-mult-eq-setsum*:
fib (*Suc n*) * *fib n* = (∑ *k* ∈ {..*n*}. *fib k* * *fib k*)
⟨*proof*⟩

end

4 Fundamental Theorem of Arithmetic (unique factorization into primes)

theory *Factorization*
imports *Main* ~~/*src*/*HOL*/*Old-Number-Theory*/*Primes Permutation*
begin

4.1 Definitions

definition
primel :: *nat list* => *bool* **where**
primel xs = (∀ *p* ∈ *set xs*. *prime p*)

consts
nondec :: *nat list* => *bool*
prod :: *nat list* => *nat*
oinsert :: *nat* => *nat list* => *nat list*
sort :: *nat list* => *nat list*

primrec
nondec [] = *True*
nondec (*x* # *xs*) = (*case xs of* [] => *True* | *y* # *ys* => *x* ≤ *y* ∧ *nondec xs*)

primrec
prod [] = *Suc 0*
prod (*x* # *xs*) = *x* ∗ *prod xs*

primrec
oinsert x [] = [*x*]
oinsert x (*y* # *ys*) = (*if x* ≤ *y then x* # *y* # *ys else y* # *oinsert x ys*)

primrec
sort [] = []
sort (*x* # *xs*) = *oinsert x* (*sort xs*)

4.2 Arithmetic

lemma *one-less-m*: (*m*::*nat*) ≠ *m* ∗ *k* ==> *m* ≠ *Suc 0* ==> *Suc 0* < *m*
⟨*proof*⟩

lemma *one-less-k*: (*m*::*nat*) ≠ *m* ∗ *k* ==> *Suc 0* < *m* ∗ *k* ==> *Suc 0* < *k*
⟨*proof*⟩

lemma *mult-left-cancel*: (*0*::*nat*) < *k* ==> *k* ∗ *n* = *k* ∗ *m* ==> *n* = *m*
⟨*proof*⟩

lemma *mn-eq-m-one*: $(0::nat) < m \Longrightarrow m * n = m \Longrightarrow n = Suc\ 0$
⟨*proof*⟩

lemma *prod-mn-less-k*:
$(0::nat) < n \Longrightarrow 0 < k \Longrightarrow Suc\ 0 < m \Longrightarrow m * n = k \Longrightarrow n < k$
⟨*proof*⟩

4.3 Prime list and product

lemma *prod-append*: $prod\ (xs\ @\ ys) = prod\ xs * prod\ ys$
⟨*proof*⟩

lemma *prod-xy-prod*:
$prod\ (x\ \#\ xs) = prod\ (y\ \#\ ys) \Longrightarrow x * prod\ xs = y * prod\ ys$
⟨*proof*⟩

lemma *primel-append*: $primel\ (xs\ @\ ys) = (primel\ xs \wedge primel\ ys)$
⟨*proof*⟩

lemma *prime-primel*: $prime\ n \Longrightarrow primel\ [n] \wedge prod\ [n] = n$
⟨*proof*⟩

lemma *prime-nd-one*: $prime\ p \Longrightarrow \neg\ p\ dvd\ Suc\ 0$
⟨*proof*⟩

lemma *hd-dvd-prod*: $prod\ (x\ \#\ xs) = prod\ ys \Longrightarrow x\ dvd\ (prod\ ys)$
⟨*proof*⟩

lemma *primel-tl*: $primel\ (x\ \#\ xs) \Longrightarrow primel\ xs$
⟨*proof*⟩

lemma *primel-hd-tl*: $(primel\ (x\ \#\ xs)) = (prime\ x \wedge primel\ xs)$
⟨*proof*⟩

lemma *primes-eq*: $prime\ p \Longrightarrow prime\ q \Longrightarrow p\ dvd\ q \Longrightarrow p = q$
⟨*proof*⟩

lemma *primel-one-empty*: $primel\ xs \Longrightarrow prod\ xs = Suc\ 0 \Longrightarrow xs = []$
⟨*proof*⟩

lemma *prime-g-one*: $prime\ p \Longrightarrow Suc\ 0 < p$
⟨*proof*⟩

lemma *prime-g-zero*: $prime\ p \Longrightarrow 0 < p$
⟨*proof*⟩

lemma *primel-nempty-g-one*:
$primel\ xs \Longrightarrow xs \neq [] \Longrightarrow Suc\ 0 < prod\ xs$
⟨*proof*⟩

lemma *primel-prod-gz*: *primel xs ==> 0 < prod xs*
⟨*proof*⟩

4.4 Sorting

lemma *nondec-oinsert*: *nondec xs ⟹ nondec (oinsert x xs)*
⟨*proof*⟩

lemma *nondec-sort*: *nondec (sort xs)*
⟨*proof*⟩

lemma *x-less-y-oinsert*: *x ≤ y ==> l = y # ys ==> x # l = oinsert x l*
⟨*proof*⟩

lemma *nondec-sort-eq* [*rule-format*]: *nondec xs ⟶ xs = sort xs*
⟨*proof*⟩

lemma *oinsert-x-y*: *oinsert x (oinsert y l) = oinsert y (oinsert x l)*
⟨*proof*⟩

4.5 Permutation

lemma *perm-primel* [*rule-format*]: *xs <~~> ys ==> primel xs −−> primel ys*
⟨*proof*⟩

lemma *perm-prod*: *xs <~~> ys ==> prod xs = prod ys*
⟨*proof*⟩

lemma *perm-subst-oinsert*: *xs <~~> ys ==> oinsert a xs <~~> oinsert a ys*
⟨*proof*⟩

lemma *perm-oinsert*: *x # xs <~~> oinsert x xs*
⟨*proof*⟩

lemma *perm-sort*: *xs <~~> sort xs*
⟨*proof*⟩

lemma *perm-sort-eq*: *xs <~~> ys ==> sort xs = sort ys*
⟨*proof*⟩

4.6 Existence

lemma *ex-nondec-lemma*:
primel xs ==> ∃ ys. primel ys ∧ nondec ys ∧ prod ys = prod xs
⟨*proof*⟩

lemma *not-prime-ex-mk*:
Suc 0 < n ∧ ¬ prime n ==>
*∃ m k. Suc 0 < m ∧ Suc 0 < k ∧ m < n ∧ k < n ∧ n = m * k*

⟨*proof*⟩

lemma *split-primel*:
primel xs ⟹ *primel ys* ⟹ ∃ *l*. *primel l* ∧ *prod l* = *prod xs* ∗ *prod ys*
⟨*proof*⟩

lemma *factor-exists* [*rule-format*]: *Suc 0* < *n* −−> (∃ *l*. *primel l* ∧ *prod l* = *n*)
⟨*proof*⟩

lemma *nondec-factor-exists*: *Suc 0* < *n* ==> ∃ *l*. *primel l* ∧ *nondec l* ∧ *prod l* = *n*
⟨*proof*⟩

4.7 Uniqueness

lemma *prime-dvd-mult-list* [*rule-format*]:
prime p ==> *p dvd* (*prod xs*) −−> (∃ *m*. *m*:*set xs* ∧ *p dvd m*)
⟨*proof*⟩

lemma *hd-xs-dvd-prod*:
primel (*x* # *xs*) ==> *primel ys* ==> *prod* (*x* # *xs*) = *prod ys*
==> ∃ *m*. *m* ∈ *set ys* ∧ *x dvd m*
⟨*proof*⟩

lemma *prime-dvd-eq*: *primel* (*x* # *xs*) ==> *primel ys* ==> *m* ∈ *set ys* ==> *x dvd m* ==> *x* = *m*
⟨*proof*⟩

lemma *hd-xs-eq-prod*:
primel (*x* # *xs*) ==>
primel ys ==> *prod* (*x* # *xs*) = *prod ys* ==> *x* ∈ *set ys*
⟨*proof*⟩

lemma *perm-primel-ex*:
primel (*x* # *xs*) ==>
primel ys ==> *prod* (*x* # *xs*) = *prod ys* ==> ∃ *l*. *ys* <~~> (*x* # *l*)
⟨*proof*⟩

lemma *primel-prod-less*:
primel (*x* # *xs*) ==>
primel ys ==> *prod* (*x* # *xs*) = *prod ys* ==> *prod xs* < *prod ys*
⟨*proof*⟩

lemma *prod-one-empty*:
primel xs ==> *p* ∗ *prod xs* = *p* ==> *prime p* ==> *xs* = []
⟨*proof*⟩

lemma *uniq-ex-aux*:
∀ *m*. *m* < *prod ys* −−> (∀ *xs ys*. *primel xs* ∧ *primel ys* ∧

prod xs = prod ys ∧ prod xs = m −−> xs <~~> ys) ==>
primel list ==> primel x ==> prod list = prod x ==> prod x < prod ys
==> x <~~> list
⟨*proof*⟩

lemma *factor-unique* [*rule-format*]:
∀ *xs ys. primel xs ∧ primel ys ∧ prod xs = prod ys ∧ prod xs = n*
−−> xs <~~> ys
⟨*proof*⟩

lemma *perm-nondec-unique*:
xs <~~> ys ==> nondec xs ==> nondec ys ==> xs = ys
⟨*proof*⟩

theorem *unique-prime-factorization* [*rule-format*]:
∀ *n. Suc 0 < n −−> (∃!l. primel l ∧ nondec l ∧ prod l = n)*
⟨*proof*⟩

end

5 Divisibility and prime numbers (on integers)

theory *IntPrimes*
imports *Main Primes*
begin

The *dvd* relation, GCD, Euclid's extended algorithm, primes, congruences (all on the Integers). Comparable to theory *Primes*, but *dvd* is included here as it is not present in main HOL. Also includes extended GCD and congruences not present in *Primes*.

5.1 Definitions

fun
xzgcda :: int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int => (int ∗ int ∗ int)
where
xzgcda m n r′ r s′ s t′ t =
(if r ≤ 0 then (r′, s′, t′)
else xzgcda m n r (r′ mod r)
s (s′ − (r′ div r) ∗ s)
t (t′ − (r′ div r) ∗ t))

definition
zprime :: int ⇒ bool **where**
zprime p = (1 < p ∧ (∀ m. 0 <= m & m dvd p −−> m = 1 ∨ m = p))

definition

xzgcd :: *int* => *int* => *int* * *int* * *int* **where**
xzgcd m n = *xzgcda m n m n 1 0 0 1*

definition
zcong :: *int* => *int* => *int* => *bool* ((*1*[- = -] '(*mod* -')) **where**
[*a* = *b*] (*mod m*) = (*m dvd* (*a* − *b*))

5.2 Euclid's Algorithm and GCD

lemma *zrelprime-zdvd-zmult-aux*:
zgcd n k = *1* ==> *k dvd m* * *n* ==> *0* ≤ *m* ==> *k dvd m*
⟨*proof*⟩

lemma *zrelprime-zdvd-zmult*: *zgcd n k* = *1* ==> *k dvd m* * *n* ==> *k dvd m*
⟨*proof*⟩

lemma *zgcd-geq-zero*: *0* <= *zgcd x y*
⟨*proof*⟩

This is merely a sanity check on zprime, since the previous version denoted the empty set.

lemma *zprime 2*
⟨*proof*⟩

lemma *zprime-imp-zrelprime*:
zprime p ==> ¬ *p dvd n* ==> *zgcd n p* = *1*
⟨*proof*⟩

lemma *zless-zprime-imp-zrelprime*:
zprime p ==> *0* < *n* ==> *n* < *p* ==> *zgcd n p* = *1*
⟨*proof*⟩

lemma *zprime-zdvd-zmult*:
0 ≤ (*m*::*int*) ==> *zprime p* ==> *p dvd m* * *n* ==> *p dvd m* ∨ *p dvd n*
⟨*proof*⟩

lemma *zgcd-zadd-zmult* [*simp*]: *zgcd* (*m* + *n* * *k*) *n* = *zgcd m n*
⟨*proof*⟩

lemma *zgcd-zdvd-zgcd-zmult*: *zgcd m n dvd zgcd* (*k* * *m*) *n*
⟨*proof*⟩

lemma *zgcd-zmult-zdvd-zgcd*:
zgcd k n = *1* ==> *zgcd* (*k* * *m*) *n dvd zgcd m n*
⟨*proof*⟩

lemma *zgcd-zmult-cancel*: *zgcd k n* = *1* ==> *zgcd* (*k* * *m*) *n* = *zgcd m n*
⟨*proof*⟩

lemma *zgcd-zgcd-zmult*:
*zgcd k m = 1 ==> zgcd n m = 1 ==> zgcd (k * n) m = 1*
⟨*proof*⟩

lemma *zdvd-iff-zgcd*: *0 < m ==> m dvd n ⟷ zgcd n m = m*
⟨*proof*⟩

5.3 Congruences

lemma *zcong-1* [*simp*]: [*a = b*] (*mod 1*)
⟨*proof*⟩

lemma *zcong-refl* [*simp*]: [*k = k*] (*mod m*)
⟨*proof*⟩

lemma *zcong-sym*: [*a = b*] (*mod m*) = [*b = a*] (*mod m*)
⟨*proof*⟩

lemma *zcong-zadd*:
[*a = b*] (*mod m*) ==> [*c = d*] (*mod m*) ==> [*a + c = b + d*] (*mod m*)
⟨*proof*⟩

lemma *zcong-zdiff*:
[*a = b*] (*mod m*) ==> [*c = d*] (*mod m*) ==> [*a - c = b - d*] (*mod m*)
⟨*proof*⟩

lemma *zcong-trans*:
[*a = b*] (*mod m*) ==> [*b = c*] (*mod m*) ==> [*a = c*] (*mod m*)
⟨*proof*⟩

lemma *zcong-zmult*:
[*a = b*] (*mod m*) ==> [*c = d*] (*mod m*) ==> [*a * c = b * d*] (*mod m*)
⟨*proof*⟩

lemma *zcong-scalar*: [*a = b*] (*mod m*) ==> [*a * k = b * k*] (*mod m*)
⟨*proof*⟩

lemma *zcong-scalar2*: [*a = b*] (*mod m*) ==> [*k * a = k * b*] (*mod m*)
⟨*proof*⟩

lemma *zcong-zmult-self*: [*a * m = b * m*] (*mod m*)
⟨*proof*⟩

lemma *zcong-square*:
[| *zprime p*; *0 < a*; [*a * a = 1*] (*mod p*)|]
==> [*a = 1*] (*mod p*) ∨ [*a = p - 1*] (*mod p*)
⟨*proof*⟩

lemma *zcong-cancel*:

$0 \leq m \Longrightarrow$
$zgcd\ k\ m = 1 \Longrightarrow [a * k = b * k]\ (mod\ m) = [a = b]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-cancel2*:
$0 \leq m \Longrightarrow$
$zgcd\ k\ m = 1 \Longrightarrow [k * a = k * b]\ (mod\ m) = [a = b]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zgcd-zmult-zmod*:
$[a = b]\ (mod\ m) \Longrightarrow [a = b]\ (mod\ n) \Longrightarrow zgcd\ m\ n = 1$
$\Longrightarrow [a = b]\ (mod\ m * n)$
⟨*proof*⟩

lemma *zcong-zless-imp-eq*:
$0 \leq a \Longrightarrow$
$a < m \Longrightarrow 0 \leq b \Longrightarrow b < m \Longrightarrow [a = b]\ (mod\ m) \Longrightarrow a = b$
⟨*proof*⟩

lemma *zcong-square-zless*:
$zprime\ p \Longrightarrow 0 < a \Longrightarrow a < p \Longrightarrow$
$[a * a = 1]\ (mod\ p) \Longrightarrow a = 1 \vee a = p - 1$
⟨*proof*⟩

lemma *zcong-not*:
$0 < a \Longrightarrow a < m \Longrightarrow 0 < b \Longrightarrow b < a \Longrightarrow \neg\ [a = b]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zless-0*:
$0 \leq a \Longrightarrow a < m \Longrightarrow [a = 0]\ (mod\ m) \Longrightarrow a = 0$
⟨*proof*⟩

lemma *zcong-zless-unique*:
$0 < m \Longrightarrow (\exists!b.\ 0 \leq b \wedge b < m \wedge [a = b]\ (mod\ m))$
⟨*proof*⟩

lemma *zcong-iff-lin*: $([a = b]\ (mod\ m)) = (\exists\, k.\ b = a + m * k)$
⟨*proof*⟩

lemma *zgcd-zcong-zgcd*:
$0 < m \Longrightarrow$
$zgcd\ a\ m = 1 \Longrightarrow [a = b]\ (mod\ m) \Longrightarrow zgcd\ b\ m = 1$
⟨*proof*⟩

lemma *zcong-zmod-aux*:
$a - b = (m::int) * (a\ div\ m - b\ div\ m) + (a\ mod\ m - b\ mod\ m)$
⟨*proof*⟩

lemma *zcong-zmod*: $[a = b]\ (mod\ m) = [a\ mod\ m = b\ mod\ m]\ (mod\ m)$

⟨*proof*⟩

lemma *zcong-zmod-eq*: $0 < m \Longrightarrow [a = b]\ (mod\ m) = (a\ mod\ m = b\ mod\ m)$
⟨*proof*⟩

lemma *zcong-zminus* [*iff*]: $[a = b]\ (mod\ -m) = [a = b]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zero* [*iff*]: $[a = b]\ (mod\ 0) = (a = b)$
⟨*proof*⟩

lemma $[a = b]\ (mod\ m) = (a\ mod\ m = b\ mod\ m)$
⟨*proof*⟩

5.4 Modulo

lemma *zmod-zdvd-zmod*:
$0 < (m::int) \Longrightarrow m\ dvd\ b \Longrightarrow (a\ mod\ b\ mod\ m) = (a\ mod\ m)$
⟨*proof*⟩

5.5 Extended GCD

declare *xzgcda.simps* [*simp del*]

lemma *xzgcd-correct-aux1*:
$zgcd\ r'\ r = k \longrightarrow 0 < r \longrightarrow$
$(\exists sn\ tn.\ xzgcda\ m\ n\ r'\ r\ s'\ s\ t'\ t = (k,\ sn,\ tn))$
⟨*proof*⟩

lemma *xzgcd-correct-aux2*:
$(\exists sn\ tn.\ xzgcda\ m\ n\ r'\ r\ s'\ s\ t'\ t = (k,\ sn,\ tn)) \longrightarrow 0 < r \longrightarrow$
$zgcd\ r'\ r = k$
⟨*proof*⟩

lemma *xzgcd-correct*:
$0 < n \Longrightarrow (zgcd\ m\ n = k) = (\exists s\ t.\ xzgcd\ m\ n = (k,\ s,\ t))$
⟨*proof*⟩

xzgcd linear

lemma *xzgcda-linear-aux1*:
$(a - r * b) * m + (c - r * d) * (n::int) =$
$(a * m + c * n) - r * (b * m + d * n)$
⟨*proof*⟩

lemma *xzgcda-linear-aux2*:
$r' = s' * m + t' * n \Longrightarrow r = s * m + t * n$
$\Longrightarrow (r'\ mod\ r) = (s' - (r'\ div\ r) * s) * m + (t' - (r'\ div\ r) * t) * (n::int)$
⟨*proof*⟩

lemma *order-le-neq-implies-less*: $(x::'a::order) \leq y \Longrightarrow x \neq y \Longrightarrow x < y$
⟨*proof*⟩

lemma *xzgcda-linear* [*rule-format*]:
$0 < r \longrightarrow xzgcda\ m\ n\ r'\ r\ s'\ s\ t'\ t = (rn,\ sn,\ tn) \longrightarrow$
$r' = s' * m + t' * n \longrightarrow\ r = s * m + t * n \longrightarrow rm = sn * m + tn * n$
⟨*proof*⟩

lemma *xzgcd-linear*:
$0 < n \Longrightarrow xzgcd\ m\ n = (r,\ s,\ t) \Longrightarrow r = s * m + t * n$
⟨*proof*⟩

lemma *zgcd-ex-linear*:
$0 < n \Longrightarrow zgcd\ m\ n = k \Longrightarrow (\exists\, s\ t.\ k = s * m + t * n)$
⟨*proof*⟩

lemma *zcong-lineq-ex*:
$0 < n \Longrightarrow zgcd\ a\ n = 1 \Longrightarrow \exists\, x.\ [a * x = 1]\ (mod\ n)$
⟨*proof*⟩

lemma *zcong-lineq-unique*:
$0 < n \Longrightarrow$
$zgcd\ a\ n = 1 \Longrightarrow \exists!x.\ 0 \leq x \wedge x < n \wedge [a * x = b]\ (mod\ n)$
⟨*proof*⟩

end

6 The Chinese Remainder Theorem

theory *Chinese*
imports *IntPrimes*
begin

The Chinese Remainder Theorem for an arbitrary finite number of equations. (The one-equation case is included in theory *IntPrimes*. Uses functions for indexing.[1]

6.1 Definitions

consts
$funprod :: (nat \Rightarrow int) \Rightarrow nat \Rightarrow nat \Rightarrow int$
$funsum :: (nat \Rightarrow int) \Rightarrow nat \Rightarrow nat \Rightarrow int$

primrec
$funprod\ f\ i\ 0 = f\ i$
$funprod\ f\ i\ (Suc\ n) = f\ (Suc\ (i + n)) * funprod\ f\ i\ n$

[1] Maybe *funprod* and *funsum* should be based on general *fold* on indices?

primrec
funsum f i 0 = f i
funsum f i (Suc n) = f (Suc (i + n)) + funsum f i n

definition
m-cond :: *nat => (nat => int) => bool* **where**
m-cond n mf =
((∀ i. i ≤ n --> 0 < mf i) ∧
(∀ i j. i ≤ n ∧ j ≤ n ∧ i ≠ j --> zgcd (mf i) (mf j) = 1))

definition
km-cond :: *nat => (nat => int) => (nat => int) => bool* **where**
km-cond n kf mf = (∀ i. i ≤ n --> zgcd (kf i) (mf i) = 1)

definition
lincong-sol ::
nat => (nat => int) => (nat => int) => (nat => int) => int => bool
where
*lincong-sol n kf bf mf x = (∀ i. i ≤ n --> zcong (kf i * x) (bf i) (mf i))*

definition
mhf :: *(nat => int) => nat => nat => int* **where**
mhf mf n i =
(if i = 0 then funprod mf (Suc 0) (n − Suc 0)
else if i = n then funprod mf 0 (n − Suc 0)
*else funprod mf 0 (i − Suc 0) * funprod mf (Suc i) (n − Suc 0 − i))*

definition
xilin-sol ::
nat => nat => (nat => int) => (nat => int) => (nat => int) => int
where
xilin-sol i n kf bf mf =
(if 0 < n ∧ i ≤ n ∧ m-cond n mf ∧ km-cond n kf mf then
*(SOME x. 0 ≤ x ∧ x < mf i ∧ zcong (kf i * mhf mf n i * x) (bf i) (mf i))*
else 0)

definition
x-sol :: *nat => (nat => int) => (nat => int) => (nat => int) => int* **where**
*x-sol n kf bf mf = funsum (λi. xilin-sol i n kf bf mf * mhf mf n i) 0 n*

funprod and *funsum*

lemma *funprod-pos*: *(∀ i. i ≤ n --> 0 < mf i) ==> 0 < funprod mf 0 n*
⟨*proof*⟩

lemma *funprod-zgcd* [*rule-format* (*no-asm*)]:
(∀ i. k ≤ i ∧ i ≤ k + l --> zgcd (mf i) (mf m) = 1) -->
zgcd (funprod mf k l) (mf m) = 1
⟨*proof*⟩

lemma *funprod-zdvd* [*rule-format*]:
$k \leq i \longrightarrow i \leq k + l \longrightarrow mf\ i\ dvd\ funprod\ mf\ k\ l$
⟨*proof*⟩

lemma *funsum-mod*:
$funsum\ f\ k\ l\ mod\ m = funsum\ (\lambda i.\ (f\ i)\ mod\ m)\ k\ l\ mod\ m$
⟨*proof*⟩

lemma *funsum-zero* [*rule-format* (*no-asm*)]:
$(\forall i.\ k \leq i \wedge i \leq k + l \longrightarrow f\ i = 0) \longrightarrow (funsum\ f\ k\ l) = 0$
⟨*proof*⟩

lemma *funsum-oneelem* [*rule-format* (*no-asm*)]:
$k \leq j \longrightarrow j \leq k + l \longrightarrow$
$(\forall i.\ k \leq i \wedge i \leq k + l \wedge i \neq j \longrightarrow f\ i = 0) \longrightarrow$
$funsum\ f\ k\ l = f\ j$
⟨*proof*⟩

6.2 Chinese: uniqueness

lemma *zcong-funprod-aux*:
$m\text{-}cond\ n\ mf \Longrightarrow km\text{-}cond\ n\ kf\ mf$
$\Longrightarrow lincong\text{-}sol\ n\ kf\ bf\ mf\ x \Longrightarrow lincong\text{-}sol\ n\ kf\ bf\ mf\ y$
$\Longrightarrow [x = y]\ (mod\ mf\ n)$
⟨*proof*⟩

lemma *zcong-funprod* [*rule-format*]:
$m\text{-}cond\ n\ mf \longrightarrow km\text{-}cond\ n\ kf\ mf \longrightarrow$
$lincong\text{-}sol\ n\ kf\ bf\ mf\ x \longrightarrow lincong\text{-}sol\ n\ kf\ bf\ mf\ y \longrightarrow$
$[x = y]\ (mod\ funprod\ mf\ 0\ n)$
⟨*proof*⟩

6.3 Chinese: existence

lemma *unique-xi-sol*:
$0 < n \Longrightarrow i \leq n \Longrightarrow m\text{-}cond\ n\ mf \Longrightarrow km\text{-}cond\ n\ kf\ mf$
$\Longrightarrow \exists!x.\ 0 \leq x \wedge x < mf\ i \wedge [kf\ i * mhf\ mf\ n\ i * x = bf\ i]\ (mod\ mf\ i)$
⟨*proof*⟩

lemma *x-sol-lin-aux*:
$0 < n \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow j \neq i \Longrightarrow mf\ j\ dvd\ mhf\ mf\ n\ i$
⟨*proof*⟩

lemma *x-sol-lin*:
$0 < n \Longrightarrow i \leq n$
$\Longrightarrow x\text{-}sol\ n\ kf\ bf\ mf\ mod\ mf\ i =$
$xilin\text{-}sol\ i\ n\ kf\ bf\ mf\ *\ mhf\ mf\ n\ i\ mod\ mf\ i$
⟨*proof*⟩

6.4 Chinese

lemma *chinese-remainder*:
0 < n ==> m-cond n mf ==> km-cond n kf mf
==> ∃!x. 0 ≤ x ∧ x < funprod mf 0 n ∧ lincong-sol n kf bf mf x
⟨*proof*⟩

end

7 Bijections between sets

theory *BijectionRel* **imports** *Main* **begin**

Inductive definitions of bijections between two different sets and between the same set. Theorem for relating the two definitions.

inductive-set
bijR :: (*'a => 'b => bool*) => (*'a set * 'b set*) *set*
for *P* :: *'a => 'b => bool*
where
empty [*simp*]: ({}, {}) ∈ *bijR P*
| *insert*: *P a b ==> a ∉ A ==> b ∉ B ==>* (*A*, *B*) ∈ *bijR P*
==> (*insert a A*, *insert b B*) ∈ *bijR P*

Add extra condition to *insert*: ∀ *b*∈*B*. ¬ *P a b* (and similar for *A*).

definition
bijP :: (*'a => 'a => bool*) => *'a set => bool* **where**
bijP P F = (∀ *a b*. *a* ∈ *F* ∧ *P a b* −−> *b* ∈ *F*)

definition
uniqP :: (*'a => 'a => bool*) => *bool* **where**
uniqP P = (∀ *a b c d*. *P a b* ∧ *P c d* −−> (*a* = *c*) = (*b* = *d*))

definition
symP :: (*'a => 'a => bool*) => *bool* **where**
symP P = (∀ *a b*. *P a b* = *P b a*)

inductive-set
bijER :: (*'a => 'a => bool*) => *'a set set*
for *P* :: *'a => 'a => bool*
where
empty [*simp*]: {} ∈ *bijER P*
| *insert1*: *P a a ==> a ∉ A ==> A ∈ bijER P ==> insert a A ∈ bijER P*
| *insert2*: *P a b ==> a ≠ b ==> a ∉ A ==> b ∉ A ==> A ∈ bijER P*
==> insert a (*insert b A*) ∈ *bijER P*

bijR

lemma *fin-bijRl*: $(A, B) \in bijR\ P \Longrightarrow finite\ A$
⟨*proof*⟩

lemma *fin-bijRr*: $(A, B) \in bijR\ P \Longrightarrow finite\ B$
⟨*proof*⟩

lemma *aux-induct*:
assumes *major*: *finite F*
and *subs*: $F \subseteq A$
and *cases*: $P\ \{\}$
$!!F\ a.\ F \subseteq A \Longrightarrow a \in A \Longrightarrow a \notin F \Longrightarrow P\ F \Longrightarrow P\ (insert\ a\ F)$
shows $P\ F$
⟨*proof*⟩

lemma *inj-func-bijR-aux1*:
$A \subseteq B \Longrightarrow a \notin A \Longrightarrow a \in B \Longrightarrow inj\text{-}on\ f\ B \Longrightarrow f\ a \notin f\ `\ A$
⟨*proof*⟩

lemma *inj-func-bijR-aux2*:
$\forall a.\ a \in A \longrightarrow P\ a\ (f\ a) \Longrightarrow inj\text{-}on\ f\ A \Longrightarrow finite\ A \Longrightarrow F <= A$
$\Longrightarrow (F, f\ `\ F) \in bijR\ P$
⟨*proof*⟩

lemma *inj-func-bijR*:
$\forall a.\ a \in A \longrightarrow P\ a\ (f\ a) \Longrightarrow inj\text{-}on\ f\ A \Longrightarrow finite\ A$
$\Longrightarrow (A, f\ `\ A) \in bijR\ P$
⟨*proof*⟩

bijER

lemma *fin-bijER*: $A \in bijER\ P \Longrightarrow finite\ A$
⟨*proof*⟩

lemma *aux1*:
$a \notin A \Longrightarrow a \notin B \Longrightarrow F \subseteq insert\ a\ A \Longrightarrow F \subseteq insert\ a\ B \Longrightarrow a \in F$
$\Longrightarrow \exists C.\ F = insert\ a\ C \wedge a \notin C \wedge C <= A \wedge C <= B$
⟨*proof*⟩

lemma *aux2*: $a \neq b \Longrightarrow a \notin A \Longrightarrow b \notin B \Longrightarrow a \in F \Longrightarrow b \in F$
$\Longrightarrow F \subseteq insert\ a\ A \Longrightarrow F \subseteq insert\ b\ B$
$\Longrightarrow \exists C.\ F = insert\ a\ (insert\ b\ C) \wedge a \notin C \wedge b \notin C \wedge C \subseteq A \wedge C \subseteq B$
⟨*proof*⟩

lemma *aux-uniq*: $uniqP\ P \Longrightarrow P\ a\ b \Longrightarrow P\ c\ d \Longrightarrow (a = c) = (b = d)$
⟨*proof*⟩

lemma *aux-sym*: $symP\ P \Longrightarrow P\ a\ b = P\ b\ a$
⟨*proof*⟩

lemma *aux-in1*:
uniqP P ==> b ∉ C ==> P b b ==> bijP P (insert b C) ==> bijP P C
⟨*proof*⟩

lemma *aux-in2*:
symP P ==> uniqP P ==> a ∉ C ==> b ∉ C ==> a ≠ b ==> P a b
==> bijP P (insert a (insert b C)) ==> bijP P C
⟨*proof*⟩

lemma *aux-foo*: *∀ a b. Q a ∧ P a b −−> R b ==> P a b ==> Q a ==> R b*
⟨*proof*⟩

lemma *aux-bij*: *bijP P F ==> symP P ==> P a b ==> (a ∈ F) = (b ∈ F)*
⟨*proof*⟩

lemma *aux-bijRER*:
(A, B) ∈ bijR P ==> uniqP P ==> symP P
==> ∀ F. bijP P F ∧ F ⊆ A ∧ F ⊆ B −−> F ∈ bijER P
⟨*proof*⟩

lemma *bijR-bijER*:
(A, A) ∈ bijR P ==>
bijP P A ==> uniqP P ==> symP P ==> A ∈ bijER P
⟨*proof*⟩

end

8 Factorial on integers

theory *IntFact* **imports** *IntPrimes* **begin**

Factorial on integers and recursively defined set including all Integers from *2* up to *a*. Plus definition of product of finite set.

fun
zfact :: int => int
where
*zfact n = (if n ≤ 0 then 1 else n * zfact (n − 1))*

fun
d22set :: int => int set
where
d22set a = (if 1 < a then insert a (d22set (a − 1)) else {})

d22set — recursively defined set including all integers from *2* up to *a*

declare *d22set.simps* [*simp del*]

lemma *d22set-induct*:
assumes !!*a*. *P* {} *a*
and !!*a*. *1* < (*a*::*int*) ==> *P* (*d22set* (*a* − *1*)) (*a* − *1*) ==> *P* (*d22set a*) *a*
shows *P* (*d22set u*) *u*
⟨*proof*⟩

lemma *d22set-g-1* [*rule-format*]: *b* ∈ *d22set a* −−> *1* < *b*
⟨*proof*⟩

lemma *d22set-le* [*rule-format*]: *b* ∈ *d22set a* −−> *b* ≤ *a*
⟨*proof*⟩

lemma *d22set-le-swap*: *a* < *b* ==> *b* ∉ *d22set a*
⟨*proof*⟩

lemma *d22set-mem*: *1* < *b* ⟹ *b* ≤ *a* ⟹ *b* ∈ *d22set a*
⟨*proof*⟩

lemma *d22set-fin*: *finite* (*d22set a*)
⟨*proof*⟩

declare *zfact.simps* [*simp del*]

lemma *d22set-prod-zfact*: ∏(*d22set a*) = *zfact a*
⟨*proof*⟩

end

9 Fermat's Little Theorem extended to Euler's Totient function

theory *EulerFermat*
imports *BijectionRel IntFact*
begin

Fermat's Little Theorem extended to Euler's Totient function. More abstract approach than Boyer-Moore (which seems necessary to achieve the extended version).

9.1 Definitions and lemmas

inductive-set
RsetR :: *int* => *int set set*
for *m* :: *int*

where
empty [*simp*]: $\{\} \in RsetR\ m$
| *insert*: $A \in RsetR\ m \Longrightarrow zgcd\ a\ m = 1 \Longrightarrow$
$\forall a'.\ a' \in A \longrightarrow \neg\ zcong\ a\ a'\ m \Longrightarrow insert\ a\ A \in RsetR\ m$

fun
BnorRset :: *int* ⇒ *int* => *int set*
where
BnorRset a m =
(*if 0 < a then*
let na = *BnorRset* (*a* − *1*) *m*
in (*if zgcd a m* = *1 then insert a na else na*)
else {})

definition
norRRset :: *int* => *int set* **where**
norRRset m = *BnorRset* (*m* − *1*) *m*

definition
noXRRset :: *int* => *int* => *int set* **where**
noXRRset m x = (λ*a*. *a* ∗ *x*) ‘ *norRRset m*

definition
phi :: *int* => *nat* **where**
phi m = *card* (*norRRset m*)

definition
is-RRset :: *int set* => *int* => *bool* **where**
is-RRset A m = (*A* ∈ *RsetR m* ∧ *card A* = *phi m*)

definition
RRset2norRR :: *int set* => *int* => *int* => *int* **where**
RRset2norRR A m a =
(*if 1 < m* ∧ *is-RRset A m* ∧ *a* ∈ *A then*
SOME b. *zcong a b m* ∧ *b* ∈ *norRRset m*
else 0)

definition
zcongm :: *int* => *int* => *int* => *bool* **where**
zcongm m = (λ*a b*. *zcong a b m*)

lemma *abs-eq-1-iff* [*iff*]: (*abs z* = (*1*::*int*)) = (*z* = *1* ∨ *z* = −*1*)
— LCP: not sure why this lemma is needed now
⟨*proof*⟩

norRRset

declare *BnorRset.simps* [*simp del*]

lemma *BnorRset-induct*:

assumes !!$a\ m.\ P\ \{\}\ a\ m$
and !!$a\ m :: int.\ 0 < a \Longrightarrow P\ (BnorRset\ (a - 1)\ m)\ (a - 1)\ m \Longrightarrow P\ (BnorRset\ a\ m)\ a\ m$
shows $P\ (BnorRset\ u\ v)\ u\ v$
⟨*proof*⟩

lemma *Bnor-mem-zle* [*rule-format*]: $b \in BnorRset\ a\ m \longrightarrow b \leq a$
⟨*proof*⟩

lemma *Bnor-mem-zle-swap*: $a < b \Longrightarrow b \notin BnorRset\ a\ m$
⟨*proof*⟩

lemma *Bnor-mem-zg* [*rule-format*]: $b \in BnorRset\ a\ m \dashrightarrow 0 < b$
⟨*proof*⟩

lemma *Bnor-mem-if* [*rule-format*]:
$zgcd\ b\ m = 1 \dashrightarrow 0 < b \dashrightarrow b \leq a \dashrightarrow b \in BnorRset\ a\ m$
⟨*proof*⟩

lemma *Bnor-in-RsetR* [*rule-format*]: $a < m \dashrightarrow BnorRset\ a\ m \in RsetR\ m$
⟨*proof*⟩

lemma *Bnor-fin*: *finite* $(BnorRset\ a\ m)$
⟨*proof*⟩

lemma *norR-mem-unique-aux*: $a \leq b - 1 \Longrightarrow a < (b::int)$
⟨*proof*⟩

lemma *norR-mem-unique*:
$1 < m \Longrightarrow zgcd\ a\ m = 1 \Longrightarrow \exists!b.\ [a = b]\ (mod\ m) \wedge b \in norRRset\ m$
⟨*proof*⟩

noXRRset

lemma *RRset-gcd* [*rule-format*]:
$is\text{-}RRset\ A\ m \Longrightarrow a \in A \dashrightarrow zgcd\ a\ m = 1$
⟨*proof*⟩

lemma *RsetR-zmult-mono*:
$A \in RsetR\ m \Longrightarrow 0 < m \Longrightarrow zgcd\ x\ m = 1 \Longrightarrow (\lambda a.\ a * x)\ `\ A \in RsetR\ m$
⟨*proof*⟩

lemma *card-nor-eq-noX*:
$0 < m \Longrightarrow zgcd\ x\ m = 1 \Longrightarrow card\ (noXRRset\ m\ x) = card\ (norRRset\ m)$
⟨*proof*⟩

lemma *noX-is-RRset*:

$0 < m \Longrightarrow zgcd\ x\ m = 1 \Longrightarrow is\text{-}RRset\ (noXRRset\ m\ x)\ m$
⟨*proof*⟩

lemma *aux-some*:
$1 < m \Longrightarrow is\text{-}RRset\ A\ m \Longrightarrow a \in A$
$\Longrightarrow zcong\ a\ (SOME\ b.\ [a = b]\ (mod\ m) \wedge b \in norRRset\ m)\ m \wedge$
$(SOME\ b.\ [a = b]\ (mod\ m) \wedge b \in norRRset\ m) \in norRRset\ m$
⟨*proof*⟩

lemma *RRset2norRR-correct*:
$1 < m \Longrightarrow is\text{-}RRset\ A\ m \Longrightarrow a \in A \Longrightarrow$
$[a = RRset2norRR\ A\ m\ a]\ (mod\ m) \wedge RRset2norRR\ A\ m\ a \in norRRset\ m$
⟨*proof*⟩

lemmas *RRset2norRR-correct1* =
RRset2norRR-correct [*THEN conjunct1*, *standard*]
lemmas *RRset2norRR-correct2* =
RRset2norRR-correct [*THEN conjunct2*, *standard*]

lemma *RsetR-fin*: $A \in RsetR\ m \Longrightarrow finite\ A$
⟨*proof*⟩

lemma *RRset-zcong-eq* [*rule-format*]:
$1 < m \Longrightarrow$
$is\text{-}RRset\ A\ m \Longrightarrow [a = b]\ (mod\ m) \Longrightarrow a \in A \longrightarrow b \in A \longrightarrow a = b$
⟨*proof*⟩

lemma *aux*:
$P\ (SOME\ a.\ P\ a) \Longrightarrow Q\ (SOME\ a.\ Q\ a) \Longrightarrow$
$(SOME\ a.\ P\ a) = (SOME\ a.\ Q\ a) \Longrightarrow \exists a.\ P\ a \wedge Q\ a$
⟨*proof*⟩

lemma *RRset2norRR-inj*:
$1 < m \Longrightarrow is\text{-}RRset\ A\ m \Longrightarrow inj\text{-}on\ (RRset2norRR\ A\ m)\ A$
⟨*proof*⟩

lemma *RRset2norRR-eq-norR*:
$1 < m \Longrightarrow is\text{-}RRset\ A\ m \Longrightarrow RRset2norRR\ A\ m\ `\ A = norRRset\ m$
⟨*proof*⟩

lemma *Bnor-prod-power-aux*: $a \notin A \Longrightarrow inj\ f \Longrightarrow f\ a \notin f\ `\ A$
⟨*proof*⟩

lemma *Bnor-prod-power* [*rule-format*]:
$x \neq 0 \Longrightarrow a < m \longrightarrow \prod((\lambda a.\ a * x)\ `\ BnorRset\ a\ m) =$
$\prod(BnorRset\ a\ m) * x\hat{}card\ (BnorRset\ a\ m)$
⟨*proof*⟩

9.2 Fermat

lemma *bijzcong-zcong-prod*:
$(A, B) \in bijR\ (zcongm\ m) ==> [\prod A = \prod B]\ (mod\ m)$
⟨*proof*⟩

lemma *Bnor-prod-zgcd* [*rule-format*]:
$a < m --> zgcd\ (\prod (BnorRset\ a\ m))\ m = 1$
⟨*proof*⟩

theorem *Euler-Fermat*:
$0 < m ==> zgcd\ x\ m = 1 ==> [x\hat{}(phi\ m) = 1]\ (mod\ m)$
⟨*proof*⟩

lemma *Bnor-prime*:
$[\![\ zprime\ p;\ a < p\ ]\!] \Longrightarrow card\ (BnorRset\ a\ p) = nat\ a$
⟨*proof*⟩

lemma *phi-prime*: $zprime\ p ==> phi\ p = nat\ (p - 1)$
⟨*proof*⟩

theorem *Little-Fermat*:
$zprime\ p ==> \neg\ p\ dvd\ x ==> [x\hat{}(nat\ (p - 1)) = 1]\ (mod\ p)$
⟨*proof*⟩

end

10 Wilson's Theorem according to Russinoff

theory *WilsonRuss* **imports** *EulerFermat* **begin**

Wilson's Theorem following quite closely Russinoff's approach using Boyer-Moore (using finite sets instead of lists, though).

10.1 Definitions and lemmas

definition
$inv :: int => int => int$ **where**
$inv\ p\ a = (a\hat{}(nat\ (p - 2)))\ mod\ p$

fun
$wset :: int \Rightarrow int => int\ set$
where
$wset\ a\ p =$
$(if\ 1 < a\ then$
$let\ ws = wset\ (a - 1)\ p$
$in\ (if\ a \in ws\ then\ ws\ else\ insert\ a\ (insert\ (inv\ p\ a)\ ws))\ else\ \{\})$

inv

lemma *inv-is-inv-aux*: $1 < m \Longrightarrow Suc\ (nat\ (m - 2)) = nat\ (m - 1)$
⟨*proof*⟩

lemma *inv-is-inv*:
$zprime\ p \Longrightarrow 0 < a \Longrightarrow a < p \Longrightarrow [a * inv\ p\ a = 1]\ (mod\ p)$
⟨*proof*⟩

lemma *inv-distinct*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 \Longrightarrow a \neq inv\ p\ a$
⟨*proof*⟩

lemma *inv-not-0*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 \Longrightarrow inv\ p\ a \neq 0$
⟨*proof*⟩

lemma *inv-not-1*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 \Longrightarrow inv\ p\ a \neq 1$
⟨*proof*⟩

lemma *inv-not-p-minus-1-aux*:
$[a * (p - 1) = 1]\ (mod\ p) = [a = p - 1]\ (mod\ p)$
⟨*proof*⟩

lemma *inv-not-p-minus-1*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 \Longrightarrow inv\ p\ a \neq p - 1$
⟨*proof*⟩

lemma *inv-g-1*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 \Longrightarrow 1 < inv\ p\ a$
⟨*proof*⟩

lemma *inv-less-p-minus-1*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 \Longrightarrow inv\ p\ a < p - 1$
⟨*proof*⟩

lemma *inv-inv-aux*: $5 \leq p \Longrightarrow$
$nat\ (p - 2) * nat\ (p - 2) = Suc\ (nat\ (p - 1) * nat\ (p - 3))$
⟨*proof*⟩

lemma *zcong-zpower-zmult*:
$[x\hat{}y = 1]\ (mod\ p) \Longrightarrow [x\hat{}(y * z) = 1]\ (mod\ p)$
⟨*proof*⟩

lemma *inv-inv*: $zprime\ p \Longrightarrow$
$5 \leq p \Longrightarrow 0 < a \Longrightarrow a < p \Longrightarrow inv\ p\ (inv\ p\ a) = a$
⟨*proof*⟩

wset

declare *wset.simps* [*simp del*]

lemma *wset-induct*:
assumes !!*a p*. *P* {} *a p*
and !!*a p*. *1* < (*a*::*int*) ⟹
P (*wset* (*a* − *1*) *p*) (*a* − *1*) *p* ==> *P* (*wset a p*) *a p*
shows *P* (*wset u v*) *u v*
⟨*proof*⟩

lemma *wset-mem-imp-or* [*rule-format*]:
1 < *a* ⟹ *b* ∉ *wset* (*a* − *1*) *p*
==> *b* ∈ *wset a p* −−> *b* = *a* ∨ *b* = *inv p a*
⟨*proof*⟩

lemma *wset-mem-mem* [*simp*]: *1* < *a* ==> *a* ∈ *wset a p*
⟨*proof*⟩

lemma *wset-subset*: *1* < *a* ⟹ *b* ∈ *wset* (*a* − *1*) *p* ==> *b* ∈ *wset a p*
⟨*proof*⟩

lemma *wset-g-1* [*rule-format*]:
zprime p −−> *a* < *p* − *1* −−> *b* ∈ *wset a p* −−> *1* < *b*
⟨*proof*⟩

lemma *wset-less* [*rule-format*]:
zprime p −−> *a* < *p* − *1* −−> *b* ∈ *wset a p* −−> *b* < *p* − *1*
⟨*proof*⟩

lemma *wset-mem* [*rule-format*]:
zprime p −−>
a < *p* − *1* −−> *1* < *b* −−> *b* ≤ *a* −−> *b* ∈ *wset a p*
⟨*proof*⟩

lemma *wset-mem-inv-mem* [*rule-format*]:
zprime p −−> *5* ≤ *p* −−> *a* < *p* − *1* −−> *b* ∈ *wset a p*
−−> *inv p b* ∈ *wset a p*
⟨*proof*⟩

lemma *wset-inv-mem-mem*:
zprime p ⟹ *5* ≤ *p* ⟹ *a* < *p* − *1* ⟹ *1* < *b* ⟹ *b* < *p* − *1*
⟹ *inv p b* ∈ *wset a p* ⟹ *b* ∈ *wset a p*
⟨*proof*⟩

lemma *wset-fin*: *finite* (*wset a p*)
⟨*proof*⟩

lemma *wset-zcong-prod-1* [*rule-format*]:
zprime p −−>
5 ≤ *p* −−> *a* < *p* − *1* −−> [(∏ *x*∈*wset a p*. *x*) = *1*] (*mod p*)

⟨*proof*⟩

lemma *d22set-eq-wset*: *zprime p ==> d22set (p − 2) = wset (p − 2) p*
⟨*proof*⟩

10.2 Wilson

lemma *prime-g-5*: *zprime p ⟹ p ≠ 2 ⟹ p ≠ 3 ==> 5 ≤ p*
⟨*proof*⟩

theorem *Wilson-Russ*:
zprime p ==> [zfact (p − 1) = −1] (mod p)
⟨*proof*⟩

end

11 Wilson's Theorem using a more abstract approach

theory *WilsonBij* **imports** *BijectionRel IntFact* **begin**

Wilson's Theorem using a more "abstract" approach based on bijections between sets. Does not use Fermat's Little Theorem (unlike Russinoff).

11.1 Definitions and lemmas

definition
reciR :: int => int => int => bool **where**
reciR p = (λa b. zcong (a ∗ b) 1 p ∧ 1 < a ∧ a < p − 1 ∧ 1 < b ∧ b < p − 1)

definition
inv :: int => int => int **where**
inv p a =
(if zprime p ∧ 0 < a ∧ a < p then
(SOME x. 0 ≤ x ∧ x < p ∧ zcong (a ∗ x) 1 p)
else 0)

Inverse

lemma *inv-correct*:
zprime p ==> 0 < a ==> a < p
==> 0 ≤ inv p a ∧ inv p a < p ∧ [a ∗ inv p a = 1] (mod p)
⟨*proof*⟩

lemmas *inv-ge = inv-correct [THEN conjunct1, standard]*
lemmas *inv-less = inv-correct [THEN conjunct2, THEN conjunct1, standard]*
lemmas *inv-is-inv = inv-correct [THEN conjunct2, THEN conjunct2, standard]*

lemma *inv-not-0*:
zprime p ==> 1 < a ==> a < p − 1 ==> inv p a ≠ 0
— same as *WilsonRuss*
⟨*proof*⟩

lemma *inv-not-1*:
zprime p ==> 1 < a ==> a < p − 1 ==> inv p a ≠ 1
— same as *WilsonRuss*
⟨*proof*⟩

lemma *aux*: *[a * (p − 1) = 1] (mod p) = [a = p − 1] (mod p)*
— same as *WilsonRuss*
⟨*proof*⟩

lemma *inv-not-p-minus-1*:
zprime p ==> 1 < a ==> a < p − 1 ==> inv p a ≠ p − 1
— same as *WilsonRuss*
⟨*proof*⟩

Below is slightly different as we don't expand *inv* but use "*correct*" theorems.

lemma *inv-g-1*: *zprime p ==> 1 < a ==> a < p − 1 ==> 1 < inv p a*
⟨*proof*⟩

lemma *inv-less-p-minus-1*:
zprime p ==> 1 < a ==> a < p − 1 ==> inv p a < p − 1
— ditto
⟨*proof*⟩

Bijection

lemma *aux1*: *1 < x ==> 0 ≤ (x::int)*
⟨*proof*⟩

lemma *aux2*: *1 < x ==> 0 < (x::int)*
⟨*proof*⟩

lemma *aux3*: *x ≤ p − 2 ==> x < (p::int)*
⟨*proof*⟩

lemma *aux4*: *x ≤ p − 2 ==> x < (p::int) − 1*
⟨*proof*⟩

lemma *inv-inj*: *zprime p ==> inj-on (inv p) (d22set (p − 2))*
⟨*proof*⟩

lemma *inv-d22set-d22set*:
zprime p ==> inv p ' d22set (p − 2) = d22set (p − 2)
⟨*proof*⟩

lemma *d22set-d22set-bij*:
zprime p ==> (d22set (p − 2), d22set (p − 2)) ∈ bijR (reciR p)
⟨*proof*⟩

lemma *reciP-bijP*: *zprime p ==> bijP (reciR p) (d22set (p − 2))*
⟨*proof*⟩

lemma *reciP-uniq*: *zprime p ==> uniqP (reciR p)*
⟨*proof*⟩

lemma *reciP-sym*: *zprime p ==> symP (reciR p)*
⟨*proof*⟩

lemma *bijER-d22set*: *zprime p ==> d22set (p − 2) ∈ bijER (reciR p)*
⟨*proof*⟩

11.2 Wilson

lemma *bijER-zcong-prod-1*:
zprime p ==> A ∈ bijER (reciR p) ==> [∏ A = 1] (mod p)
⟨*proof*⟩

theorem *Wilson-Bij*: *zprime p ==> [zfact (p − 1) = −1] (mod p)*
⟨*proof*⟩

end

12 Finite Sets and Finite Sums

theory *Finite2*
imports *Main IntFact Infinite-Set*
begin

These are useful for combinatorial and number-theoretic counting arguments.

12.1 Useful properties of sums and products

lemma *setsum-same-function-zcong*:
assumes *a*: ∀ *x* ∈ *S*. [*f x* = *g x*](*mod m*)
shows [*setsum f S* = *setsum g S*] (*mod m*)
⟨*proof*⟩

lemma *setprod-same-function-zcong*:
assumes *a*: ∀ *x* ∈ *S*. [*f x* = *g x*](*mod m*)
shows [*setprod f S* = *setprod g S*] (*mod m*)
⟨*proof*⟩

lemma *setsum-const*: *finite X ==> setsum (%x. (c :: int)) X = c * int(card X)*
⟨*proof*⟩

lemma *setsum-const2*: *finite X ==> int (setsum (%x. (c :: nat)) X) =*
*int(c) * int(card X)*
⟨*proof*⟩

lemma *setsum-const-mult*: *finite A ==> setsum (%x. c * ((f x)::int)) A =*
*c * setsum f A*
⟨*proof*⟩

12.2 Cardinality of explicit finite sets

lemma *finite-surjI*: *[| B ⊆ f ‘ A; finite A |] ==> finite B*
⟨*proof*⟩

lemma *bdd-nat-set-l-finite*: *finite {y::nat . y < x}*
⟨*proof*⟩

lemma *bdd-nat-set-le-finite*: *finite {y::nat . y ≤ x}*
⟨*proof*⟩

lemma *bdd-int-set-l-finite*: *finite {x::int. 0 ≤ x & x < n}*
⟨*proof*⟩

lemma *bdd-int-set-le-finite*: *finite {x::int. 0 ≤ x & x ≤ n}*
⟨*proof*⟩

lemma *bdd-int-set-l-l-finite*: *finite {x::int. 0 < x & x < n}*
⟨*proof*⟩

lemma *bdd-int-set-l-le-finite*: *finite {x::int. 0 < x & x ≤ n}*
⟨*proof*⟩

lemma *card-bdd-nat-set-l*: *card {y::nat . y < x} = x*
⟨*proof*⟩

lemma *card-bdd-nat-set-le*: *card { y::nat. y ≤ x} = Suc x*
⟨*proof*⟩

lemma *card-bdd-int-set-l*: *0 ≤ (n::int) ==> card {y. 0 ≤ y & y < n} = nat n*
⟨*proof*⟩

lemma *card-bdd-int-set-le*: *0 ≤ (n::int) ==> card {y. 0 ≤ y & y ≤ n} =*
nat n + 1
⟨*proof*⟩

lemma *card-bdd-int-set-l-le*: *0 ≤ (n::int) ==>*
card {x. 0 < x & x ≤ n} = nat n

⟨*proof*⟩

lemma *card-bdd-int-set-l-l*: $0 < (n::int) ==>$
$card\ \{x.\ 0 < x\ \&\ x < n\} = nat\ n - 1$
⟨*proof*⟩

lemma *int-card-bdd-int-set-l-l*: $0 < n ==>$
$int(card\ \{x.\ 0 < x\ \&\ x < n\}) = n - 1$
⟨*proof*⟩

lemma *int-card-bdd-int-set-l-le*: $0 \leq n ==>$
$int(card\ \{x.\ 0 < x\ \&\ x \leq n\}) = n$
⟨*proof*⟩

12.3 Cardinality of finite cartesian products

Lemmas for counting arguments.

lemma *setsum-bij-eq*: [| *finite* A; *finite* B; $f\ `\ A \subseteq B$; *inj-on* $f\ A$;
$g\ `\ B \subseteq A$; *inj-on* $g\ B$ |] ==> $setsum\ g\ B = setsum\ (g \circ f)\ A$
⟨*proof*⟩

lemma *setprod-bij-eq*: [| *finite* A; *finite* B; $f\ `\ A \subseteq B$; *inj-on* $f\ A$;
$g\ `\ B \subseteq A$; *inj-on* $g\ B$ |] ==> $setprod\ g\ B = setprod\ (g \circ f)\ A$
⟨*proof*⟩

end

13 Integers: Divisibility and Congruences

theory *Int2*
imports *Finite2 WilsonRuss*
begin

definition
$MultInv :: int => int => int$ **where**
$MultInv\ p\ x = x\ \hat{}\ nat\ (p - 2)$

13.1 Useful lemmas about dvd and powers

lemma *zpower-zdvd-prop1*:
$0 < n \Longrightarrow p\ dvd\ y \Longrightarrow p\ dvd\ ((y::int)\ \hat{}\ n)$
⟨*proof*⟩

lemma *zdvd-bounds*: $n\ dvd\ m ==> m \leq (0::int) \mid n \leq m$
⟨*proof*⟩

lemma *zprime-zdvd-zmult-better*: [| *zprime* p; $p\ dvd\ (m * n)$ |] ==>

$(p\ dvd\ m)\ |\ (p\ dvd\ n)$
⟨*proof*⟩

lemma *zpower-zdvd-prop2*:
$zprime\ p \Longrightarrow p\ dvd\ ((y::int)\ \hat{}\ n) \Longrightarrow 0 < n \Longrightarrow p\ dvd\ y$
⟨*proof*⟩

lemma *div-prop1*: $[|\ 0 < z;\ (x::int) < y * z\ |] ==> x\ div\ z < y$
⟨*proof*⟩

lemma *div-prop2*: $[|\ 0 < z;\ (x::int) < (y * z) + z\ |] ==> x\ div\ z \leq y$
⟨*proof*⟩

lemma *zdiv-leq-prop*: $[|\ 0 < y\ |] ==> y * (x\ div\ y) \leq (x::int)$
⟨*proof*⟩

13.2 Useful properties of congruences

lemma *zcong-eq-zdvd-prop*: $[x = 0](mod\ p) = (p\ dvd\ x)$
⟨*proof*⟩

lemma *zcong-id*: $[m = 0]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-shift*: $[a = b]\ (mod\ m) ==> [a + c = b + c]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zpower*: $[x = y](mod\ m) ==> [x\hat{}z = y\hat{}z](mod\ m)$
⟨*proof*⟩

lemma *zcong-eq-trans*: $[|\ [a = b](mod\ m);\ b = c;\ [c = d](mod\ m)\ |] ==>$
$[a = d](mod\ m)$
⟨*proof*⟩

lemma *aux1*: $a - b = (c::int) ==> a = c + b$
⟨*proof*⟩

lemma *zcong-zmult-prop1*: $[a = b](mod\ m) ==> ([c = a * d](mod\ m) =$
$[c = b * d]\ (mod\ m))$
⟨*proof*⟩

lemma *zcong-zmult-prop2*: $[a = b](mod\ m) ==>$
$([c = d * a](mod\ m) = [c = d * b]\ (mod\ m))$
⟨*proof*⟩

lemma *zcong-zmult-prop3*: $[|\ zprime\ p;\ \sim[x = 0]\ (mod\ p);$
$\sim[y = 0]\ (mod\ p)\ |] ==> \sim[x * y = 0]\ (mod\ p)$
⟨*proof*⟩

lemma *zcong-less-eq*: [| $0 < x$; $0 < y$; $0 < m$; $[x = y]$ $(mod\ m)$;
$x < m$; $y < m$ |] ==> $x = y$
⟨*proof*⟩

lemma *zcong-neg-1-impl-ne-1*: [| $2 < p$; $[x = -1]$ $(mod\ p)$ |] ==>
~($[x = 1]$ $(mod\ p)$)
⟨*proof*⟩

lemma *zcong-zero-equiv-div*: $[a = 0]$ $(mod\ m) = (m\ dvd\ a)$
⟨*proof*⟩

lemma *zcong-zprime-prod-zero*: [| *zprime* p; $0 < a$ |] ==>
$[a * b = 0]$ $(mod\ p)$ ==> $[a = 0]$ $(mod\ p)$ | $[b = 0]$ $(mod\ p)$
⟨*proof*⟩

lemma *zcong-zprime-prod-zero-contra*: [| *zprime* p; $0 < a$ |] ==>
~$[a = 0](mod\ p)$ & ~$[b = 0](mod\ p)$ ==> ~$[a * b = 0]$ $(mod\ p)$
⟨*proof*⟩

lemma *zcong-not-zero*: [| $0 < x$; $x < m$ |] ==> ~$[x = 0]$ $(mod\ m)$
⟨*proof*⟩

lemma *zcong-zero*: [| $0 \leq x$; $x < m$; $[x = 0](mod\ m)$ |] ==> $x = 0$
⟨*proof*⟩

lemma *all-relprime-prod-relprime*: [| *finite* A; $\forall x \in A.\ zgcd\ x\ y = 1$ |]
==> $zgcd\ (setprod\ id\ A)\ y = 1$
⟨*proof*⟩

13.3 Some properties of MultInv

lemma *MultInv-prop1*: [| $2 < p$; $[x = y]$ $(mod\ p)$ |] ==>
$[(MultInv\ p\ x) = (MultInv\ p\ y)]$ $(mod\ p)$
⟨*proof*⟩

lemma *MultInv-prop2*: [| $2 < p$; *zprime* p; ~$([x = 0](mod\ p))$ |] ==>
$[(x * (MultInv\ p\ x)) = 1]$ $(mod\ p)$
⟨*proof*⟩

lemma *MultInv-prop2a*: [| $2 < p$; *zprime* p; ~$([x = 0](mod\ p))$ |] ==>
$[(MultInv\ p\ x) * x = 1]$ $(mod\ p)$
⟨*proof*⟩

lemma *aux-1*: $2 < p$ ==> $((nat\ p) - 2) = (nat\ (p - 2))$
⟨*proof*⟩

lemma *aux-2*: $2 < p$ ==> $0 < nat\ (p - 2)$
⟨*proof*⟩

lemma *MultInv-prop3*: [| 2 < p; zprime p; ~([x = 0](mod p)) |] ==>
~([MultInv p x = 0](mod p))
⟨*proof*⟩

lemma *aux--1*: [| 2 < p; zprime p; ~([x = 0](mod p))|] ==>
[(MultInv p (MultInv p x)) = (x * (MultInv p x) *
(MultInv p (MultInv p x)))] (mod p)
⟨*proof*⟩

lemma *aux--2*: [| 2 < p; zprime p; ~([x = 0](mod p))|] ==>
[(x * (MultInv p x) * (MultInv p (MultInv p x))) = x] (mod p)
⟨*proof*⟩

lemma *MultInv-prop4*: [| 2 < p; zprime p; ~([x = 0](mod p)) |] ==>
[(MultInv p (MultInv p x)) = x] (mod p)
⟨*proof*⟩

lemma *MultInv-prop5*: [| 2 < p; zprime p; ~([x = 0](mod p));
~([y = 0](mod p)); [(MultInv p x) = (MultInv p y)] (mod p) |] ==>
[x = y] (mod p)
⟨*proof*⟩

lemma *MultInv-zcong-prop1*: [| 2 < p; [j = k] (mod p) |] ==>
[a * MultInv p j = a * MultInv p k] (mod p)
⟨*proof*⟩

lemma *aux---1*: [j = a * MultInv p k] (mod p) ==>
[j * k = a * MultInv p k * k] (mod p)
⟨*proof*⟩

lemma *aux---2*: [|2 < p; zprime p; ~([k = 0](mod p));
[j * k = a * MultInv p k * k] (mod p) |] ==> [j * k = a] (mod p)
⟨*proof*⟩

lemma *aux---3*: [j * k = a] (mod p) ==> [(MultInv p j) * j * k =
(MultInv p j) * a] (mod p)
⟨*proof*⟩

lemma *aux---4*: [|2 < p; zprime p; ~([j = 0](mod p));
[(MultInv p j) * j * k = (MultInv p j) * a] (mod p) |]
==> [k = a * (MultInv p j)] (mod p)
⟨*proof*⟩

lemma *MultInv-zcong-prop2*: [| 2 < p; zprime p; ~([k = 0](mod p));
~([j = 0](mod p)); [j = a * MultInv p k] (mod p) |] ==>
[k = a * MultInv p j] (mod p)
⟨*proof*⟩

lemma *MultInv-zcong-prop3*: [| 2 < p; zprime p; ~([a = 0](mod p));

~([k = 0](mod p)); ~([j = 0](mod p));
[a * MultInv p j = a * MultInv p k] (mod p) |] ==>
[j = k] (mod p)
⟨proof⟩

end

14 Residue Sets

theory *Residues* **imports** *Int2* **begin**

Define the residue of a set, the standard residue, quadratic residues, and prove some basic properties.

definition
ResSet :: *int* => *int set* => *bool* **where**
ResSet m X = (∀ *y1 y2*. (*y1* ∈ *X* & *y2* ∈ *X* & [*y1* = *y2*] (*mod m*) −−> *y1* = *y2*))

definition
StandardRes :: *int* => *int* => *int* **where**
StandardRes m x = *x mod m*

definition
QuadRes :: *int* => *int* => *bool* **where**
QuadRes m x = (∃ *y*. ([(*y* ^ 2) = *x*] (*mod m*)))

definition
Legendre :: *int* => *int* => *int* **where**
Legendre a p = (*if* ([*a* = 0] (*mod p*)) *then* 0
else if (*QuadRes p a*) *then* 1
else −1)

definition
SR :: *int* => *int set* **where**
SR p = {*x*. (0 ≤ *x*) & (*x* < *p*)}

definition
SRStar :: *int* => *int set* **where**
SRStar p = {*x*. (0 < *x*) & (*x* < *p*)}

14.1 Some useful properties of StandardRes

lemma *StandardRes-prop1*: [*x* = *StandardRes m x*] (*mod m*)
⟨*proof*⟩

lemma *StandardRes-prop2*: 0 < *m* ==> (*StandardRes m x1* = *StandardRes m x2*)

$= ([x1 = x2]\ (mod\ m))$
⟨*proof*⟩

lemma *StandardRes-prop3*: $(^{\sim}[x = 0]\ (mod\ p)) = (^{\sim}(StandardRes\ p\ x = 0))$
⟨*proof*⟩

lemma *StandardRes-prop4*: $2 < m$
$==> [StandardRes\ m\ x * StandardRes\ m\ y = (x * y)]\ (mod\ m)$
⟨*proof*⟩

lemma *StandardRes-lbound*: $0 < p ==> 0 \leq StandardRes\ p\ x$
⟨*proof*⟩

lemma *StandardRes-ubound*: $0 < p ==> StandardRes\ p\ x < p$
⟨*proof*⟩

lemma *StandardRes-eq-zcong*:
$(StandardRes\ m\ x = 0) = ([x = 0](mod\ m))$
⟨*proof*⟩

14.2 Relations between StandardRes, SRStar, and SR

lemma *SRStar-SR-prop*: $x \in SRStar\ p ==> x \in SR\ p$
⟨*proof*⟩

lemma *StandardRes-SR-prop*: $x \in SR\ p ==> StandardRes\ p\ x = x$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop1*: $2 < p ==> (StandardRes\ p\ x \in SRStar\ p)$
$= (^{\sim}[x = 0]\ (mod\ p))$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop1a*: $x \in SRStar\ p ==> {}^{\sim}([x = 0]\ (mod\ p))$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop2*: $[|\ 2 < p;\ zprime\ p;\ x \in SRStar\ p\ |]$
$==> StandardRes\ p\ (MultInv\ p\ x) \in SRStar\ p$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop3*: $x \in SRStar\ p ==> StandardRes\ p\ x = x$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop4*: $[|\ zprime\ p;\ 2 < p;\ x \in SRStar\ p\ |]$
$==> StandardRes\ p\ x \in SRStar\ p$
⟨*proof*⟩

lemma *SRStar-mult-prop1*: $[|\ zprime\ p;\ 2 < p;\ x \in SRStar\ p;\ y \in SRStar\ p|]$
$==> (StandardRes\ p\ (x * y)):SRStar\ p$
⟨*proof*⟩

lemma *SRStar-mult-prop2*: [| *zprime p*; *2 < p*; ~([*a = 0*](*mod p*)); *x ∈ SRStar p* |] ==> *StandardRes p* (*a * MultInv p x*) ∈ *SRStar p*
⟨*proof*⟩

lemma *SRStar-card*: *2 < p* ==> *int*(*card*(*SRStar p*)) = *p − 1*
⟨*proof*⟩

lemma *SRStar-finite*: *2 < p* ==> *finite*(*SRStar p*)
⟨*proof*⟩

14.3 Properties relating ResSets with StandardRes

lemma *aux*: *x mod m = y mod m* ==> [*x = y*] (*mod m*)
⟨*proof*⟩

lemma *StandardRes-inj-on-ResSet*: *ResSet m X* ==> (*inj-on* (*StandardRes m*) *X*)
⟨*proof*⟩

lemma *StandardRes-Sum*: [| *finite X*; *0 < m* |] ==> [*setsum f X = setsum* (*StandardRes m o f*) *X*](*mod m*)
⟨*proof*⟩

lemma *SR-pos*: *0 < m* ==> (*StandardRes m ' X*) ⊆ {*x*. *0* ≤ *x* & *x < m*}
⟨*proof*⟩

lemma *ResSet-finite*: *0 < m* ==> *ResSet m X* ==> *finite X*
⟨*proof*⟩

lemma *mod-mod-is-mod*: [*x = x mod m*](*mod m*)
⟨*proof*⟩

lemma *StandardRes-prod*: [| *finite X*; *0 < m* |] ==> [*setprod f X = setprod* (*StandardRes m o f*) *X*] (*mod m*)
⟨*proof*⟩

lemma *ResSet-image*:
[| *0 < m*; *ResSet m A*; ∀ *x* ∈ *A*. ∀ *y* ∈ *A*. ([*f x = f y*](*mod m*) −−> *x = y*) |] ==>
ResSet m (*f ' A*)
⟨*proof*⟩

14.4 Property for SRStar

lemma *ResSet-SRStar-prop*: *ResSet p* (*SRStar p*)
⟨*proof*⟩

end

15 Parity: Even and Odd Integers

theory *EvenOdd*
imports *Int2*
begin

definition
zOdd :: *int set* **where**
zOdd = {*x*. ∃ *k*. *x* = *2* ∗ *k* + *1*}

definition
zEven :: *int set* **where**
zEven = {*x*. ∃ *k*. *x* = *2* ∗ *k*}

15.1 Some useful properties about even and odd

lemma *zOddI* [*intro?*]: *x* = *2* ∗ *k* + *1* ⟹ *x* ∈ *zOdd*
and *zOddE* [*elim?*]: *x* ∈ *zOdd* ⟹ (!!*k*. *x* = *2* ∗ *k* + *1* ⟹ *C*) ⟹ *C*
⟨*proof*⟩

lemma *zEvenI* [*intro?*]: *x* = *2* ∗ *k* ⟹ *x* ∈ *zEven*
and *zEvenE* [*elim?*]: *x* ∈ *zEven* ⟹ (!!*k*. *x* = *2* ∗ *k* ⟹ *C*) ⟹ *C*
⟨*proof*⟩

lemma *one-not-even*: ~(*1* ∈ *zEven*)
⟨*proof*⟩

lemma *even-odd-conj*: ~(*x* ∈ *zOdd* & *x* ∈ *zEven*)
⟨*proof*⟩

lemma *even-odd-disj*: (*x* ∈ *zOdd* | *x* ∈ *zEven*)
⟨*proof*⟩

lemma *not-odd-impl-even*: ~(*x* ∈ *zOdd*) ==> *x* ∈ *zEven*
⟨*proof*⟩

lemma *odd-mult-odd-prop*: (*x*∗*y*):*zOdd* ==> *x* ∈ *zOdd*
⟨*proof*⟩

lemma *odd-minus-one-even*: *x* ∈ *zOdd* ==> (*x* − *1*):*zEven*
⟨*proof*⟩

lemma *even-div-2-prop1*: *x* ∈ *zEven* ==> (*x mod 2*) = *0*
⟨*proof*⟩

lemma *even-div-2-prop2*: *x* ∈ *zEven* ==> (*2* ∗ (*x div 2*)) = *x*
⟨*proof*⟩

lemma *even-plus-even*: $[|\ x \in zEven;\ y \in zEven\ |] ==> x + y \in zEven$
⟨*proof*⟩

lemma *even-times-either*: $x \in zEven ==> x * y \in zEven$
⟨*proof*⟩

lemma *even-minus-even*: $[|\ x \in zEven;\ y \in zEven\ |] ==> x - y \in zEven$
⟨*proof*⟩

lemma *odd-minus-odd*: $[|\ x \in zOdd;\ y \in zOdd\ |] ==> x - y \in zEven$
⟨*proof*⟩

lemma *even-minus-odd*: $[|\ x \in zEven;\ y \in zOdd\ |] ==> x - y \in zOdd$
⟨*proof*⟩

lemma *odd-minus-even*: $[|\ x \in zOdd;\ y \in zEven\ |] ==> x - y \in zOdd$
⟨*proof*⟩

lemma *odd-times-odd*: $[|\ x \in zOdd;\ \ y \in zOdd\ |] ==> x * y \in zOdd$
⟨*proof*⟩

lemma *odd-iff-not-even*: $(x \in zOdd) = (\sim (x \in zEven))$
⟨*proof*⟩

lemma *even-product*: $x * y \in zEven ==> x \in zEven \mid y \in zEven$
⟨*proof*⟩

lemma *even-diff*: $x - y \in zEven = ((x \in zEven) = (y \in zEven))$
⟨*proof*⟩

lemma *neg-one-even-power*: $[|\ x \in zEven;\ 0 \leq x\ |] ==> (-1::int)\hat{}(nat\ x) = 1$
⟨*proof*⟩

lemma *neg-one-odd-power*: $[|\ x \in zOdd;\ 0 \leq x\ |] ==> (-1::int)\hat{}(nat\ x) = -1$
⟨*proof*⟩

lemma *neg-one-power-parity*: $[|\ 0 \leq x;\ 0 \leq y;\ (x \in zEven) = (y \in zEven)\ |] ==>$
$(-1::int)\hat{}(nat\ x) = (-1::int)\hat{}(nat\ y)$
⟨*proof*⟩

lemma *one-not-neg-one-mod-m*: $2 < m ==> \sim([1 = -1]\ (mod\ m))$
⟨*proof*⟩

lemma *even-div-2-l*: $[|\ y \in zEven;\ x < y\ |] ==> x\ div\ 2 < y\ div\ 2$
⟨*proof*⟩

lemma *even-sum-div-2*: $[|\ x \in zEven;\ y \in zEven\ |] ==> (x + y)\ div\ 2 = x\ div\ 2$

$+$ *y div 2*
⟨*proof*⟩

lemma *even-prod-div-2*: [| *x* ∈ *zEven* |] ==> (*x* ∗ *y*) *div 2* = (*x div 2*) ∗ *y*
⟨*proof*⟩

lemma *zprime-zOdd-eq-grt-2*: *zprime p* ==> (*p* ∈ *zOdd*) = (*2* < *p*)
⟨*proof*⟩

lemma *neg-one-special*: *finite A* ==>
((−*1* :: *int*) ^ *card A*) ∗ (−*1* ^ *card A*) = *1*
⟨*proof*⟩

lemma *neg-one-power*: (−*1*::*int*) ^*n* = *1* | (−*1*::*int*) ^*n* = −*1*
⟨*proof*⟩

lemma *neg-one-power-eq-mod-m*: [| *2* < *m*; [(−*1*::*int*) ^*j* = (−*1*::*int*) ^*k*] (*mod m*)
|]
==> ((−*1*::*int*) ^*j* = (−*1*::*int*) ^*k*)
⟨*proof*⟩

end

16 Euler's criterion

theory *Euler* **imports** *Residues EvenOdd* **begin**

definition
MultInvPair :: *int* => *int* => *int* => *int set* **where**
MultInvPair a p j = {*StandardRes p j*, *StandardRes p* (*a* ∗ (*MultInv p j*))}

definition
SetS :: *int* => *int* => *int set set* **where**
SetS a p = (*MultInvPair a p* ‘ *SRStar p*)

16.1 Property for MultInvPair

lemma *MultInvPair-prop1a*:
[| *zprime p*; *2* < *p*; ~([*a* = *0*](*mod p*));
X ∈ (*SetS a p*); *Y* ∈ (*SetS a p*);
~((*X* ∩ *Y*) = {}) |] ==> *X* = *Y*
⟨*proof*⟩

lemma *MultInvPair-prop1b*:

[| *zprime p*; *2* < *p*; ~([*a* = *0*](*mod p*));
X ∈ (*SetS a p*); *Y* ∈ (*SetS a p*);
X ≠ *Y* |] ==> *X* ∩ *Y* = {}
⟨*proof*⟩

lemma *MultInvPair-prop1c*: [| *zprime p*; *2* < *p*; ~([*a* = *0*](*mod p*)) |] ==>
∀ *X* ∈ *SetS a p*. ∀ *Y* ∈ *SetS a p*. *X* ≠ *Y* −−> *X*∩*Y* = {}
⟨*proof*⟩

lemma *MultInvPair-prop2*: [| *zprime p*; *2* < *p*; ~([*a* = *0*](*mod p*)) |] ==>
Union (*SetS a p*) = *SRStar p*
⟨*proof*⟩

lemma *MultInvPair-distinct*: [| *zprime p*; *2* < *p*; ~([*a* = *0*] (*mod p*));
~([*j* = *0*] (*mod p*));
~(*QuadRes p a*) |] ==>
~([*j* = *a* ∗ *MultInv p j*] (*mod p*))
⟨*proof*⟩

lemma *MultInvPair-card-two*: [| *zprime p*; *2* < *p*; ~([*a* = *0*] (*mod p*));
~(*QuadRes p a*); ~([*j* = *0*] (*mod p*)) |] ==>
card (*MultInvPair a p j*) = *2*
⟨*proof*⟩

16.2 Properties of SetS

lemma *SetS-finite*: *2* < *p* ==> *finite* (*SetS a p*)
⟨*proof*⟩

lemma *SetS-elems-finite*: ∀ *X* ∈ *SetS a p*. *finite X*
⟨*proof*⟩

lemma *SetS-elems-card*: [| *zprime p*; *2* < *p*; ~([*a* = *0*] (*mod p*));
~(*QuadRes p a*) |] ==>
∀ *X* ∈ *SetS a p*. *card X* = *2*
⟨*proof*⟩

lemma *Union-SetS-finite*: *2* < *p* ==> *finite* (*Union* (*SetS a p*))
⟨*proof*⟩

lemma *card-setsum-aux*: [| *finite S*; ∀ *X* ∈ *S*. *finite* (*X*::*int set*);
∀ *X* ∈ *S*. *card X* = *n* |] ==> *setsum card S* = *setsum* (%*x*. *n*) *S*
⟨*proof*⟩

lemma *SetS-card*: [| *zprime p*; *2* < *p*; ~([*a* = *0*] (*mod p*)); ~(*QuadRes p a*) |]
==>
int(*card*(*SetS a p*)) = (*p* − *1*) *div 2*
⟨*proof*⟩

lemma *SetS-setprod-prop*: [| *zprime p*; *2 < p*; ~([*a = 0*] (*mod p*));
~(*QuadRes p a*); *x* ∈ (*SetS a p*) |] ==>
[∏ *x = a*] (*mod p*)
⟨*proof*⟩

lemma *aux1*: [| *0 < x*; (*x*::*int*) *< a*; *x* ≠ (*a − 1*) |] ==> *x < a − 1*
⟨*proof*⟩

lemma *aux2*: [| (*a*::*int*) *< c*; *b < c* |] ==> (*a* ≤ *b* | *b* ≤ *a*)
⟨*proof*⟩

lemma *d22set-induct-old*: (⋀*a*::*int*. *1 < a* ⟶ *P* (*a − 1*) ⟹ *P a*) ⟹ *P x*
⟨*proof*⟩

lemma *SRStar-d22set-prop*: *2 < p* ⟹ (*SRStar p*) = {*1*} ∪ (*d22set* (*p − 1*))
⟨*proof*⟩

lemma *Union-SetS-setprod-prop1*: [| *zprime p*; *2 < p*; ~([*a = 0*] (*mod p*)); ~(*QuadRes p a*) |] ==>
[∏ (*Union* (*SetS a p*)) = *a* ^ *nat* ((*p − 1*) *div 2*)] (*mod p*)
⟨*proof*⟩

lemma *Union-SetS-setprod-prop2*: [| *zprime p*; *2 < p*; ~([*a = 0*](*mod p*)) |] ==>
∏ (*Union* (*SetS a p*)) = *zfact* (*p − 1*)
⟨*proof*⟩

lemma *zfact-prop*: [| *zprime p*; *2 < p*; ~([*a = 0*] (*mod p*)); ~(*QuadRes p a*) |] ==>
[*zfact* (*p − 1*) = *a* ^ *nat* ((*p − 1*) *div 2*)] (*mod p*)
⟨*proof*⟩

Prove the first part of Euler's Criterion:

lemma *Euler-part1*: [| *2 < p*; *zprime p*; ~([*x = 0*](*mod p*));
~(*QuadRes p x*) |] ==>
[*x*^(*nat* (((*p*) *− 1*) *div 2*)) = *−1*](*mod p*)
⟨*proof*⟩

Prove another part of Euler Criterion:

lemma *aux-1*: *0 < p* ==> (*a*::*int*) ^ *nat* (*p*) = *a* ∗ *a* ^ (*nat* (*p*) *− 1*)
⟨*proof*⟩

lemma *aux-2*: [| (*2*::*int*) *< p*; *p* ∈ *zOdd* |] ==> *0* < ((*p − 1*) *div 2*)
⟨*proof*⟩

lemma *Euler-part2*:
[| *2 < p*; *zprime p*; [*a = 0*] (*mod p*) |] ==> [*0 = a* ^ *nat* ((*p − 1*) *div 2*)] (*mod p*)

⟨*proof*⟩

Prove the final part of Euler's Criterion:

lemma *aux--1*: [| ~([*x* = *0*] (*mod p*)); [*y* ^ *2* = *x*] (*mod p*)|] ==> ~(*p dvd y*)
⟨*proof*⟩

lemma *aux--2*: *2* * *nat*((*p* − *1*) *div 2*) = *nat* (*2* * ((*p* − *1*) *div 2*))
⟨*proof*⟩

lemma *Euler-part3*: [| *2* < *p*; *zprime p*; ~([*x* = *0*](*mod p*)); *QuadRes p x* |] ==>

[*x*^(*nat* (((*p*) − *1*) *div 2*)) = *1*](*mod p*)
⟨*proof*⟩

Finally show Euler's Criterion:

theorem *Euler-Criterion*: [| *2* < *p*; *zprime p* |] ==> [(*Legendre a p*) =
a^(*nat* (((*p*) − *1*) *div 2*))] (*mod p*)
⟨*proof*⟩

end

17 Gauss' Lemma

theory *Gauss*
imports *Euler*
begin

locale *GAUSS* =
fixes *p* :: *int*
fixes *a* :: *int*

assumes *p-prime*: *zprime p*
assumes *p-g-2*: *2* < *p*
assumes *p-a-relprime*: ~[*a* = *0*](*mod p*)
assumes *a-nonzero*: *0* < *a*
begin

definition
A :: *int set* **where**
A = {(*x*::*int*). *0* < *x* & *x* ≤ ((*p* − *1*) *div 2*)}

definition
B :: *int set* **where**
B = (%*x*. *x* * *a*) ‘ *A*

definition

$C :: int\ set$ **where**
$C = StandardRes\ p\ `\ B$

definition
$D :: int\ set$ **where**
$D = C \cap \{x.\ x \leq ((p - 1)\ div\ 2)\}$

definition
$E :: int\ set$ **where**
$E = C \cap \{x.\ ((p - 1)\ div\ 2) < x\}$

definition
$F :: int\ set$ **where**
$F = (\%x.\ (p - x))\ `\ E$

17.1 Basic properties of p

lemma *p-odd*: $p \in zOdd$
⟨*proof*⟩

lemma *p-g-0*: $0 < p$
⟨*proof*⟩

lemma *int-nat*: $int\ (nat\ ((p - 1)\ div\ 2)) = (p - 1)\ div\ 2$
⟨*proof*⟩

lemma *p-minus-one-l*: $(p - 1)\ div\ 2 < p$
⟨*proof*⟩

lemma *p-eq*: $p = (2 * (p - 1)\ div\ 2) + 1$
⟨*proof*⟩

lemma (**in** −) *zodd-imp-zdiv-eq*: $x \in zOdd ==> 2 * (x - 1)\ div\ 2 = 2 * ((x - 1)\ div\ 2)$
⟨*proof*⟩

lemma *p-eq2*: $p = (2 * ((p - 1)\ div\ 2)) + 1$
⟨*proof*⟩

17.2 Basic Properties of the Gauss Sets

lemma *finite-A*: $finite\ (A)$
⟨*proof*⟩

lemma *finite-B*: $finite\ (B)$
⟨*proof*⟩

lemma *finite-C*: $finite\ (C)$

⟨*proof*⟩

lemma *finite-D*: *finite* (*D*)
⟨*proof*⟩

lemma *finite-E*: *finite* (*E*)
⟨*proof*⟩

lemma *finite-F*: *finite* (*F*)
⟨*proof*⟩

lemma *C-eq*: $C = D \cup E$
⟨*proof*⟩

lemma *A-card-eq*: *card A* = *nat* ((*p* − *1*) *div 2*)
⟨*proof*⟩

lemma *inj-on-xa-A*: *inj-on* (%*x*. *x* ∗ *a*) *A*
⟨*proof*⟩

lemma *A-res*: *ResSet p A*
⟨*proof*⟩

lemma *B-res*: *ResSet p B*
⟨*proof*⟩

lemma *SR-B-inj*: *inj-on* (*StandardRes p*) *B*
⟨*proof*⟩

lemma *inj-on-pminusx-E*: *inj-on* (%*x*. *p* − *x*) *E*
⟨*proof*⟩

lemma *A-ncong-p*: $x \in A \Longrightarrow \sim[x = 0](mod\ p)$
⟨*proof*⟩

lemma *A-greater-zero*: $x \in A \Longrightarrow 0 < x$
⟨*proof*⟩

lemma *B-ncong-p*: $x \in B \Longrightarrow \sim[x = 0](mod\ p)$
⟨*proof*⟩

lemma *B-greater-zero*: $x \in B \Longrightarrow 0 < x$
⟨*proof*⟩

lemma *C-ncong-p*: $x \in C \Longrightarrow \sim[x = 0](mod\ p)$
⟨*proof*⟩

lemma *C-greater-zero*: $y \in C \Longrightarrow 0 < y$
⟨*proof*⟩

lemma *D-ncong-p*: $x \in D ==> \sim[x = 0](mod\ p)$
⟨*proof*⟩

lemma *E-ncong-p*: $x \in E ==> \sim[x = 0](mod\ p)$
⟨*proof*⟩

lemma *F-ncong-p*: $x \in F ==> \sim[x = 0](mod\ p)$
⟨*proof*⟩

lemma *F-subset*: $F \subseteq \{x.\ 0 < x\ \&\ x \leq ((p - 1)\ div\ 2)\}$
⟨*proof*⟩

lemma *D-subset*: $D \subseteq \{x.\ 0 < x\ \&\ x \leq ((p - 1)\ div\ 2)\}$
⟨*proof*⟩

lemma *F-eq*: $F = \{x.\ \exists y \in A.\ (\ x = p - (StandardRes\ p\ (y*a))\ \&\ (p - 1)\ div\ 2 < StandardRes\ p\ (y*a))\}$
⟨*proof*⟩

lemma *D-eq*: $D = \{x.\ \exists y \in A.\ (\ x = StandardRes\ p\ (y*a)\ \&\ StandardRes\ p\ (y*a) \leq (p - 1)\ div\ 2)\}$
⟨*proof*⟩

lemma *D-leq*: $x \in D ==> x \leq (p - 1)\ div\ 2$
⟨*proof*⟩

lemma *F-ge*: $x \in F ==> x \leq (p - 1)\ div\ 2$
⟨*proof*⟩

lemma *all-A-relprime*: $\forall x \in A.\ zgcd\ x\ p = 1$
⟨*proof*⟩

lemma *A-prod-relprime*: $zgcd\ (setprod\ id\ A)\ p = 1$
⟨*proof*⟩

17.3 Relationships Between Gauss Sets

lemma *B-card-eq-A*: $card\ B = card\ A$
⟨*proof*⟩

lemma *B-card-eq*: $card\ B = nat\ ((p - 1)\ div\ 2)$
⟨*proof*⟩

lemma *F-card-eq-E*: $card\ F = card\ E$
⟨*proof*⟩

lemma *C-card-eq-B*: $card\ C = card\ B$
⟨*proof*⟩

lemma *D-E-disj*: $D \cap E = \{\}$
⟨*proof*⟩

lemma *C-card-eq-D-plus-E*: $card\ C = card\ D + card\ E$
⟨*proof*⟩

lemma *C-prod-eq-D-times-E*: $setprod\ id\ E * setprod\ id\ D = setprod\ id\ C$
⟨*proof*⟩

lemma *C-B-zcong-prod*: $[setprod\ id\ C = setprod\ id\ B]\ (mod\ p)$
⟨*proof*⟩

lemma *F-Un-D-subset*: $(F \cup D) \subseteq A$
⟨*proof*⟩

lemma *F-D-disj*: $(F \cap D) = \{\}$
⟨*proof*⟩

lemma *F-Un-D-card*: $card\ (F \cup D) = nat\ ((p - 1)\ div\ 2)$
⟨*proof*⟩

lemma *F-Un-D-eq-A*: $F \cup D = A$
⟨*proof*⟩

lemma *prod-D-F-eq-prod-A*:
$(setprod\ id\ D) * (setprod\ id\ F) = setprod\ id\ A$
⟨*proof*⟩

lemma *prod-F-zcong*:
$[setprod\ id\ F = ((-1)\ \hat{}\ (card\ E)) * (setprod\ id\ E)]\ (mod\ p)$
⟨*proof*⟩

17.4 Gauss' Lemma

lemma *aux*: $setprod\ id\ A * -1\ \hat{}\ card\ E * a\ \hat{}\ card\ A * -1\ \hat{}\ card\ E = setprod\ id\ A * a\ \hat{}\ card\ A$
⟨*proof*⟩

theorem *pre-gauss-lemma*:
$[a\ \hat{}\ nat((p - 1)\ div\ 2) = (-1)\ \hat{}\ (card\ E)]\ (mod\ p)$
⟨*proof*⟩

theorem *gauss-lemma*: $(Legendre\ a\ p) = (-1)\ \hat{}\ (card\ E)$
⟨*proof*⟩

end

end

18 The law of Quadratic reciprocity

theory *Quadratic-Reciprocity*
imports *Gauss*
begin

Lemmas leading up to the proof of theorem 3.3 in Niven and Zuckerman's presentation.

context *GAUSS*
begin

lemma *QRLemma1*: $a * setsum\ id\ A =$
$p * setsum\ (\%x.\ ((x * a)\ div\ p))\ A + setsum\ id\ D + setsum\ id\ E$
⟨*proof*⟩

lemma *QRLemma2*: $setsum\ id\ A = p * int\ (card\ E) - setsum\ id\ E +$
$setsum\ id\ D$
⟨*proof*⟩

lemma *QRLemma3*: $(a - 1) * setsum\ id\ A =$
$p * (setsum\ (\%x.\ ((x * a)\ div\ p))\ A - int(card\ E)) + 2 * setsum\ id\ E$
⟨*proof*⟩

lemma *QRLemma4*: $a \in zOdd ==>$
$(setsum\ (\%x.\ ((x * a)\ div\ p))\ A \in zEven) = (int(card\ E): zEven)$
⟨*proof*⟩

lemma *QRLemma5*: $a \in zOdd ==>$
$(-1::int)\hat{}(card\ E) = (-1::int)\hat{}(nat(setsum\ (\%x.\ ((x * a)\ div\ p))\ A))$
⟨*proof*⟩

end

lemma *MainQRLemma*: $[|\ a \in zOdd;\ 0 < a;\ \sim([a = 0]\ (mod\ p));\ zprime\ p;\ 2 < p;$
$A = \{x.\ 0 < x\ \&\ x \leq (p - 1)\ div\ 2\}\ |] ==>$
$(Legendre\ a\ p) = (-1::int)\hat{}(nat(setsum\ (\%x.\ ((x * a)\ div\ p))\ A))$
⟨*proof*⟩

18.1 Stuff about S, S1 and S2

locale *QRTEMP* =
fixes p :: *int*
fixes q :: *int*

assumes *p-prime*: *zprime p*
assumes *p-g-2*: $2 < p$

assumes *q-prime*: *zprime q*
assumes *q-g-2*: $2 < q$
assumes *p-neq-q*: $p \neq q$
begin

definition
P-set :: *int set* **where**
P-set = $\{x.\ 0 < x\ \&\ x \leq ((p - 1)\ div\ 2)\ \}$

definition
Q-set :: *int set* **where**
Q-set = $\{x.\ 0 < x\ \&\ x \leq ((q - 1)\ div\ 2)\ \}$

definition
S :: (*int* $*$ *int*) *set* **where**
S = *P-set* <$*$> *Q-set*

definition
S1 :: (*int* $*$ *int*) *set* **where**
S1 = $\{\ (x,\ y).\ (x,\ y)$:*S* & $((p * y) < (q * x))\ \}$

definition
S2 :: (*int* $*$ *int*) *set* **where**
S2 = $\{\ (x,\ y).\ (x,\ y)$:*S* & $((q * x) < (p * y))\ \}$

definition
f1 :: *int* => (*int* $*$ *int*) *set* **where**
f1 j = $\{\ (j1,\ y).\ (j1,\ y)$:*S* & *j1* = *j* & $(y \leq (q * j)\ div\ p)\ \}$

definition
f2 :: *int* => (*int* $*$ *int*) *set* **where**
f2 j = $\{\ (x,\ j1).\ (x,\ j1)$:*S* & *j1* = *j* & $(x \leq (p * j)\ div\ q)\ \}$

lemma *p-fact*: $0 < (p - 1)\ div\ 2$
⟨*proof*⟩

lemma *q-fact*: $0 < (q - 1)\ div\ 2$
⟨*proof*⟩

lemma *pb-neq-qa*: $[|1 \leq b;\ b \leq (q - 1)\ div\ 2\ |] ==>$
$(p * b \neq q * a)$
⟨*proof*⟩

lemma *P-set-finite*: *finite* (*P-set*)
⟨*proof*⟩

lemma *Q-set-finite*: *finite* (*Q-set*)
⟨*proof*⟩

lemma *S-finite*: *finite S*
⟨*proof*⟩

lemma *S1-finite*: *finite S1*
⟨*proof*⟩

lemma *S2-finite*: *finite S2*
⟨*proof*⟩

lemma *P-set-card*: $(p - 1)\ div\ 2 = int\ (card\ (P\text{-}set))$
⟨*proof*⟩

lemma *Q-set-card*: $(q - 1)\ div\ 2 = int\ (card\ (Q\text{-}set))$
⟨*proof*⟩

lemma *S-card*: $((p - 1)\ div\ 2) * ((q - 1)\ div\ 2) = int\ (card(S))$
⟨*proof*⟩

lemma *S1-Int-S2-prop*: $S1 \cap S2 = \{\}$
⟨*proof*⟩

lemma *S1-Union-S2-prop*: $S = S1 \cup S2$
⟨*proof*⟩

lemma *card-sum-S1-S2*: $((p - 1)\ div\ 2) * ((q - 1)\ div\ 2) =$
$int(card(S1)) + int(card(S2))$
⟨*proof*⟩

lemma *aux1a*: $[|\ 0 < a;\ a \leq (p - 1)\ div\ 2;$
$0 < b;\ b \leq (q - 1)\ div\ 2\ |] ==>$
$(p * b < q * a) = (b \leq q * a\ div\ p)$
⟨*proof*⟩

lemma *aux1b*: $[|\ 0 < a;\ a \leq (p - 1)\ div\ 2;$
$0 < b;\ b \leq (q - 1)\ div\ 2\ |] ==>$
$(q * a < p * b) = (a \leq p * b\ div\ q)$
⟨*proof*⟩

lemma (**in** −) *aux2*: $[|\ zprime\ p;\ zprime\ q;\ 2 < p;\ 2 < q\ |] ==>$
$(q * ((p - 1)\ div\ 2))\ div\ p \leq (q - 1)\ div\ 2$
⟨*proof*⟩

lemma *aux3a*: $\forall j \in P\text{-}set.\ int\ (card\ (f1\ j)) = (q * j)\ div\ p$
⟨*proof*⟩

lemma *aux3b*: $\forall j \in Q\text{-}set.\ int\ (card\ (f2\ j)) = (p * j)\ div\ q$
⟨*proof*⟩

lemma *S1-card*: $int\ (card(S1)) = setsum\ (\%j.\ (q * j)\ div\ p)\ P\text{-}set$

⟨*proof*⟩

lemma *S2-card*: *int* (*card*(*S2*)) = *setsum* (%*j*. (*p* ∗ *j*) *div* *q*) *Q-set*
⟨*proof*⟩

lemma *S1-carda*: *int* (*card*(*S1*)) =
setsum (%*j*. (*j* ∗ *q*) *div* *p*) *P-set*
⟨*proof*⟩

lemma *S2-carda*: *int* (*card*(*S2*)) =
setsum (%*j*. (*j* ∗ *p*) *div* *q*) *Q-set*
⟨*proof*⟩

lemma *pq-sum-prop*: (*setsum* (%*j*. (*j* ∗ *p*) *div* *q*) *Q-set*) +
(*setsum* (%*j*. (*j* ∗ *q*) *div* *p*) *P-set*) = ((*p* − *1*) *div* *2*) ∗ ((*q* − *1*) *div* *2*)
⟨*proof*⟩

lemma (**in** −) *pq-prime-neq*: [| *zprime* *p*; *zprime* *q*; *p* ≠ *q* |] ==> (~[*p* = *0*] (*mod* *q*))
⟨*proof*⟩

lemma *QR-short*: (*Legendre* *p* *q*) ∗ (*Legendre* *q* *p*) =
(−*1*::*int*) ^*nat*(((*p* − *1*) *div* *2*)∗((*q* − *1*) *div* *2*))
⟨*proof*⟩

end

theorem *Quadratic-Reciprocity*:
[| *p* ∈ *zOdd*; *zprime* *p*; *q* ∈ *zOdd*; *zprime* *q*;
p ≠ *q* |]
==> (*Legendre* *p* *q*) ∗ (*Legendre* *q* *p*) =
(−*1*::*int*) ^*nat*(((*p* − *1*) *div* *2*)∗((*q* − *1*) *div* *2*))
⟨*proof*⟩

end

19 Pocklington's Theorem for Primes

theory *Pocklington*
imports *Main* *Primes*
begin

definition *modeq*:: *nat* => *nat* => *nat* => *bool* ((*1*[- = -] '(*mod* -')))
where [*a* = *b*] (*mod* *p*) == ((*a* *mod* *p*) = (*b* *mod* *p*))

definition *modneq*:: *nat* => *nat* => *nat* => *bool* ((*1*[- ≠ -] '(*mod* -')))

where $[a \neq b]\ (mod\ p) == ((a\ mod\ p) \neq (b\ mod\ p))$

lemma *modeq-trans*:
$[\![\ [a = b]\ (mod\ p);\ [b = c]\ (mod\ p)\ ]\!] \Longrightarrow [a = c]\ (mod\ p)$
⟨*proof*⟩

lemma *nat-mod-lemma*: **assumes** *xyn*: $[x = y]\ (mod\ n)$ **and** *xy*:$y \leq x$
shows $\exists\, q.\ x = y + n * q$
⟨*proof*⟩

lemma *nat-mod*[*algebra*]: $[x = y]\ (mod\ n) \longleftrightarrow (\exists\, q1\ q2.\ x + n * q1 = y + n * q2)$
⟨*proof*⟩

lemma *prime*: $prime\ p \longleftrightarrow p \neq 0 \wedge p{\neq}1 \wedge (\forall\, m.\ 0 < m \wedge m < p \longrightarrow coprime\ p\ m)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

lemma *finite-number-segment*: $card\ \{\ m.\ 0 < m \wedge m < n\ \} = n - 1$
⟨*proof*⟩

lemma *coprime-mod*: **assumes** *n*: $n \neq 0$ **shows** $coprime\ (a\ mod\ n)\ n \longleftrightarrow coprime\ a\ n$
⟨*proof*⟩

lemma *cong-mod-01*[*simp*,*presburger*]:
$[x = y]\ (mod\ 0) \longleftrightarrow x = y\ [x = y]\ (mod\ 1)\ [x = 0]\ (mod\ n) \longleftrightarrow n\ dvd\ x$
⟨*proof*⟩

lemma *cong-sub-cases*:
$[x = y]\ (mod\ n) \longleftrightarrow (if\ x <= y\ then\ [y - x = 0]\ (mod\ n)\ else\ [x - y = 0]\ (mod\ n))$
⟨*proof*⟩

lemma *cong-mult-lcancel*: **assumes** *an*: $coprime\ a\ n$ **and** *axy*:$[a * x = a * y]\ (mod\ n)$
shows $[x = y]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-mult-rcancel*: **assumes** *an*: $coprime\ a\ n$ **and** *axy*:$[x{*}a = y{*}a]\ (mod\ n)$
shows $[x = y]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-refl*: $[x = x]\ (mod\ n)$ ⟨*proof*⟩

lemma *eq-imp-cong*: $a = b \Longrightarrow [a = b]\ (mod\ n)$ ⟨*proof*⟩

lemma *cong-commute*: $[x = y]\ (mod\ n) \longleftrightarrow [y = x]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-trans*[*trans*]: $[x = y]\ (mod\ n) \Longrightarrow [y = z]\ (mod\ n) \Longrightarrow [x = z]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-add*: **assumes** *xx'*: $[x = x']\ (mod\ n)$ **and** *yy'*:$[y = y']\ (mod\ n)$
shows $[x + y = x' + y']\ (mod\ n)$
⟨*proof*⟩

lemma *cong-mult*: **assumes** *xx'*: $[x = x']\ (mod\ n)$ **and** *yy'*:$[y = y']\ (mod\ n)$
shows $[x * y = x' * y']\ (mod\ n)$
⟨*proof*⟩

lemma *cong-exp*: $[x = y]\ (mod\ n) \Longrightarrow [x\hat{}k = y\hat{}k]\ (mod\ n)$
⟨*proof*⟩
lemma *cong-sub*: **assumes** *xx'*: $[x = x']\ (mod\ n)$ **and** *yy'*: $[y = y']\ (mod\ n)$
and *yx*: $y <= x$ **and** *yx'*: $y' <= x'$
shows $[x - y = x' - y']\ (mod\ n)$
⟨*proof*⟩

lemma *cong-mult-lcancel-eq*: **assumes** *an*: *coprime a n*
shows $[a * x = a * y]\ (mod\ n) \longleftrightarrow [x = y]\ (mod\ n)$ (**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

lemma *cong-mult-rcancel-eq*: **assumes** *an*: *coprime a n*
shows $[x * a = y * a]\ (mod\ n) \longleftrightarrow [x = y]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-add-lcancel-eq*: $[a + x = a + y]\ (mod\ n) \longleftrightarrow [x = y]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-add-rcancel-eq*: $[x + a = y + a]\ (mod\ n) \longleftrightarrow [x = y]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-add-rcancel*: $[x + a = y + a]\ (mod\ n) \Longrightarrow [x = y]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-add-lcancel*: $[a + x = a + y]\ (mod\ n) \Longrightarrow [x = y]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-add-lcancel-eq-0*: $[a + x = a]\ (mod\ n) \longleftrightarrow [x = 0]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-add-rcancel-eq-0*: $[x + a = a]\ (mod\ n) \longleftrightarrow [x = 0]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-imp-eq*: **assumes** *xn*: $x < n$ **and** *yn*: $y < n$ **and** *xy*: $[x = y]\ (mod\ n)$
shows $x = y$
⟨*proof*⟩

lemma *cong-divides-modulus*: $[x = y]\ (mod\ m) \Longrightarrow n\ dvd\ m \Longrightarrow [x = y]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-0-divides*: $[x = 0]\ (mod\ n) \longleftrightarrow n\ dvd\ x$ ⟨*proof*⟩

lemma *cong-1-divides*:$[x = 1]\ (mod\ n) \Longrightarrow n\ dvd\ x - 1$
⟨*proof*⟩

lemma *cong-divides*: $[x = y]\ (mod\ n) \Longrightarrow n\ dvd\ x \longleftrightarrow n\ dvd\ y$
⟨*proof*⟩

lemma *cong-coprime*: **assumes** *xy*: $[x = y]\ (mod\ n)$
shows $coprime\ n\ x \longleftrightarrow coprime\ n\ y$
⟨*proof*⟩

lemma *cong-mod*: $\sim(n = 0) \Longrightarrow [a\ mod\ n = a]\ (mod\ n)$ ⟨*proof*⟩

lemma *mod-mult-cong*: $\sim(a = 0) \Longrightarrow \sim(b = 0)$
$\Longrightarrow [x\ mod\ (a * b) = y]\ (mod\ a) \longleftrightarrow [x = y]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-mod-mult*: $[x = y]\ (mod\ n) \Longrightarrow m\ dvd\ n \Longrightarrow [x = y]\ (mod\ m)$
⟨*proof*⟩

lemma *cong-le*: $y <= x \Longrightarrow [x = y]\ (mod\ n) \longleftrightarrow (\exists\, q.\ x = q * n + y)$
⟨*proof*⟩

lemma *cong-to-1*: $[a = 1]\ (mod\ n) \longleftrightarrow a = 0 \wedge n = 1 \vee (\exists\, m.\ a = 1 + m * n)$
⟨*proof*⟩

lemma *cong-solve*: **assumes** *an*: $coprime\ a\ n$ **shows** $\exists\, x.\ [a * x = b]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-solve-unique*: **assumes** *an*: $coprime\ a\ n$ **and** *nz*: $n \neq 0$

shows $\exists!x.\ x < n \wedge [a * x = b]\ (mod\ n)$
⟨*proof*⟩

lemma *cong-solve-unique-nontrivial*:
assumes *p*: *prime p* **and** *pa*: *coprime p a* **and** *x0*: $0 < x$ **and** *xp*: $x < p$
shows $\exists!y.\ 0 < y \wedge y < p \wedge [x * y = a]\ (mod\ p)$
⟨*proof*⟩
lemma *cong-unique-inverse-prime*:
assumes *p*: *prime p* **and** *x0*: $0 < x$ **and** *xp*: $x < p$
shows $\exists!y.\ 0 < y \wedge y < p \wedge [x * y = 1]\ (mod\ p)$
⟨*proof*⟩

lemma *cong-chinese*:
assumes *ab*: *coprime a b* **and** *xya*: $[x = y]\ (mod\ a)$
and *xyb*: $[x = y]\ (mod\ b)$
shows $[x = y]\ (mod\ a*b)$
⟨*proof*⟩

lemma *chinese-remainder-unique*:
assumes *ab*: *coprime a b* **and** *az*: $a \neq 0$ **and** *bz*: $b \neq 0$
shows $\exists!x.\ x < a * b \wedge [x = m]\ (mod\ a) \wedge [x = n]\ (mod\ b)$
⟨*proof*⟩

lemma *chinese-remainder-coprime-unique*:
assumes *ab*: *coprime a b* **and** *az*: $a \neq 0$ **and** *bz*: $b \neq 0$
and *ma*: *coprime m a* **and** *nb*: *coprime n b*
shows $\exists!x.\ coprime\ x\ (a * b) \wedge x < a * b \wedge [x = m]\ (mod\ a) \wedge [x = n]\ (mod\ b)$
⟨*proof*⟩

definition *phi-def*: $\varphi\ n = card\ \{\ m.\ 0 < m \wedge m <= n \wedge coprime\ m\ n\ \}$

lemma *phi-0*[*simp*]: $\varphi\ 0 = 0$
⟨*proof*⟩

lemma *phi-finite*[*simp*]: *finite* $(\{\ m.\ 0 < m \wedge m <= n \wedge coprime\ m\ n\ \})$
⟨*proof*⟩

declare *coprime-1*[*presburger*]
lemma *phi-1*[*simp*]: $\varphi\ 1 = 1$
⟨*proof*⟩

lemma [*simp*]: $\varphi\ (Suc\ 0) = Suc\ 0$ ⟨*proof*⟩

lemma *phi-alt*: $\varphi(n) = card\ \{\ m.\ coprime\ m\ n \wedge m < n\}$

⟨*proof*⟩

lemma *phi-finite-lemma*[*simp*]: *finite* {*m*. *coprime m n* ∧ *m* < *n*} (**is** *finite ?S*)
⟨*proof*⟩

lemma *phi-another*: **assumes** *n*: *n*≠*1*
shows φ *n* = *card* {*m*. *0* < *m* ∧ *m* < *n* ∧ *coprime m n* }
⟨*proof*⟩

lemma *phi-limit*: φ *n* ≤ *n*
⟨*proof*⟩

lemma *stupid*[*simp*]: {*m*. (*0*::*nat*) < *m* ∧ *m* < *n*} = {*1*..<*n*}
⟨*proof*⟩

lemma *phi-limit-strong*: **assumes** *n*: *n*≠*1*
shows φ(*n*) ≤ *n* − *1*
⟨*proof*⟩

lemma *phi-lowerbound-1-strong*: **assumes** *n*: *n* ≥ *1*
shows φ(*n*) ≥ *1*
⟨*proof*⟩

lemma *phi-lowerbound-1*: *2* <= *n* ==> *1* <= φ(*n*)
⟨*proof*⟩

lemma *phi-lowerbound-2*: **assumes** *n*: *3* <= *n* **shows** *2* <= φ (*n*)
⟨*proof*⟩

lemma *phi-prime*: φ *n* = *n* − *1* ∧ *n*≠*0* ∧ *n*≠*1* ⟷ *prime n*
⟨*proof*⟩

lemma *phi-multiplicative*: **assumes** *ab*: *coprime a b*
shows φ (*a* ∗ *b*) = φ *a* ∗ φ *b*
⟨*proof*⟩

lemma *nproduct-mod*:
assumes *fS*: *finite S* **and** *n0*: *n* ≠ *0*
shows [*setprod* (λ*m*. *a*(*m*) *mod n*) *S* = *setprod a S*] (*mod n*)
⟨*proof*⟩

lemma *nproduct-cmul*:
assumes *fS*:*finite S*
shows *setprod* (λ*m*. (*c*::'*a*::{*comm-monoid-mult*})∗ *a*(*m*)) *S* = *c* ^ (*card S*) ∗

setprod a S
⟨*proof*⟩

lemma *coprime-nproduct*:
assumes *fS*: *finite S* **and** *Sn*: ∀ *x*∈*S*. *coprime n* (*a x*)
shows *coprime n* (*setprod a S*)
⟨*proof*⟩

lemma *fermat-little*: **assumes** *an*: *coprime a n*
shows [*a* ^ (*φ n*) = *1*] (*mod n*)
⟨*proof*⟩

lemma *fermat-little-prime*: **assumes** *p*: *prime p* **and** *ap*: *coprime a p*
shows [*a* ^ (*p* − *1*) = *1*] (*mod p*)
⟨*proof*⟩

lemma *lucas-coprime-lemma*:
assumes *m*: *m*≠*0* **and** *am*: [*a*^*m* = *1*] (*mod n*)
shows *coprime a n*
⟨*proof*⟩

lemma *lucas-weak*:
assumes *n*: *n* ≥ *2* **and** *an*:[*a*^(*n* − *1*) = *1*] (*mod n*)
and *nm*: ∀ *m*. *0* <*m* ∧ *m* < *n* − *1* ⟶ ¬ [*a*^*m* = *1*] (*mod n*)
shows *prime n*
⟨*proof*⟩

lemma *nat-exists-least-iff*: (∃ (*n*::*nat*). *P n*) ⟷ (∃ *n*. *P n* ∧ (∀ *m* < *n*. ¬ *P m*))
(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

lemma *nat-exists-least-iff'*: (∃ (*n*::*nat*). *P n*) ⟷ (*P* (*Least P*) ∧ (∀ *m* < (*Least P*). ¬ *P m*))
(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

lemma *power-mod*: ((*x*::*nat*) *mod m*) ^*n mod m* = *x* ^*n mod m*
⟨*proof*⟩

lemma *lucas*:
assumes *n2*: *n* ≥ *2* **and** *an1*: [*a*^(*n* − *1*) = *1*] (*mod n*)
and *pn*: ∀ *p*. *prime p* ∧ *p dvd n* − *1* ⟶ ¬ [*a*^((*n* − *1*) *div p*) = *1*] (*mod n*)
shows *prime n*
⟨*proof*⟩

definition *ord n a* = (*if coprime n a then Least* (λ*d*. *d* > *0* ∧ [*a* ^*d* = *1*] (*mod n*)) *else 0*)

lemma *coprime-ord*:
assumes *na*: *coprime n a*
shows *ord n a* > *0* ∧ [*a* ^(*ord n a*) = *1*] (*mod n*) ∧ (∀ *m*. *0* < *m* ∧ *m* < *ord n a* ⟶ ¬ [*a*^ *m* = *1*] (*mod n*))
⟨*proof*⟩

lemma *ord-works*:
[*a* ^ (*ord n a*) = *1*] (*mod n*) ∧ (∀ *m*. *0* < *m* ∧ *m* < *ord n a* ⟶ ~[*a*^ *m* = *1*] (*mod n*))
⟨*proof*⟩

lemma *ord*: [*a*^(*ord n a*) = *1*] (*mod n*) ⟨*proof*⟩
lemma *ord-minimal*: *0* < *m* ⟹ *m* < *ord n a* ⟹ ~[*a*^*m* = *1*] (*mod n*)
⟨*proof*⟩
lemma *ord-eq-0*: *ord n a* = *0* ⟷ ~*coprime n a*
⟨*proof*⟩

lemma *ord-divides*:
[*a* ^ *d* = *1*] (*mod n*) ⟷ *ord n a dvd d* (**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

lemma *order-divides-phi*: *coprime n a* ⟹ *ord n a dvd* φ *n*
⟨*proof*⟩
lemma *order-divides-expdiff*:
assumes *na*: *coprime n a*
shows [*a*^*d* = *a*^*e*] (*mod n*) ⟷ [*d* = *e*] (*mod* (*ord n a*))
⟨*proof*⟩

lemma *prime-prime-factor*:
prime n ⟷ *n* ≠ *1*∧ (∀ *p*. *prime p* ∧ *p dvd n* ⟶ *p* = *n*)
⟨*proof*⟩

lemma *prime-divisor-sqrt*:
prime n ⟷ *n* ≠ *1* ∧ (∀ *d*. *d dvd n* ∧ *d*^*2* ≤ *n* ⟶ *d* = *1*)
⟨*proof*⟩
lemma *prime-prime-factor-sqrt*:
prime n ⟷ *n* ≠ *0* ∧ *n* ≠ *1* ∧ ¬ (∃ *p*. *prime p* ∧ *p dvd n* ∧ *p*^*2* ≤ *n*)
(**is** *?lhs* ⟷*?rhs*)
⟨*proof*⟩

lemma *pocklington-lemma*:
assumes *n*: $n \geq 2$ **and** *nqr*: $n - 1 = q*r$ **and** *an*: $[a\hat{}\ (n - 1) = 1]\ (mod\ n)$
and *aq*:$\forall\, p.\ prime\ p \wedge p\ dvd\ q \longrightarrow coprime\ (a\hat{}\ ((n - 1)\ div\ p) - 1)\ n$
and *pp*: *prime p* **and** *pn*: *p dvd n*
shows $[p = 1]\ (mod\ q)$
⟨*proof*⟩

lemma *pocklington*:
assumes *n*: $n \geq 2$ **and** *nqr*: $n - 1 = q*r$ **and** *sqr*: $n \leq q\hat{}2$
and *an*: $[a\hat{}\ (n - 1) = 1]\ (mod\ n)$
and *aq*:$\forall\, p.\ prime\ p \wedge p\ dvd\ q \longrightarrow coprime\ (a\hat{}\ ((n - 1)\ div\ p) - 1)\ n$
shows *prime n*
⟨*proof*⟩

lemma *pocklington-alt*:
assumes *n*: $n \geq 2$ **and** *nqr*: $n - 1 = q*r$ **and** *sqr*: $n \leq q\hat{}2$
and *an*: $[a\hat{}\ (n - 1) = 1]\ (mod\ n)$
and *aq*:$\forall\, p.\ prime\ p \wedge p\ dvd\ q \longrightarrow (\exists\, b.\ [a\hat{}((n - 1)\ div\ p) = b]\ (mod\ n) \wedge coprime\ (b - 1)\ n)$
shows *prime n*
⟨*proof*⟩

definition $primefact\ ps\ n = (foldr\ op *\ ps\ \ 1 = n \wedge (\forall\, p{\in}\ set\ ps.\ prime\ p))$

lemma *primefact*: **assumes** *n*: $n \neq 0$
shows $\exists\, ps.\ primefact\ ps\ n$
⟨*proof*⟩

lemma *primefact-contains*:
assumes *pf*: *primefact ps n* **and** *p*: *prime p* **and** *pn*: *p dvd n*
shows $p \in set\ ps$
⟨*proof*⟩

lemma *primefact-variant*: $primefact\ ps\ n \longleftrightarrow foldr\ op *\ ps\ 1 = n \wedge list\text{-}all\ prime\ ps$ ⟨*proof*⟩

lemma *lucas-primefact*:
assumes *n*: $n \geq 2$ **and** *an*: $[a\hat{}(n - 1) = 1]\ (mod\ n)$
and *psn*: $foldr\ op *\ ps\ 1 = n - 1$
and *psp*: $list\text{-}all\ (\lambda p.\ prime\ p \wedge \neg\ [a\hat{}((n - 1)\ div\ p) = 1]\ (mod\ n))\ ps$
shows *prime n*
⟨*proof*⟩

lemma *mod-le*: **assumes** n: $n \neq (0::nat)$ **shows** $m\ mod\ n \leq m$
⟨*proof*⟩

lemma *pocklington-primefact*:
  assumes n: $n \geq 2$ **and** *qrn*: $q*r = n - 1$ **and** *nq2*: $n \leq q\hat{}2$
  and *arnb*: $(a\hat{}r)\ mod\ n = b$ **and** *psq*: $foldr\ op *\ ps\ 1 = q$
  and *bqn*: $(b\hat{}q)\ mod\ n = 1$
  and *psp*: $list\text{-}all\ (\lambda p.\ prime\ p \wedge coprime\ ((b\hat{}(q\ div\ p))\ mod\ n - 1)\ n)\ ps$
  shows *prime n*
⟨*proof*⟩

end