

ZF

Lawrence C Paulson and others

June 21, 2010

Contents

1	ZF: Zermelo-Fraenkel Set Theory	13
1.1	Substitution	18
1.2	Bounded universal quantifier	19
1.3	Bounded existential quantifier	19
1.4	Rules for subsets	20
1.5	Rules for equality	21
1.6	Rules for Replace – the derived form of replacement	21
1.7	Rules for RepFun	22
1.8	Rules for Collect – forming a subset by separation	22
1.9	Rules for Unions	23
1.10	Rules for Unions of families	23
1.11	Rules for the empty set	23
1.12	Rules for Inter	24
1.13	Rules for Intersections of families	24
1.14	Rules for Powersets	24
1.15	Cantor’s Theorem: There is no surjection from a set to its powerset.	25
2	upair: Unordered Pairs	25
2.1	Unordered Pairs: constant <i>Upair</i>	25
2.2	Rules for Binary Union, Defined via <i>Upair</i>	25
2.3	Rules for Binary Intersection, Defined via <i>Upair</i>	26
2.4	Rules for Set Difference, Defined via <i>Upair</i>	26
2.5	Rules for <i>cons</i>	27
2.6	Singletons	27
2.7	Descriptions	27
2.8	Conditional Terms: <i>if-then-else</i>	28
2.9	Consequences of Foundation	29
2.10	Rules for Successor	30
2.11	Miniscoping of the Bounded Universal Quantifier	30
2.12	Miniscoping of the Bounded Existential Quantifier	31

2.13	Miniscoping of the Replacement Operator	32
2.14	Miniscoping of Unions	32
2.15	Miniscoping of Intersections	33
2.16	Other simprules	34
3	pair: Ordered Pairs	34
3.1	Sigma: Disjoint Union of a Family of Sets	35
3.2	Projections <i>fst</i> and <i>snd</i>	36
3.3	The Eliminator, <i>split</i>	36
3.4	A version of <i>split</i> for Formulae: Result Type <i>o</i>	36
4	equalities: Basic Equalities and Inclusions	37
4.1	Bounded Quantifiers	37
4.2	Converse of a Relation	38
4.3	Finite Set Constructions Using <i>cons</i>	38
4.4	Binary Intersection	40
4.5	Binary Union	41
4.6	Set Difference	42
4.7	Big Union and Intersection	44
4.8	Unions and Intersections of Families	45
4.9	Image of a Set under a Function or Relation	51
4.10	Inverse Image of a Set under a Function or Relation	52
4.11	Powerset Operator	54
4.12	RepFun	55
4.13	Collect	55
5	Fixedpt: Least and Greatest Fixed Points; the Knaster-Tarski Theorem	56
5.1	Monotone Operators	56
5.2	Proof of Knaster-Tarski Theorem using <i>lfp</i>	57
5.3	General Induction Rule for Least Fixedpoints	58
5.4	Proof of Knaster-Tarski Theorem using <i>gfp</i>	59
5.5	Coinduction Rules for Greatest Fixed Points	59
6	Bool: Booleans in Zermelo-Fraenkel Set Theory	60
6.1	Laws About 'not'	62
6.2	Laws About 'and'	63
6.3	Laws About 'or'	63
7	Sum: Disjoint Sums	64
7.1	Rules for the <i>Part</i> Primitive	64
7.2	Rules for Disjoint Sums	65
7.3	The Eliminator: <i>case</i>	66
7.4	More Rules for <i>Part</i> (<i>A</i> , <i>h</i>)	67

8	func: Functions, Function Spaces, Lambda-Abstraction	67
8.1	The Pi Operator: Dependent Function Space	68
8.2	Function Application	68
8.3	Lambda Abstraction	70
8.4	Extensionality	71
8.5	Images of Functions	72
8.6	Properties of $\text{restrict}(f, A)$	72
8.7	Unions of Functions	73
8.8	Domain and Range of a Function or Relation	74
8.9	Extensions of Functions	74
8.10	Function Updates	75
8.11	Monotonicity Theorems	76
8.11.1	Replacement in its Various Forms	76
8.11.2	Standard Products, Sums and Function Spaces	76
8.11.3	Converse, Domain, Range, Field	77
8.11.4	Images	77
9	QPair: Quine-Inspired Ordered Pairs and Disjoint Sums	78
9.1	Quine ordered pairing	79
9.1.1	QSigma: Disjoint union of a family of sets Generalizes Cartesian product	79
9.1.2	Projections: qfst , qsnd	80
9.1.3	Eliminator: qsplit	80
9.1.4	qsplit for predicates: result type o	81
9.1.5	qconverse	81
9.2	The Quine-inspired notion of disjoint sum	82
9.2.1	Eliminator – qcase	83
9.2.2	Monotonicity	83
10	Perm: Injections, Surjections, Bijections, Composition	84
10.1	Surjections	85
10.2	Injections	85
10.3	Bijections	86
10.4	Identity Function	86
10.5	Converse of a Function	87
10.6	Converses of Injections, Surjections, Bijections	87
10.7	Composition of Two Relations	88
10.8	Domain and Range – see Suppes, Section 3.1	88
10.9	Other Results	88
10.10	Composition Preserves Functions, Injections, and Surjections	89
10.11	Dual Properties of inj and surj	90
10.11.1	Inverses of Composition	90
10.11.2	Proving that a Function is a Bijection	90
10.11.3	Unions of Functions	91

10.11.4 Restrictions as Surjections and Bijections	91
10.11.5 Lemmas for Ramsey's Theorem	91
11 Transl: Relations: Their General Properties and Transitive Closure	92
11.1 General properties of relations	93
11.1.1 irreflexivity	93
11.1.2 symmetry	93
11.1.3 antisymmetry	93
11.1.4 transitivity	93
11.2 Transitive closure of a relation	94
12 WF: Well-Founded Recursion	97
12.1 Well-Founded Relations	98
12.1.1 Equivalences between <i>wf</i> and <i>wf-on</i>	98
12.1.2 Introduction Rules for <i>wf-on</i>	98
12.1.3 Well-founded Induction	99
12.2 Basic Properties of Well-Founded Relations	100
12.3 The Predicate <i>is-recfun</i>	100
12.4 Recursion: Main Existence Lemma	101
12.5 Unfolding <i>wftrec</i> (<i>r</i> , <i>a</i> , <i>H</i>)	101
12.5.1 Removal of the Premise <i>trans</i> (<i>r</i>)	102
13 Ordinal: Transitive Sets and Ordinals	102
13.1 Rules for Transset	103
13.1.1 Three Neat Characterisations of Transset	103
13.1.2 Consequences of Downwards Closure	103
13.1.3 Closure Properties	104
13.2 Lemmas for Ordinals	104
13.3 The Construction of Ordinals: 0, succ, Union	105
13.4 \leq is 'less Than' for Ordinals	105
13.5 Natural Deduction Rules for Memrel	107
13.6 Transfinite Induction	108
13.6.1 Proving That \leq is a Linear Ordering on the Ordinals	108
13.6.2 Some Rewrite Rules for \leq , le	109
13.7 Results about Less-Than or Equals	109
13.7.1 Transitivity Laws	110
13.7.2 Union and Intersection	111
13.8 Results about Limits	112
13.9 Limit Ordinals – General Properties	113
13.9.1 Traditional 3-Way Case Analysis on Ordinals	113

14 OrdQuant: Special quantifiers	114
14.1 Quantifiers and union operator for ordinals	114
14.1.1 simplification of the new quantifiers	115
14.1.2 Union over ordinals	115
14.1.3 universal quantifier for ordinals	116
14.1.4 existential quantifier for ordinals	117
14.1.5 Rules for Ordinal-Indexed Unions	117
14.2 Quantification over a class	118
14.2.1 Relativized universal quantifier	118
14.2.2 Relativized existential quantifier	119
14.2.3 One-point rule for bounded quantifiers	120
14.2.4 Sets as Classes	121
15 Nat-ZF: The Natural numbers As a Least Fixed Point	121
15.1 Injectivity Properties and Induction	123
15.2 Variations on Mathematical Induction	124
15.3 <i>quasinat</i> : to allow a case-split rule for <i>nat-case</i>	125
15.4 Recursion on the Natural Numbers	125
16 Inductive-ZF: Inductive and Coinductive Definitions	126
17 Epsilon: Epsilon Induction and Recursion	127
17.1 Basic Closure Properties	127
17.2 Leastness of <i>eclose</i>	128
17.3 Epsilon Recursion	128
17.4 Rank	130
17.5 Corollaries of Leastness	131
18 Order: Partial and Total Orderings: Basic Definitions and Properties	132
18.1 Immediate Consequences of the Definitions	133
18.2 Restricting an Ordering's Domain	134
18.3 Empty and Unit Domains	135
18.3.1 Relations over the Empty Set	135
18.3.2 The Empty Relation Well-Orders the Unit Set	136
18.4 Order-Isomorphisms	136
18.5 Main results of Kunen, Chapter 1 section 6	138
18.6 Towards Kunen's Theorem 6.3: Linearity of the Similarity Relation	139
18.7 Miscellaneous Results by Krzysztof Grabczewski	140
18.8 Lemmas for the Reflexive Orders	141

19 OrderArith: Combining Orderings: Foundations of Ordinal Arithmetic	142
19.1 Addition of Relations – Disjoint Sum	142
19.1.1 Rewrite rules. Can be used to obtain introduction rules	142
19.1.2 Elimination Rule	143
19.1.3 Type checking	143
19.1.4 Linearity	143
19.1.5 Well-foundedness	143
19.1.6 An <i>ord-iso</i> congruence law	144
19.1.7 Associativity	144
19.2 Multiplication of Relations – Lexicographic Product	144
19.2.1 Rewrite rule. Can be used to obtain introduction rules	144
19.2.2 Type checking	145
19.2.3 Linearity	145
19.2.4 Well-foundedness	145
19.2.5 An <i>ord-iso</i> congruence law	145
19.2.6 Distributive law	146
19.2.7 Associativity	146
19.3 Inverse Image of a Relation	146
19.3.1 Rewrite rule	146
19.3.2 Type checking	146
19.3.3 Partial Ordering Properties	147
19.3.4 Linearity	147
19.3.5 Well-foundedness	147
19.4 Every well-founded relation is a subset of some inverse image of an ordinal	148
19.5 Other Results	148
19.5.1 The Empty Relation	149
19.5.2 The "measure" relation is useful with wfrec	149
19.5.3 Well-foundedness of Unions	149
19.5.4 Bijections involving Powersets	150
20 OrderType: Order Types and Ordinal Arithmetic	150
20.1 Proofs needing the combination of Ordinal.thy and Order.thy	151
20.2 Ordermap and ordertype	152
20.2.1 Unfolding of ordermap	152
20.2.2 Showing that ordermap, ordertype yield ordinals	152
20.2.3 ordermap preserves the orderings in both directions	152
20.2.4 Isomorphisms involving ordertype	153
20.2.5 Basic equalities for ordertype	153
20.2.6 A fundamental unfolding law for ordertype.	153
20.3 Alternative definition of ordinal	154
20.4 Ordinal Addition	154
20.4.1 Order Type calculations for radd	154

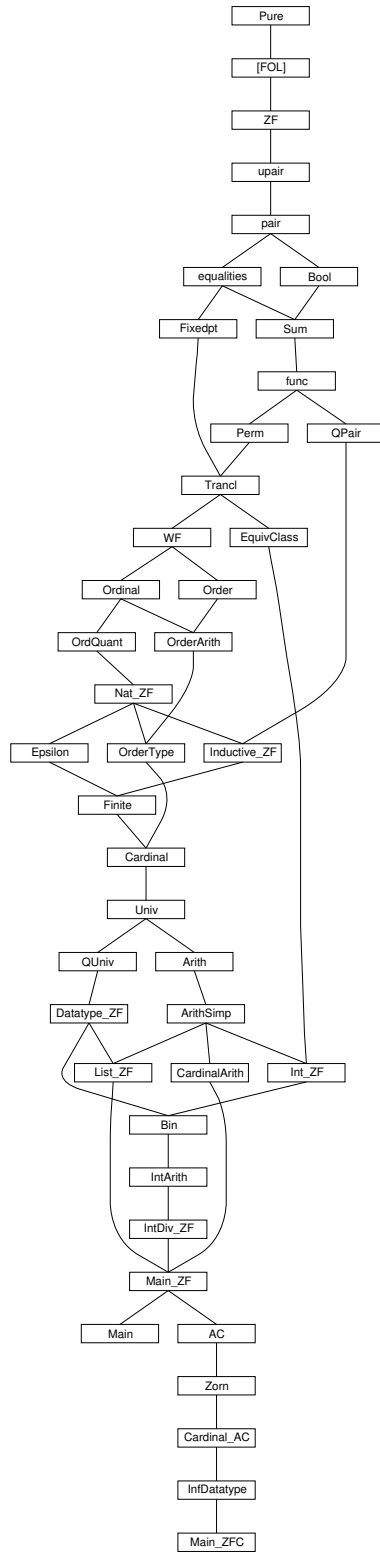
20.4.2	ordify: trivial coercion to an ordinal	155
20.4.3	Basic laws for ordinal addition	155
20.4.4	Ordinal addition with successor – via associativity! . . .	157
20.5	Ordinal Subtraction	158
20.6	Ordinal Multiplication	159
20.6.1	A useful unfolding law	159
20.6.2	Basic laws for ordinal multiplication	159
20.6.3	Ordering/monotonicity properties of ordinal multipli- cation	160
20.7	The Relation Lt	161
21	Finite: Finite Powerset Operator and Finite Function Space	161
21.1	Finite Powerset Operator	162
21.2	Finite Function Space	163
21.3	The Contents of a Singleton Set	164
22	Cardinal: Cardinal Numbers Without the Axiom of Choice	164
22.1	The Schroeder-Bernstein Theorem	165
22.2	lesspoll: contributions by Krzysztof Grabczewski	167
22.3	The finite cardinals	171
22.4	The first infinite cardinal: Omega, or nat	172
22.5	Towards Cardinal Arithmetic	172
22.6	Lemmas by Krzysztof Grabczewski	173
22.7	Finite and infinite sets	174
23	Univ: The Cumulative Hierarchy and a Small Universe for Recursive Types	177
23.1	Immediate Consequences of the Definition of $Vfrom(A, i)$. .	177
23.1.1	Monotonicity	178
23.1.2	A fundamental equality: $Vfrom$ does not require or- dinals!	178
23.2	Basic Closure Properties	178
23.2.1	Finite sets and ordered pairs	178
23.3	0, Successor and Limit Equations for $Vfrom$	179
23.4	$Vfrom$ applied to Limit Ordinals	179
23.4.1	Closure under Disjoint Union	180
23.5	Properties assuming $Transset(A)$	180
23.5.1	Products	181
23.5.2	Disjoint Sums, or Quine Ordered Pairs	181
23.5.3	Function Space!	181
23.6	The Set $Vset(i)$	182
23.6.1	Characterisation of the elements of $Vset(i)$	182
23.6.2	Reasoning about Sets in Terms of Their Elements' Ranks	182
23.6.3	Set Up an Environment for Simplification	183

23.6.4	Recursion over Vset Levels!	183
23.7	The Datatype Universe: $univ(A)$	183
23.7.1	The Set $univ(A)$ as a Limit	183
23.8	Closure Properties for $univ(A)$	184
23.8.1	Closure under Unordered and Ordered Pairs	184
23.8.2	The Natural Numbers	184
23.8.3	Instances for 1 and 2	185
23.8.4	Closure under Disjoint Union	185
23.9	Finite Branching Closure Properties	185
23.9.1	Closure under Finite Powerset	185
23.9.2	Closure under Finite Powers: Functions from a Natural Number	186
23.9.3	Closure under Finite Function Space	186
23.10*	For QUniv. Properties of Vfrom analogous to the "take-lemma" *	186
24	QUniv: A Small Universe for Lazy Recursive Types	187
24.1	Properties involving Transset and Sum	187
24.2	Introduction and Elimination Rules	188
24.3	Closure Properties	188
24.4	Quine Disjoint Sum	189
24.5	Closure for Quine-Inspired Products and Sums	189
24.6	Quine Disjoint Sum	189
24.7	The Natural Numbers	190
24.8	"Take-Lemma" Rules	190
25	Datatype-ZF: Datatype and CoDatatype Definitions	190
26	Arith: Arithmetic Operators and Their Definitions	191
26.1	$natify$, the Coercion to nat	192
26.2	Typing rules	194
26.3	Addition	194
26.4	Monotonicity of Addition	196
26.5	Multiplication	198
27	ArithSimp: Arithmetic with simplification	199
27.1	Difference	199
27.2	Remainder	200
27.3	Division	201
27.4	Further Facts about Remainder	202
27.5	Additional theorems about \leq	202
27.6	Cancellation Laws for Common Factors in Comparisons	203
27.7	More Lemmas about Remainder	204
27.7.1	More Lemmas About Difference	205

28 List-ZF: Lists in Zermelo-Fraenkel Set Theory	206
28.1 The function zip	219
29 EquivClass: Equivalence Relations	225
29.1 Suppes, Theorem 70: r is an equiv relation iff $\text{converse}(r) \circ$ $r = r$	225
29.2 Defining Unary Operations upon Equivalence Classes	227
29.3 Defining Binary Operations upon Equivalence Classes	227
30 Int-ZF: The Integers as Equivalence Classes Over Pairs of Natural Numbers	228
30.1 Proving that <i>intrel</i> is an equivalence relation	230
30.2 Collapsing rules: to remove <i>intify</i> from arithmetic expressions	232
30.3 <i>zminus</i> : unary negation on <i>int</i>	233
30.4 <i>znegative</i> : the test for negative integers	233
30.5 <i>nat-of</i> : Coercion of an Integer to a Natural Number	234
30.6 <i>zmagnitude</i> : magnitude of an integer, as a natural number . .	234
30.7 <i>op \$+</i> : addition on int	235
30.8 <i>op \$×</i> : Integer Multiplication	237
30.9 The "Less Than" Relation	239
30.10 Less Than or Equals	241
30.11 More subtraction laws (for <i>zcompare-rls</i>)	241
30.12 Monotonicity and Cancellation Results for Instantiation of the CancelNumerals Simprocs	242
30.13 Comparison laws	243
30.13.1 More inequality lemmas	243
30.13.2 The next several equations are permutative: watch out!	244
31 Bin: Arithmetic on Binary Integers	244
31.0.3 The Carry and Borrow Functions, <i>bin-succ</i> and <i>bin-pred</i>	247
31.0.4 <i>bin-minus</i> : Unary Negation of Binary Integers	247
31.0.5 <i>bin-add</i> : Binary Addition	247
31.0.6 <i>bin-mult</i> : Binary Multiplication	248
31.1 Computations	248
31.2 Simplification Rules for Comparison of Binary Numbers . . .	250
32 IntDiv-ZF: The Division Operators Div and Mod	255
32.1 Uniqueness and monotonicity of quotients and remainders . .	260
32.2 Correctness of posDivAlg, the Division Algorithm for $a \geq 0$ and $b > 0$	260
32.3 Some convenient biconditionals for products of signs	261
32.4 Correctness of negDivAlg, the division algorithm for $a \neq 0$ and $b \neq 0$	262
32.5 Existence shown by proving the division algorithm to be correct	263

32.6	division of a number by itself	267
32.7	Computation of division and remainder	267
32.8	Monotonicity in the first argument (divisor)	269
32.9	Monotonicity in the second argument (dividend)	270
32.10	More algebraic laws for zdiv and zmod	270
32.11	proving $a \text{ zdiv } (b * c) = (a \text{ zdiv } b) \text{ zdiv } c$	273
32.12	Cancellation of common factors in " zdiv "	273
32.13	Distribution of factors over " zmod "	274
33	CardinalArith: Cardinal Arithmetic Without the Axiom of Choice	275
33.1	Cardinal addition	276
33.1.1	Cardinal addition is commutative	276
33.1.2	Cardinal addition is associative	276
33.1.3	0 is the identity for addition	277
33.1.4	Addition by another cardinal	277
33.1.5	Monotonicity of addition	277
33.1.6	Addition of finite cardinals is "ordinary" addition	277
33.2	Cardinal multiplication	278
33.2.1	Cardinal multiplication is commutative	278
33.2.2	Cardinal multiplication is associative	278
33.2.3	Cardinal multiplication distributes over addition	278
33.2.4	Multiplication by 0 yields 0	278
33.2.5	1 is the identity for multiplication	278
33.3	Some inequalities for multiplication	278
33.3.1	Multiplication by a non-zero cardinal	279
33.3.2	Monotonicity of multiplication	279
33.4	Multiplication of finite cardinals is "ordinary" multiplication	279
33.5	Infinite Cardinals are Limit Ordinals	280
33.5.1	Establishing the well-ordering	280
33.5.2	Characterising initial segments of the well-ordering	280
33.5.3	The cardinality of initial segments	281
33.5.4	Toward's Kunen's Corollary 10.13 (1)	282
33.6	For Every Cardinal Number There Exists A Greater One	282
33.7	Basic Properties of Successor Cardinals	283
33.7.1	Removing elements from a finite set decreases its cardinality	283
33.7.2	Theorems by Krzysztof Grabczewski, proofs by lcp	284
34	Main-ZF: Theory Main: Everything Except AC	284
34.1	Iteration of the function F	284
34.2	Transfinite Recursion	285
35	AC: The Axiom of Choice	286

36 Zorn: Zorn's Lemma	287
36.1 Mathematical Preamble	288
36.2 The Transfinite Construction	288
36.3 Some Properties of the Transfinite Construction	288
36.4 Hausdorff's Theorem: Every Set Contains a Maximal Chain .	289
36.5 Zorn's Lemma: If All Chains in S Have Upper Bounds In S, then S contains a Maximal Element	290
36.6 Zermelo's Theorem: Every Set can be Well-Ordered	291
36.7 Zorn's Lemma for Partial Orders	291
37 Cardinal-AC: Cardinal Arithmetic Using AC	292
37.1 Strengthened Forms of Existing Theorems on Cardinals . . .	292
37.2 The relationship between cardinality and le-pollence	293
37.3 Other Applications of AC	293
38 InfDatatype: Infinite-Branching Datatype Definitions	294



1 ZF: Zermelo-Fraenkel Set Theory

theory *ZF* **imports** *FOL* **begin**

$\langle ML \rangle$

global

typedecl *i*

arities *i* :: *term*

consts

0 :: *i* (*0*) — the empty set
Pow :: *i* => *i* — power sets
Inf :: *i* — infinite set

Bounded Quantifiers

consts

Ball :: [*i*, *i* => *o*] => *o*
Bex :: [*i*, *i* => *o*] => *o*

General Union and Intersection

consts

Union :: *i* => *i*
Inter :: *i* => *i*

Variations on Replacement

consts

PrimReplace :: [*i*, [*i*, *i*] => *o*] => *i*
Replace :: [*i*, [*i*, *i*] => *o*] => *i*
RepFun :: [*i*, *i* => *i*] => *i*
Collect :: [*i*, *i* => *o*] => *i*

Definite descriptions – via Replace over the set "1"

consts

The :: (*i* => *o*) => *i* (**binder** *THE* 10)
If :: [*o*, *i*, *i*] => *i* ((*if* (-)/ *then* (-)/ *else* (-)) [10] 10)

abbreviation (*input*)

old-if :: [*o*, *i*, *i*] => *i* (*if* '(-,-,-')) **where**
if(*P*,*a*,*b*) == *If*(*P*,*a*,*b*)

Finite Sets

consts

Upair :: [*i*, *i*] => *i*
cons :: [*i*, *i*] => *i*
succ :: *i* => *i*

Ordered Pairing

consts

Pair :: $[i, i] \Rightarrow i$
fst :: $i \Rightarrow i$
snd :: $i \Rightarrow i$
split :: $[[i, i] \Rightarrow 'a, i] \Rightarrow 'a::\{\}$ — for pattern-matching

Sigma and Pi Operators

consts

Sigma :: $[i, i \Rightarrow i] \Rightarrow i$
Pi :: $[i, i \Rightarrow i] \Rightarrow i$

Relations and Functions

consts

domain :: $i \Rightarrow i$
range :: $i \Rightarrow i$
field :: $i \Rightarrow i$
converse :: $i \Rightarrow i$
relation :: $i \Rightarrow o$ — recognizes sets of pairs
function :: $i \Rightarrow o$ — recognizes functions; can have non-pairs
Lambda :: $[i, i \Rightarrow i] \Rightarrow i$
restrict :: $[i, i] \Rightarrow i$

Infixes in order of decreasing precedence

consts

Image :: $[i, i] \Rightarrow i$ (**infixl** “ 90) — image
vimage :: $[i, i] \Rightarrow i$ (**infixl** -“ 90) — inverse image
apply :: $[i, i] \Rightarrow i$ (**infixl** ‘ 90) — function application
Int :: $[i, i] \Rightarrow i$ (**infixl** *Int* 70) — binary intersection
Un :: $[i, i] \Rightarrow i$ (**infixl** *Un* 65) — binary union
Diff :: $[i, i] \Rightarrow i$ (**infixl** - 65) — set difference
Subset :: $[i, i] \Rightarrow o$ (**infixl** <= 50) — subset relation
mem :: $[i, i] \Rightarrow o$ (**infixl** : 50) — membership relation

abbreviation

not-mem :: $[i, i] \Rightarrow o$ (**infixl** ~: 50) — negated membership relation
where $x \sim: y == \sim (x : y)$

abbreviation

cart-prod :: $[i, i] \Rightarrow i$ (**infixr** * 80) — Cartesian product
where $A * B == \text{Sigma}(A, \%-. B)$

abbreviation

function-space :: $[i, i] \Rightarrow i$ (**infixr** -> 60) — function space
where $A -> B == \text{Pi}(A, \%-. B)$

nonterminals *is patterns*

syntax

```

:: i => is          (-)
-Enum    :: [i, is] => is      (-, / -)

-Finset   :: is => i          ({(-)})
-Tuple    :: [i, is] => i      (<(-, / -)>)
-Collect  :: [pttrn, i, o] => i ((1{- . / - . -}) )
-Replace  :: [pttrn, pttrn, i, o] => i ((1{- . / - . -}) )
-RepFun   :: [i, pttrn, i] => i ((1{- . / - . -}) [51,0,51])
-INTER    :: [pttrn, i, i] => i ((3INT -.- / -) 10)
-UNION    :: [pttrn, i, i] => i ((3UN -.- / -) 10)
-PROD     :: [pttrn, i, i] => i ((3PROD -.- / -) 10)
-SUM      :: [pttrn, i, i] => i ((3SUM -.- / -) 10)
-lam      :: [pttrn, i, i] => i ((3lam -.- / -) 10)
-Ball     :: [pttrn, i, o] => o ((3ALL -.- / -) 10)
-Bex      :: [pttrn, i, o] => o ((3EX -.- / -) 10)

```

```

-pattern  :: patterns => pttrn      (<->)
          :: pttrn => patterns      (-)
-patterns :: [pttrn, patterns] => patterns (-, / -)

```

translations

```

{x, xs}    == CONST cons(x, {xs})
{x}         == CONST cons(x, 0)
{x:A. P}    == CONST Collect(A, %x. P)
{y. x:A. Q} == CONST Replace(A, %x y. Q)
{b. x:A}    == CONST RepFun(A, %x. b)
INT x:A. B  == CONST Inter({B. x:A})
UN x:A. B   == CONST Union({B. x:A})
PROD x:A. B == CONST Pi(A, %x. B)
SUM x:A. B  == CONST Sigma(A, %x. B)
lam x:A. f  == CONST Lambda(A, %x. f)
ALL x:A. P  == CONST Ball(A, %x. P)
EX x:A. P   == CONST Bex(A, %x. P)

<x, y, z>   == <x, <y, z>>
<x, y>      == CONST Pair(x, y)
%<x,y,zs>.b == CONST split(%x <y,zs>.b)
%<x,y>.b    == CONST split(%x y. b)

```

notation (*xsymbols*)

```

cart-prod  (infixr × 80) and
Int        (infixl ∩ 70) and
Un         (infixl ∪ 65) and

```

function-space (**infixr** $\rightarrow 60$) **and**
Subset (**infixl** $\subseteq 50$) **and**
mem (**infixl** $\in 50$) **and**
not-mem (**infixl** $\notin 50$) **and**
Union (**l** \cup $[90]$ 90) **and**
Inter (**l** \cap $[90]$ 90)

syntax (*xsymbols*)

-Collect :: $[pttrn, i, o] \Rightarrow i$ $((1\{- \in - ./ -\}))$
-Replace :: $[pttrn, pttrn, i, o] \Rightarrow i$ $((1\{- ./ - \in -, -\}))$
-RepFun :: $[i, pttrn, i] \Rightarrow i$ $((1\{- ./ - \in -\}) [51, 0, 51])$
-UNION :: $[pttrn, i, i] \Rightarrow i$ $((3\cup - \in - ./ -) 10)$
-INTER :: $[pttrn, i, i] \Rightarrow i$ $((3\cap - \in - ./ -) 10)$
-PROD :: $[pttrn, i, i] \Rightarrow i$ $((3\Pi - \in - ./ -) 10)$
-SUM :: $[pttrn, i, i] \Rightarrow i$ $((3\Sigma - \in - ./ -) 10)$
-lam :: $[pttrn, i, i] \Rightarrow i$ $((3\lambda - \in - ./ -) 10)$
-Ball :: $[pttrn, i, o] \Rightarrow o$ $((3\forall - \in - ./ -) 10)$
-Bex :: $[pttrn, i, o] \Rightarrow o$ $((3\exists - \in - ./ -) 10)$
-Tuple :: $[i, is] \Rightarrow i$ $((\langle -, - \rangle))$
-pattern :: $patterns \Rightarrow pttrn$ $((\langle - \rangle))$

notation (*HTML output*)

cart-prod (**infixr** $\times 80$) **and**
Int (**infixl** $\cap 70$) **and**
Un (**infixl** $\cup 65$) **and**
Subset (**infixl** $\subseteq 50$) **and**
mem (**infixl** $\in 50$) **and**
not-mem (**infixl** $\notin 50$) **and**
Union (**l** \cup $[90]$ 90) **and**
Inter (**l** \cap $[90]$ 90)

syntax (*HTML output*)

-Collect :: $[pttrn, i, o] \Rightarrow i$ $((1\{- \in - ./ -\}))$
-Replace :: $[pttrn, pttrn, i, o] \Rightarrow i$ $((1\{- ./ - \in -, -\}))$
-RepFun :: $[i, pttrn, i] \Rightarrow i$ $((1\{- ./ - \in -\}) [51, 0, 51])$
-UNION :: $[pttrn, i, i] \Rightarrow i$ $((3\cup - \in - ./ -) 10)$
-INTER :: $[pttrn, i, i] \Rightarrow i$ $((3\cap - \in - ./ -) 10)$
-PROD :: $[pttrn, i, i] \Rightarrow i$ $((3\Pi - \in - ./ -) 10)$
-SUM :: $[pttrn, i, i] \Rightarrow i$ $((3\Sigma - \in - ./ -) 10)$
-lam :: $[pttrn, i, i] \Rightarrow i$ $((3\lambda - \in - ./ -) 10)$
-Ball :: $[pttrn, i, o] \Rightarrow o$ $((3\forall - \in - ./ -) 10)$
-Bex :: $[pttrn, i, o] \Rightarrow o$ $((3\exists - \in - ./ -) 10)$
-Tuple :: $[i, is] \Rightarrow i$ $((\langle -, - \rangle))$
-pattern :: $patterns \Rightarrow pttrn$ $((\langle - \rangle))$

finalconsts

0 Pow Inf Union PrimReplace mem

defs

Ball-def: $Ball(A, P) == \forall x. x \in A \rightarrow P(x)$

Bex-def: $Bex(A, P) == \exists x. x \in A \ \& \ P(x)$

subset-def: $A \leq B == \forall x \in A. x \in B$

local

axioms

extension: $A = B \leftrightarrow A \leq B \ \& \ B \leq A$

Union-iff: $A \in Union(C) \leftrightarrow (\exists B \in C. A \in B)$

Pow-iff: $A \in Pow(B) \leftrightarrow A \leq B$

infinity: $0 \in Inf \ \& \ (\forall y \in Inf. succ(y) \in Inf)$

foundation: $A = 0 \mid (\exists x \in A. \forall y \in x. y \sim A)$

replacement: $(\forall x \in A. \forall y \ z. P(x, y) \ \& \ P(x, z) \rightarrow y = z) \implies$
 $b \in PrimReplace(A, P) \leftrightarrow (\exists x \in A. P(x, b))$

defs

Replace-def: $Replace(A, P) == PrimReplace(A, \lambda x y. (EX!z. P(x, z)) \ \& \ P(x, y))$

RepFun-def: $RepFun(A, f) == \{y \mid x \in A, y = f(x)\}$

Collect-def: $Collect(A, P) == \{y \mid x \in A, x = y \ \& \ P(x)\}$

Upair-def: $Upair(a, b) == \{y. x \in Pow(Pow(0)), (x = 0 \ \& \ y = a) \mid (x = Pow(0) \ \& \ y = b)\}$

cons-def: $\text{cons}(a,A) == \text{Upair}(a,a) \text{ Un } A$
succ-def: $\text{succ}(i) == \text{cons}(i, i)$

Diff-def: $A - B == \{ x \in A . \sim(x \in B) \}$
Inter-def: $\text{Inter}(A) == \{ x \in \text{Union}(A) . \forall y \in A. x \in y \}$
Un-def: $A \text{ Un } B == \text{Union}(\text{Upair}(A,B))$
Int-def: $A \text{ Int } B == \text{Inter}(\text{Upair}(A,B))$

the-def: $\text{The}(P) == \text{Union}(\{y . x \in \{0\}, P(y)\})$
if-def: $\text{if}(P,a,b) == \text{THE } z. P \ \& \ z=a \mid \sim P \ \& \ z=b$

Pair-def: $\langle a,b \rangle == \{\{a,a\}, \{a,b\}\}$
fst-def: $\text{fst}(p) == \text{THE } a. \exists b. p = \langle a,b \rangle$
snd-def: $\text{snd}(p) == \text{THE } b. \exists a. p = \langle a,b \rangle$
split-def: $\text{split}(c) == \%p. c(\text{fst}(p), \text{snd}(p))$
Sigma-def: $\text{Sigma}(A,B) == \bigcup x \in A. \bigcup y \in B(x). \{ \langle x,y \rangle \}$

converse-def: $\text{converse}(r) == \{ z. w \in r, \exists x y. w = \langle x,y \rangle \ \& \ z = \langle y,x \rangle \}$

domain-def: $\text{domain}(r) == \{ x. w \in r, \exists y. w = \langle x,y \rangle \}$
range-def: $\text{range}(r) == \text{domain}(\text{converse}(r))$
field-def: $\text{field}(r) == \text{domain}(r) \text{ Un } \text{range}(r)$
relation-def: $\text{relation}(r) == \forall z \in r. \exists x y. z = \langle x,y \rangle$
function-def: $\text{function}(r) ==$
 $\quad \forall x y. \langle x,y \rangle : r \dashrightarrow (\forall y'. \langle x,y' \rangle : r \dashrightarrow y=y')$
image-def: $r \text{ `` } A == \{ y : \text{range}(r) . \exists x \in A. \langle x,y \rangle : r \}$
vimage-def: $r \text{ -`` } A == \text{converse}(r) \text{ `` } A$

lam-def: $\text{Lambda}(A,b) == \{ \langle x,b(x) \rangle . x \in A \}$
apply-def: $f'a == \text{Union}(f'\{a\})$
Pi-def: $\text{Pi}(A,B) == \{ f \in \text{Pow}(\text{Sigma}(A,B)). A \leq \text{domain}(f) \ \& \ \text{function}(f) \}$

restrict-def: $\text{restrict}(r,A) == \{ z : r. \exists x \in A. \exists y. z = \langle x,y \rangle \}$

1.1 Substitution

lemma *subst-elem:* $[[b \in A; \ a=b]] ==> a \in A$
<proof>

1.2 Bounded universal quantifier

lemma *ballI* [*intro!*]: $[\![\! \! x. x \in A \implies P(x) \! \!]\!] \implies \forall x \in A. P(x)$
 $\langle proof \rangle$

lemmas *strip* = *impI allI ballI*

lemma *bspec* [*dest?*]: $[\![\forall x \in A. P(x); x: A \!]\!] \implies P(x)$
 $\langle proof \rangle$

lemma *rev-ballE* [*elim*]:
 $[\![\forall x \in A. P(x); x \sim : A \implies Q; P(x) \implies Q \!]\!] \implies Q$
 $\langle proof \rangle$

lemma *ballE*: $[\![\forall x \in A. P(x); P(x) \implies Q; x \sim : A \implies Q \!]\!] \implies Q$
 $\langle proof \rangle$

lemma *rev-bspec*: $[\![x: A; \forall x \in A. P(x) \!]\!] \implies P(x)$
 $\langle proof \rangle$

lemma *ball-triv* [*simp*]: $(\forall x \in A. P) <-> ((\exists x. x \in A) \dashv\vdash P)$
 $\langle proof \rangle$

lemma *ball-cong* [*cong*]:
 $[\![A=A'; \! \! x. x \in A' \implies P(x) <-> P'(x) \!]\!] \implies (\forall x \in A. P(x)) <-> (\forall x \in A'. P'(x))$
 $\langle proof \rangle$

lemma *atomize-ball*:
 $(\! \! x. x \in A \implies P(x)) == Trueprop (\forall x \in A. P(x))$
 $\langle proof \rangle$

lemmas [*symmetric, rulify*] = *atomize-ball*
and [*symmetric, defn*] = *atomize-ball*

1.3 Bounded existential quantifier

lemma *bexI* [*intro*]: $[\![P(x); x: A \!]\!] \implies \exists x \in A. P(x)$
 $\langle proof \rangle$

lemma *rev-bexI*: $[\![x \in A; P(x) \!]\!] \implies \exists x \in A. P(x)$
 $\langle proof \rangle$

lemma *bexCI*: $[\![\forall x \in A. \sim P(x) \implies P(a); a: A \!]\!] \implies \exists x \in A. P(x)$

$\langle proof \rangle$

lemma *bexE* [*elim!*]: $[\exists x \in A. P(x); !!x. [x \in A; P(x)] ==> Q] ==> Q$
 $\langle proof \rangle$

lemma *bex-triv* [*simp*]: $(\exists x \in A. P) <-> ((\exists x. x \in A) \& P)$
 $\langle proof \rangle$

lemma *bex-cong* [*cong*]:
 $[A=A'; !!x. x \in A' ==> P(x) <-> P'(x)]$
 $==> (\exists x \in A. P(x)) <-> (\exists x \in A'. P'(x))$
 $\langle proof \rangle$

1.4 Rules for subsets

lemma *subsetI* [*intro!*]:
 $(!!x. x \in A ==> x \in B) ==> A <= B$
 $\langle proof \rangle$

lemma *subsetD* [*elim*]: $[A <= B; c \in A] ==> c \in B$
 $\langle proof \rangle$

lemma *subsetCE* [*elim*]:
 $[A <= B; c \sim A ==> P; c \in B ==> P] ==> P$
 $\langle proof \rangle$

lemma *rev-subsetD*: $[c \in A; A <= B] ==> c \in B$
 $\langle proof \rangle$

lemma *contra-subsetD*: $[A <= B; c \sim B] ==> c \sim A$
 $\langle proof \rangle$

lemma *rev-contra-subsetD*: $[c \sim B; A <= B] ==> c \sim A$
 $\langle proof \rangle$

lemma *subset-refl* [*simp*]: $A <= A$
 $\langle proof \rangle$

lemma *subset-trans*: $[A <= B; B <= C] ==> A <= C$
 $\langle proof \rangle$

lemma *subset-iff*:
 $A <= B <-> (\forall x. x \in A \longrightarrow x \in B)$
 $\langle proof \rangle$

1.5 Rules for equality

lemma *equalityI* [*intro*]: $\llbracket A \leq B; B \leq A \rrbracket \implies A = B$
 $\langle \text{proof} \rangle$

lemma *equality-iffI*: $(!!x. x \in A \leftrightarrow x \in B) \implies A = B$
 $\langle \text{proof} \rangle$

lemmas *equalityD1* = *extension* [*THEN iffD1*, *THEN conjunct1*, *standard*]

lemmas *equalityD2* = *extension* [*THEN iffD1*, *THEN conjunct2*, *standard*]

lemma *equalityE*: $\llbracket A = B; \llbracket A \leq B; B \leq A \rrbracket \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *equalityCE*:
 $\llbracket A = B; \llbracket c \in A; c \in B \rrbracket \implies P; \llbracket c \sim A; c \sim B \rrbracket \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *equality-iffD*:
 $A = B \implies (!!x. x : A \leftrightarrow x : B)$
 $\langle \text{proof} \rangle$

1.6 Rules for Replace – the derived form of replacement

lemma *Replace-iff*:
 $b : \{y. x \in A, P(x, y)\} \leftrightarrow (\exists x \in A. P(x, b) \ \& \ (\forall y. P(x, y) \longrightarrow y = b))$
 $\langle \text{proof} \rangle$

lemma *ReplaceI* [*intro*]:
 $\llbracket P(x, b); x : A; !!y. P(x, y) \implies y = b \rrbracket \implies$
 $b : \{y. x \in A, P(x, y)\}$
 $\langle \text{proof} \rangle$

lemma *ReplaceE*:
 $\llbracket b : \{y. x \in A, P(x, y)\};$
 $!!x. \llbracket x : A; P(x, b); \forall y. P(x, y) \longrightarrow y = b \rrbracket \implies R$
 $\rrbracket \implies R$
 $\langle \text{proof} \rangle$

lemma *ReplaceE2* [*elim!*]:
 $\llbracket b : \{y. x \in A, P(x, y)\};$
 $!!x. \llbracket x : A; P(x, b) \rrbracket \implies R$
 $\rrbracket \implies R$
 $\langle \text{proof} \rangle$

lemma *Replace-cong* [*cong*]:

$$[| A=B; \text{!!}x\ y.\ x \in B \implies P(x,y) \iff Q(x,y) |] \implies$$

$$\text{Replace}(A,P) = \text{Replace}(B,Q)$$

$$\langle \text{proof} \rangle$$

1.7 Rules for RepFun

lemma *RepFunI*: $a \in A \implies f(a) : \{f(x). x \in A\}$
 $\langle \text{proof} \rangle$

lemma *RepFun-eqI* [*intro*]: $[| b=f(a); a \in A |] \implies b : \{f(x). x \in A\}$
 $\langle \text{proof} \rangle$

lemma *RepFunE* [*elim!*]:

$$[| b : \{f(x). x \in A\};$$

$$\text{!!}x. [| x \in A; b=f(x) |] \implies P |] \implies$$

$$P$$

$$\langle \text{proof} \rangle$$

lemma *RepFun-cong* [*cong*]:

$$[| A=B; \text{!!}x. x \in B \implies f(x)=g(x) |] \implies \text{RepFun}(A,f) = \text{RepFun}(B,g)$$

$$\langle \text{proof} \rangle$$

lemma *RepFun-iff* [*simp*]: $b : \{f(x). x \in A\} \iff (\exists x \in A. b=f(x))$
 $\langle \text{proof} \rangle$

lemma *triv-RepFun* [*simp*]: $\{x. x \in A\} = A$
 $\langle \text{proof} \rangle$

1.8 Rules for Collect – forming a subset by separation

lemma *separation* [*simp*]: $a : \{x \in A. P(x)\} \iff a \in A \ \& \ P(a)$
 $\langle \text{proof} \rangle$

lemma *CollectI* [*intro!*]: $[| a \in A; P(a) |] \implies a : \{x \in A. P(x)\}$
 $\langle \text{proof} \rangle$

lemma *CollectE* [*elim!*]: $[| a : \{x \in A. P(x)\}; [| a \in A; P(a) |] \implies R |] \implies R$
 $\langle \text{proof} \rangle$

lemma *CollectD1*: $a : \{x \in A. P(x)\} \implies a \in A$
 $\langle \text{proof} \rangle$

lemma *CollectD2*: $a : \{x \in A. P(x)\} \implies P(a)$
 $\langle \text{proof} \rangle$

lemma *Collect-cong* [*cong*]:

$$[| A=B; \text{!!}x. x \in B \implies P(x) \iff Q(x) |]$$

$$\implies \text{Collect}(A, \%x. P(x)) = \text{Collect}(B, \%x. Q(x))$$

$$\langle \text{proof} \rangle$$

1.9 Rules for Unions

declare *Union-iff* [*simp*]

lemma *UnionI* [*intro*]: $[| B: C; A: B |] ==> A: \text{Union}(C)$
 $\langle \text{proof} \rangle$

lemma *UnionE* [*elim!*]: $[| A \in \text{Union}(C); !!B. [| A: B; B: C |] ==> R |] ==> R$
 $\langle \text{proof} \rangle$

1.10 Rules for Unions of families

lemma *UN-iff* [*simp*]: $b : (\bigcup_{x \in A}. B(x)) <-> (\exists x \in A. b \in B(x))$
 $\langle \text{proof} \rangle$

lemma *UN-I*: $[| a: A; b: B(a) |] ==> b: (\bigcup_{x \in A}. B(x))$
 $\langle \text{proof} \rangle$

lemma *UN-E* [*elim!*]:
 $[| b : (\bigcup_{x \in A}. B(x)); !!x. [| x: A; b: B(x) |] ==> R |] ==> R$
 $\langle \text{proof} \rangle$

lemma *UN-cong*:
 $[| A=B; !!x. x \in B ==> C(x)=D(x) |] ==> (\bigcup_{x \in A}. C(x)) = (\bigcup_{x \in B}. D(x))$
 $\langle \text{proof} \rangle$

1.11 Rules for the empty set

lemma *not-mem-empty* [*simp*]: $a \sim: 0$
 $\langle \text{proof} \rangle$

lemmas *emptyE* [*elim!*] = *not-mem-empty* [*THEN notE, standard*]

lemma *empty-subsetI* [*simp*]: $0 \leq A$
 $\langle \text{proof} \rangle$

lemma *equals0I*: $[| !!y. y \in A ==> \text{False} |] ==> A=0$
 $\langle \text{proof} \rangle$

lemma *equals0D* [*dest*]: $A=0 ==> a \sim: A$
 $\langle \text{proof} \rangle$

declare *sym* [*THEN equals0D, dest*]

lemma *not-emptyI*: $a \in A ==> A \sim= 0$

$\langle proof \rangle$

lemma *not-emptyE*: $[| A \sim 0; !!x. x \in A ==> R |] ==> R$
 $\langle proof \rangle$

1.12 Rules for Inter

lemma *Inter-iff*: $A \in \text{Inter}(C) <-> (\forall x \in C. A: x) \ \& \ C \neq 0$
 $\langle proof \rangle$

lemma *InterI* [*intro!*]:
 $[| !!x. x: C ==> A: x; C \neq 0 |] ==> A \in \text{Inter}(C)$
 $\langle proof \rangle$

lemma *InterD* [*elim*]: $[| A \in \text{Inter}(C); B \in C |] ==> A \in B$
 $\langle proof \rangle$

lemma *InterE* [*elim*]:
 $[| A \in \text{Inter}(C); B \sim C ==> R; A \in B ==> R |] ==> R$
 $\langle proof \rangle$

1.13 Rules for Intersections of families

lemma *INT-iff*: $b : (\bigcap x \in A. B(x)) <-> (\forall x \in A. b \in B(x)) \ \& \ A \neq 0$
 $\langle proof \rangle$

lemma *INT-I*: $[| !!x. x: A ==> b: B(x); A \neq 0 |] ==> b: (\bigcap x \in A. B(x))$
 $\langle proof \rangle$

lemma *INT-E*: $[| b : (\bigcap x \in A. B(x)); a: A |] ==> b \in B(a)$
 $\langle proof \rangle$

lemma *INT-cong*:
 $[| A=B; !!x. x \in B ==> C(x)=D(x) |] ==> (\bigcap x \in A. C(x)) = (\bigcap x \in B. D(x))$
 $\langle proof \rangle$

1.14 Rules for Powersets

lemma *PowI*: $A \leq B ==> A \in \text{Pow}(B)$
 $\langle proof \rangle$

lemma *PowD*: $A \in \text{Pow}(B) ==> A \leq B$
 $\langle proof \rangle$

declare *Pow-iff* [*iff*]

lemmas *Pow-bottom* = *empty-subsetI* [*THEN PowI*]

lemmas *Pow-top* = *subset-refl* [*THEN PowI*]

1.15 Cantor's Theorem: There is no surjection from a set to its powerset.

lemma *cantor*: $\exists S \in \text{Pow}(A). \forall x \in A. b(x) \sim S$
 $\langle \text{proof} \rangle$

$\langle \text{ML} \rangle$

end

2 upair: Unordered Pairs

theory *upair* **imports** *ZF*
uses *Tools/typechk.ML* **begin**

$\langle \text{ML} \rangle$

lemma *atomize-ball* [*symmetric, rulify*]:
 $(!!x. x:A ==> P(x)) == \text{Trueprop } (\text{ALL } x:A. P(x))$
 $\langle \text{proof} \rangle$

2.1 Unordered Pairs: constant *Upair*

lemma *Upair-iff* [*simp*]: $c : \text{Upair}(a,b) <-> (c=a \mid c=b)$
 $\langle \text{proof} \rangle$

lemma *UpairI1*: $a : \text{Upair}(a,b)$
 $\langle \text{proof} \rangle$

lemma *UpairI2*: $b : \text{Upair}(a,b)$
 $\langle \text{proof} \rangle$

lemma *UpairE*: $[| a : \text{Upair}(b,c); a=b ==> P; a=c ==> P |] ==> P$
 $\langle \text{proof} \rangle$

2.2 Rules for Binary Union, Defined via *Upair*

lemma *Un-iff* [*simp*]: $c : A \text{ Un } B <-> (c:A \mid c:B)$
 $\langle \text{proof} \rangle$

lemma *UnI1*: $c : A ==> c : A \text{ Un } B$
 $\langle \text{proof} \rangle$

lemma *UnI2*: $c : B ==> c : A \text{ Un } B$

$\langle proof \rangle$

declare $UnI1$ [elim?] $UnI2$ [elim?]

lemma UnE [elim!]: $[[c : A \text{ Un } B; c:A ==> P; c:B ==> P]] ==> P$
 $\langle proof \rangle$

lemma UnE' : $[[c : A \text{ Un } B; c:A ==> P; [c:B; c\sim:A]] ==> P]] ==> P$
 $\langle proof \rangle$

lemma $UnCI$ [intro!]: $(c \sim: B ==> c : A) ==> c : A \text{ Un } B$
 $\langle proof \rangle$

2.3 Rules for Binary Intersection, Defined via $Upair$

lemma $Int\text{-}iff$ [simp]: $c : A \text{ Int } B <-> (c:A \ \& \ c:B)$
 $\langle proof \rangle$

lemma $IntI$ [intro!]: $[[c : A; c : B]] ==> c : A \text{ Int } B$
 $\langle proof \rangle$

lemma $IntD1$: $c : A \text{ Int } B ==> c : A$
 $\langle proof \rangle$

lemma $IntD2$: $c : A \text{ Int } B ==> c : B$
 $\langle proof \rangle$

lemma $IntE$ [elim!]: $[[c : A \text{ Int } B; [c:A; c:B]] ==> P]] ==> P$
 $\langle proof \rangle$

2.4 Rules for Set Difference, Defined via $Upair$

lemma $Diff\text{-}iff$ [simp]: $c : A - B <-> (c:A \ \& \ c\sim:B)$
 $\langle proof \rangle$

lemma $DiffI$ [intro!]: $[[c : A; c \sim: B]] ==> c : A - B$
 $\langle proof \rangle$

lemma $DiffD1$: $c : A - B ==> c : A$
 $\langle proof \rangle$

lemma $DiffD2$: $c : A - B ==> c \sim: B$
 $\langle proof \rangle$

lemma $DiffE$ [elim!]: $[[c : A - B; [c:A; c\sim:B]] ==> P]] ==> P$
 $\langle proof \rangle$

2.5 Rules for *cons*

lemma *cons-iff* [*simp*]: $a : \text{cons}(b, A) <-> (a=b \mid a:A)$
 $\langle \text{proof} \rangle$

lemma *consI1* [*simp, TC*]: $a : \text{cons}(a, B)$
 $\langle \text{proof} \rangle$

lemma *consI2*: $a : B ==> a : \text{cons}(b, B)$
 $\langle \text{proof} \rangle$

lemma *consE* [*elim!*]: $[| a : \text{cons}(b, A); a=b ==> P; a:A ==> P |] ==> P$
 $\langle \text{proof} \rangle$

lemma *consE'*:
 $[| a : \text{cons}(b, A); a=b ==> P; [| a:A; a \sim b |] ==> P |] ==> P$
 $\langle \text{proof} \rangle$

lemma *consCI* [*intro!*]: $(a \sim b ==> a=b) ==> a : \text{cons}(b, B)$
 $\langle \text{proof} \rangle$

lemma *cons-not-0* [*simp*]: $\text{cons}(a, B) \sim = 0$
 $\langle \text{proof} \rangle$

lemmas *cons-neq-0* = *cons-not-0* [*THEN notE, standard*]

declare *cons-not-0* [*THEN not-sym, simp*]

2.6 Singletons

lemma *singleton-iff*: $a : \{b\} <-> a=b$
 $\langle \text{proof} \rangle$

lemma *singletonI* [*intro!*]: $a : \{a\}$
 $\langle \text{proof} \rangle$

lemmas *singletonE* = *singleton-iff* [*THEN iffD1, elim-format, standard, elim!*]

2.7 Descriptions

lemma *the-equality* [*intro*]:
 $[| P(a); !x. P(x) ==> x=a |] ==> (\text{THE } x. P(x)) = a$
 $\langle \text{proof} \rangle$

lemma *the-equality2*: $[| \text{EX! } x. P(x); P(a) |] ==> (\text{THE } x. P(x)) = a$

$\langle proof \rangle$

lemma *theI*: $EX! x. P(x) ==> P(THEx. P(x))$
 $\langle proof \rangle$

lemma *the-0*: $\sim (EX! x. P(x)) ==> (THEx. P(x))=0$
 $\langle proof \rangle$

lemma *theI2*:
 assumes *p1*: $\sim Q(0) ==> EX! x. P(x)$
 and *p2*: $!!x. P(x) ==> Q(x)$
 shows $Q(THEx. P(x))$
 $\langle proof \rangle$

lemma *the-eq-trivial* [*simp*]: $(THEx. x = a) = a$
 $\langle proof \rangle$

lemma *the-eq-trivial2* [*simp*]: $(THEx. a = x) = a$
 $\langle proof \rangle$

2.8 Conditional Terms: *if-then-else*

lemma *if-true* [*simp*]: $(if\ True\ then\ a\ else\ b) = a$
 $\langle proof \rangle$

lemma *if-false* [*simp*]: $(if\ False\ then\ a\ else\ b) = b$
 $\langle proof \rangle$

lemma *if-cong*:
 $[[P <-> Q; Q ==> a=c; \sim Q ==> b=d]]$
 $==> (if\ P\ then\ a\ else\ b) = (if\ Q\ then\ c\ else\ d)$
 $\langle proof \rangle$

lemma *if-weak-cong*: $P <-> Q ==> (if\ P\ then\ x\ else\ y) = (if\ Q\ then\ x\ else\ y)$
 $\langle proof \rangle$

lemma *if-P*: $P ==> (if\ P\ then\ a\ else\ b) = a$
 $\langle proof \rangle$

lemma *if-not-P*: $\sim P ==> (if\ P\ then\ a\ else\ b) = b$
 $\langle proof \rangle$

lemma *split-if* [*split*]:

$P(\text{if } Q \text{ then } x \text{ else } y) <-> ((Q \dashrightarrow P(x)) \ \& \ (\sim Q \dashrightarrow P(y)))$
 $\langle \text{proof} \rangle$

lemmas *split-if-eq1* = *split-if* [*of* %*x*. *x* = *b*, *standard*]

lemmas *split-if-eq2* = *split-if* [*of* %*x*. *a* = *x*, *standard*]

lemmas *split-if-mem1* = *split-if* [*of* %*x*. *x* : *b*, *standard*]

lemmas *split-if-mem2* = *split-if* [*of* %*x*. *a* : *x*, *standard*]

lemmas *split-ifs* = *split-if-eq1* *split-if-eq2* *split-if-mem1* *split-if-mem2*

lemma *if-iff*: $a: (\text{if } P \text{ then } x \text{ else } y) <-> P \ \& \ a:x \mid \sim P \ \& \ a:y$

$\langle \text{proof} \rangle$

lemma *if-type* [*TC*]:

$[\mid P \implies a: A; \ \sim P \implies b: A \mid] \implies (\text{if } P \text{ then } a \text{ else } b): A$
 $\langle \text{proof} \rangle$

lemma *split-if-asm*: $P(\text{if } Q \text{ then } x \text{ else } y) <-> (\sim((Q \ \& \ \sim P(x)) \mid (\sim Q \ \& \ \sim P(y))))$

$\langle \text{proof} \rangle$

lemmas *if-splits* = *split-if* *split-if-asm*

2.9 Consequences of Foundation

lemma *mem-asym*: $[\mid a:b; \ \sim P \implies b:a \mid] \implies P$

$\langle \text{proof} \rangle$

lemma *mem-irrefl*: $a:a \implies P$

$\langle \text{proof} \rangle$

lemma *mem-not-refl*: $a \sim: a$

$\langle \text{proof} \rangle$

lemma *mem-imp-not-eq*: $a:A \implies a \sim = A$

$\langle \text{proof} \rangle$

lemma *eq-imp-not-mem*: $a=A \implies a \sim: A$

$\langle proof \rangle$

2.10 Rules for Successor

lemma *succ-iff*: $i : succ(j) \leftrightarrow i=j \mid i:j$
 $\langle proof \rangle$

lemma *succI1* [*simp*]: $i : succ(i)$
 $\langle proof \rangle$

lemma *succI2*: $i : j \implies i : succ(j)$
 $\langle proof \rangle$

lemma *succE* [*elim!*]:
 $[[i : succ(j); i=j \implies P; i:j \implies P]] \implies P$
 $\langle proof \rangle$

lemma *succCI* [*intro!*]: $(i \sim j \implies i=j) \implies i : succ(j)$
 $\langle proof \rangle$

lemma *succ-not-0* [*simp*]: $succ(n) \sim 0$
 $\langle proof \rangle$

lemmas *succ-neq-0* = *succ-not-0* [*THEN notE, standard, elim!*]

declare *succ-not-0* [*THEN not-sym, simp*]
declare *sym* [*THEN succ-neq-0, elim!*]

lemmas *succ-subsetD* = *succI1* [*THEN* [2] *subsetD*]

lemmas *succ-neq-self* = *succI1* [*THEN mem-imp-not-eq, THEN not-sym, standard*]

lemma *succ-inject-iff* [*simp*]: $succ(m) = succ(n) \leftrightarrow m=n$
 $\langle proof \rangle$

lemmas *succ-inject* = *succ-inject-iff* [*THEN iffD1, standard, dest!*]

2.11 Miniscoping of the Bounded Universal Quantifier

lemma *ball-simps1*:
 $(ALL x:A. P(x) \ \& \ Q) \leftrightarrow (ALL x:A. P(x)) \ \& \ (A=0 \mid Q)$
 $(ALL x:A. P(x) \mid Q) \leftrightarrow ((ALL x:A. P(x)) \mid Q)$
 $(ALL x:A. P(x) \dashv\vdash Q) \leftrightarrow ((EX x:A. P(x)) \dashv\vdash Q)$
 $(\sim(ALL x:A. P(x))) \leftrightarrow (EX x:A. \sim P(x))$
 $(ALL x:0.P(x)) \leftrightarrow True$
 $(ALL x:succ(i).P(x)) \leftrightarrow P(i) \ \& \ (ALL x:i. P(x))$

$(ALL\ x:cons(a,B).P(x)) <-> P(a) \ \& \ (ALL\ x:B. P(x))$
 $(ALL\ x:RepFun(A,f). P(x)) <-> (ALL\ y:A. P(f(y)))$
 $(ALL\ x:Union(A).P(x)) <-> (ALL\ y:A. ALL\ x:y. P(x))$
 $\langle proof \rangle$

lemma *ball-simps2*:

$(ALL\ x:A. P \ \& \ Q(x)) <-> (A=0 \mid P) \ \& \ (ALL\ x:A. Q(x))$
 $(ALL\ x:A. P \mid Q(x)) <-> (P \mid (ALL\ x:A. Q(x)))$
 $(ALL\ x:A. P \dashv\> Q(x)) <-> (P \dashv\> (ALL\ x:A. Q(x)))$
 $\langle proof \rangle$

lemma *ball-simps3*:

$(ALL\ x:Collect(A,Q).P(x)) <-> (ALL\ x:A. Q(x) \dashv\> P(x))$
 $\langle proof \rangle$

lemmas *ball-simps* [simp] = *ball-simps1 ball-simps2 ball-simps3*

lemma *ball-conj-distrib*:

$(ALL\ x:A. P(x) \ \& \ Q(x)) <-> ((ALL\ x:A. P(x)) \ \& \ (ALL\ x:A. Q(x)))$
 $\langle proof \rangle$

2.12 Miniscoping of the Bounded Existential Quantifier

lemma *bex-simps1*:

$(EX\ x:A. P(x) \ \& \ Q) <-> ((EX\ x:A. P(x)) \ \& \ Q)$
 $(EX\ x:A. P(x) \mid Q) <-> (EX\ x:A. P(x)) \mid (A^{\sim}=0 \ \& \ Q)$
 $(EX\ x:A. P(x) \dashv\> Q) <-> ((ALL\ x:A. P(x)) \dashv\> (A^{\sim}=0 \ \& \ Q))$
 $(EX\ x:0.P(x)) <-> False$
 $(EX\ x:succ(i).P(x)) <-> P(i) \mid (EX\ x:i. P(x))$
 $(EX\ x:cons(a,B).P(x)) <-> P(a) \mid (EX\ x:B. P(x))$
 $(EX\ x:RepFun(A,f). P(x)) <-> (EX\ y:A. P(f(y)))$
 $(EX\ x:Union(A).P(x)) <-> (EX\ y:A. EX\ x:y. P(x))$
 $(\sim(EX\ x:A. P(x))) <-> (ALL\ x:A. \sim P(x))$
 $\langle proof \rangle$

lemma *bex-simps2*:

$(EX\ x:A. P \ \& \ Q(x)) <-> (P \ \& \ (EX\ x:A. Q(x)))$
 $(EX\ x:A. P \mid Q(x)) <-> (A^{\sim}=0 \ \& \ P) \mid (EX\ x:A. Q(x))$
 $(EX\ x:A. P \dashv\> Q(x)) <-> ((A=0 \mid P) \dashv\> (EX\ x:A. Q(x)))$
 $\langle proof \rangle$

lemma *bex-simps3*:

$(EX\ x:Collect(A,Q).P(x)) <-> (EX\ x:A. Q(x) \ \& \ P(x))$
 $\langle proof \rangle$

lemmas *bex-simps* [simp] = *bex-simps1 bex-simps2 bex-simps3*

lemma *bex-disj-distrib*:

$(EX\ x:A. P(x) \mid Q(x)) <-> ((EX\ x:A. P(x)) \mid (EX\ x:A. Q(x)))$

$\langle proof \rangle$

lemma *bex-triv-one-point1* [simp]: $(EX\ x:A. x=a) <-> (a:A)$
 $\langle proof \rangle$

lemma *bex-triv-one-point2* [simp]: $(EX\ x:A. a=x) <-> (a:A)$
 $\langle proof \rangle$

lemma *bex-one-point1* [simp]: $(EX\ x:A. x=a \ \&\ P(x)) <-> (a:A \ \&\ P(a))$
 $\langle proof \rangle$

lemma *bex-one-point2* [simp]: $(EX\ x:A. a=x \ \&\ P(x)) <-> (a:A \ \&\ P(a))$
 $\langle proof \rangle$

lemma *ball-one-point1* [simp]: $(ALL\ x:A. x=a \ \longrightarrow P(x)) <-> (a:A \ \longrightarrow P(a))$
 $\langle proof \rangle$

lemma *ball-one-point2* [simp]: $(ALL\ x:A. a=x \ \longrightarrow P(x)) <-> (a:A \ \longrightarrow P(a))$
 $\langle proof \rangle$

2.13 Miniscoping of the Replacement Operator

These cover both *Replace* and *Collect*

lemma *Rep-simps* [simp]:
 $\{x. y:0, R(x,y)\} = 0$
 $\{x:0. P(x)\} = 0$
 $\{x:A. Q\} = (if\ Q\ then\ A\ else\ 0)$
 $RepFun(0,f) = 0$
 $RepFun(succ(i),f) = cons(f(i), RepFun(i,f))$
 $RepFun(cons(a,B),f) = cons(f(a), RepFun(B,f))$
 $\langle proof \rangle$

2.14 Miniscoping of Unions

lemma *UN-simps1*:
 $(UN\ x:C. cons(a, B(x))) = (if\ C=0\ then\ 0\ else\ cons(a, UN\ x:C. B(x)))$
 $(UN\ x:C. A(x) \ Un\ B') = (if\ C=0\ then\ 0\ else\ (UN\ x:C. A(x)) \ Un\ B')$
 $(UN\ x:C. A' \ Un\ B(x)) = (if\ C=0\ then\ 0\ else\ A' \ Un\ (UN\ x:C. B(x)))$
 $(UN\ x:C. A(x) \ Int\ B') = ((UN\ x:C. A(x)) \ Int\ B')$
 $(UN\ x:C. A' \ Int\ B(x)) = (A' \ Int\ (UN\ x:C. B(x)))$
 $(UN\ x:C. A(x) - B') = ((UN\ x:C. A(x)) - B')$
 $(UN\ x:C. A' - B(x)) = (if\ C=0\ then\ 0\ else\ A' - (INT\ x:C. B(x)))$
 $\langle proof \rangle$

lemma *UN-simps2*:
 $(UN\ x: Union(A). B(x)) = (UN\ y:A. UN\ x:y. B(x))$

$$\begin{aligned} (UN\ z: (UN\ x:A. B(x)). C(z)) &= (UN\ x:A. UN\ z: B(x). C(z)) \\ (UN\ x: RepFun(A,f). B(x)) &= (UN\ a:A. B(f(a))) \end{aligned}$$
 $\langle proof \rangle$

lemmas $UN-simps\ [simp] = UN-simps1\ UN-simps2$

Opposite of miniscoping: pull the operator out

lemma $UN-extend-simps1$:

$$\begin{aligned} (UN\ x:C. A(x))\ Un\ B &= (if\ C=0\ then\ B\ else\ (UN\ x:C. A(x)\ Un\ B)) \\ ((UN\ x:C. A(x))\ Int\ B) &= (UN\ x:C. A(x)\ Int\ B) \\ ((UN\ x:C. A(x)) - B) &= (UN\ x:C. A(x) - B) \end{aligned}$$

$\langle proof \rangle$

lemma $UN-extend-simps2$:

$$\begin{aligned} cons(a, UN\ x:C. B(x)) &= (if\ C=0\ then\ \{a\}\ else\ (UN\ x:C. cons(a, B(x)))) \\ A\ Un\ (UN\ x:C. B(x)) &= (if\ C=0\ then\ A\ else\ (UN\ x:C. A\ Un\ B(x))) \\ (A\ Int\ (UN\ x:C. B(x))) &= (UN\ x:C. A\ Int\ B(x)) \\ A - (INT\ x:C. B(x)) &= (if\ C=0\ then\ A\ else\ (UN\ x:C. A - B(x))) \\ (UN\ y:A. UN\ x:y. B(x)) &= (UN\ x: Union(A). B(x)) \\ (UN\ a:A. B(f(a))) &= (UN\ x: RepFun(A,f). B(x)) \end{aligned}$$

$\langle proof \rangle$

lemma $UN-UN-extend$:

$$(UN\ x:A. UN\ z: B(x). C(z)) = (UN\ z: (UN\ x:A. B(x)). C(z))$$

$\langle proof \rangle$

lemmas $UN-extend-simps = UN-extend-simps1\ UN-extend-simps2\ UN-UN-extend$

2.15 Miniscoping of Intersections

lemma $INT-simps1$:

$$\begin{aligned} (INT\ x:C. A(x)\ Int\ B) &= (INT\ x:C. A(x))\ Int\ B \\ (INT\ x:C. A(x) - B) &= (INT\ x:C. A(x)) - B \\ (INT\ x:C. A(x)\ Un\ B) &= (if\ C=0\ then\ 0\ else\ (INT\ x:C. A(x))\ Un\ B) \end{aligned}$$

$\langle proof \rangle$

lemma $INT-simps2$:

$$\begin{aligned} (INT\ x:C. A\ Int\ B(x)) &= A\ Int\ (INT\ x:C. B(x)) \\ (INT\ x:C. A - B(x)) &= (if\ C=0\ then\ 0\ else\ A - (UN\ x:C. B(x))) \\ (INT\ x:C. cons(a, B(x))) &= (if\ C=0\ then\ 0\ else\ cons(a, INT\ x:C. B(x))) \\ (INT\ x:C. A\ Un\ B(x)) &= (if\ C=0\ then\ 0\ else\ A\ Un\ (INT\ x:C. B(x))) \end{aligned}$$

$\langle proof \rangle$

lemmas $INT-simps\ [simp] = INT-simps1\ INT-simps2$

Opposite of miniscoping: pull the operator out

lemma $INT-extend-simps1$:

$$\begin{aligned} (INT\ x:C. A(x))\ Int\ B &= (INT\ x:C. A(x)\ Int\ B) \\ (INT\ x:C. A(x) - B) &= (INT\ x:C. A(x) - B) \end{aligned}$$

$(INT\ x:C. A(x))\ Un\ B = (if\ C=0\ then\ B\ else\ (INT\ x:C. A(x)\ Un\ B))$
 $\langle proof \rangle$

lemma *INT-extend-simps2*:

$A\ Int\ (INT\ x:C. B(x)) = (INT\ x:C. A\ Int\ B(x))$
 $A - (UN\ x:C. B(x)) = (if\ C=0\ then\ A\ else\ (INT\ x:C. A - B(x)))$
 $cons(a, INT\ x:C. B(x)) = (if\ C=0\ then\ \{a\}\ else\ (INT\ x:C. cons(a, B(x))))$
 $A\ Un\ (INT\ x:C. B(x)) = (if\ C=0\ then\ A\ else\ (INT\ x:C. A\ Un\ B(x)))$
 $\langle proof \rangle$

lemmas *INT-extend-simps* = *INT-extend-simps1* *INT-extend-simps2*

2.16 Other simprules

lemma *misc-simps* [*simp*]:

$0\ Un\ A = A$
 $A\ Un\ 0 = A$
 $0\ Int\ A = 0$
 $A\ Int\ 0 = 0$
 $0 - A = 0$
 $A - 0 = A$
 $Union(0) = 0$
 $Union(cons(b,A)) = b\ Un\ Union(A)$
 $Inter(\{b\}) = b$
 $\langle proof \rangle$

end

3 pair: Ordered Pairs

theory *pair* **imports** *upair*

uses *simpdata.ML* **begin**

lemma *singleton-eq-iff* [*iff*]: $\{a\} = \{b\} \iff a=b$
 $\langle proof \rangle$

lemma *doubleton-eq-iff*: $\{a,b\} = \{c,d\} \iff (a=c \ \& \ b=d) \mid (a=d \ \& \ b=c)$
 $\langle proof \rangle$

lemma *Pair-iff* [*simp*]: $\langle a,b \rangle = \langle c,d \rangle \iff a=c \ \& \ b=d$
 $\langle proof \rangle$

lemmas *Pair-inject* = *Pair-iff* [*THEN* *iffD1*, *THEN* *conjE*, *standard*, *elim!*]

lemmas *Pair-inject1* = *Pair-iff* [*THEN* *iffD1*, *THEN* *conjunct1*, *standard*]

lemmas *Pair-inject2* = *Pair-iff* [*THEN* *iffD1*, *THEN* *conjunct2*, *standard*]

lemma *Pair-not-0*: $\langle a, b \rangle \sim = 0$

$\langle proof \rangle$

lemmas *Pair-neq-0* = *Pair-not-0* [*THEN notE, standard, elim!*]

declare *sym* [*THEN Pair-neq-0, elim!*]

lemma *Pair-neq-fst*: $\langle a, b \rangle = a \implies P$

$\langle proof \rangle$

lemma *Pair-neq-snd*: $\langle a, b \rangle = b \implies P$

$\langle proof \rangle$

3.1 Sigma: Disjoint Union of a Family of Sets

Generalizes Cartesian product

lemma *Sigma-iff* [*simp*]: $\langle a, b \rangle : \text{Sigma}(A, B) \iff a:A \ \& \ b:B(a)$

$\langle proof \rangle$

lemma *SigmaI* [*TC, intro!*]: $\llbracket a:A; \ b:B(a) \rrbracket \implies \langle a, b \rangle : \text{Sigma}(A, B)$

$\langle proof \rangle$

lemmas *SigmaD1* = *Sigma-iff* [*THEN iffD1, THEN conjunct1, standard*]

lemmas *SigmaD2* = *Sigma-iff* [*THEN iffD1, THEN conjunct2, standard*]

lemma *SigmaE* [*elim!*]:

$\llbracket c : \text{Sigma}(A, B);$
 $\quad !!x \ y. \llbracket x:A; \ y:B(x); \ c = \langle x, y \rangle \rrbracket \implies P$
 $\rrbracket \implies P$

$\langle proof \rangle$

lemma *SigmaE2* [*elim!*]:

$\llbracket \langle a, b \rangle : \text{Sigma}(A, B);$
 $\quad \llbracket a:A; \ b:B(a) \rrbracket \implies P$
 $\rrbracket \implies P$

$\langle proof \rangle$

lemma *Sigma-cong*:

$\llbracket A=A'; \quad !!x. \ x:A' \implies B(x)=B'(x) \rrbracket \implies$
 $\quad \text{Sigma}(A, B) = \text{Sigma}(A', B')$

$\langle proof \rangle$

lemma *Sigma-empty1* [*simp*]: $\text{Sigma}(0, B) = 0$

$\langle proof \rangle$

lemma *Sigma-empty2* [simp]: $A * 0 = 0$
 $\langle proof \rangle$

lemma *Sigma-empty-iff*: $A * B = 0 \iff A = 0 \mid B = 0$
 $\langle proof \rangle$

3.2 Projections *fst* and *snd*

lemma *fst-conv* [simp]: $fst(\langle a, b \rangle) = a$
 $\langle proof \rangle$

lemma *snd-conv* [simp]: $snd(\langle a, b \rangle) = b$
 $\langle proof \rangle$

lemma *fst-type* [TC]: $p : Sigma(A, B) \implies fst(p) : A$
 $\langle proof \rangle$

lemma *snd-type* [TC]: $p : Sigma(A, B) \implies snd(p) : B(fst(p))$
 $\langle proof \rangle$

lemma *Pair-fst-snd-eq*: $a : Sigma(A, B) \implies \langle fst(a), snd(a) \rangle = a$
 $\langle proof \rangle$

3.3 The Eliminator, *split*

lemma *split* [simp]: $split(\%x y. c(x, y), \langle a, b \rangle) == c(a, b)$
 $\langle proof \rangle$

lemma *split-type* [TC]:

$$[\mid p : Sigma(A, B);$$

$$!!x y. [\mid x : A; y : B(x)] \implies c(x, y) : C(\langle x, y \rangle)$$

$$] \implies split(\%x y. c(x, y), p) : C(p)$$
 $\langle proof \rangle$

lemma *expand-split*:

$$u : A * B \implies$$

$$R(split(c, u)) \iff (ALL x : A. ALL y : B. u = \langle x, y \rangle \iff R(c(x, y)))$$
 $\langle proof \rangle$

3.4 A version of *split* for Formulae: Result Type *o*

lemma *splitI*: $R(a, b) \implies split(R, \langle a, b \rangle)$
 $\langle proof \rangle$

lemma *splitE*:

$$[\mid split(R, z); z : Sigma(A, B);$$

$$!!x y. [\mid z = \langle x, y \rangle; R(x, y)] \implies P$$

$$] \implies P$$
 $\langle proof \rangle$

lemma *splitD*: $\text{split}(R, \langle a, b \rangle) \implies R(a, b)$
 $\langle \text{proof} \rangle$

Complex rules for Sigma.

lemma *split-paired-Bex-Sigma* [simp]:
 $(\exists z \in \text{Sigma}(A, B). P(z)) \iff (\exists x \in A. \exists y \in B(x). P(\langle x, y \rangle))$
 $\langle \text{proof} \rangle$

lemma *split-paired-Ball-Sigma* [simp]:
 $(\forall z \in \text{Sigma}(A, B). P(z)) \iff (\forall x \in A. \forall y \in B(x). P(\langle x, y \rangle))$
 $\langle \text{proof} \rangle$

end

4 equalities: Basic Equalities and Inclusions

theory *equalities* **imports** *pair* **begin**

These cover union, intersection, converse, domain, range, etc. Philippe de Groote proved many of the inclusions.

lemma *in-mono*: $A \subseteq B \implies x \in A \implies x \in B$
 $\langle \text{proof} \rangle$

lemma *the-eq-0* [simp]: $(\text{THE } x. \text{False}) = 0$
 $\langle \text{proof} \rangle$

4.1 Bounded Quantifiers

The following are not added to the default simpset because (a) they duplicate the body and (b) there are no similar rules for *Int*.

lemma *ball-Un*: $(\forall x \in A \cup B. P(x)) \iff (\forall x \in A. P(x)) \ \& \ (\forall x \in B. P(x))$
 $\langle \text{proof} \rangle$

lemma *bex-Un*: $(\exists x \in A \cup B. P(x)) \iff (\exists x \in A. P(x)) \ \vee \ (\exists x \in B. P(x))$
 $\langle \text{proof} \rangle$

lemma *ball-UN*: $(\forall z \in (\bigcup x \in A. B(x)). P(z)) \iff (\forall x \in A. \forall z \in B(x). P(z))$
 $\langle \text{proof} \rangle$

lemma *bex-UN*: $(\exists z \in (\bigcup x \in A. B(x)). P(z)) \iff (\exists x \in A. \exists z \in B(x). P(z))$
 $\langle \text{proof} \rangle$

4.2 Converse of a Relation

lemma *converse-iff* [simp]: $\langle a, b \rangle \in \text{converse}(r) \iff \langle b, a \rangle \in r$
 <proof>

lemma *converseI* [intro!]: $\langle a, b \rangle \in r \implies \langle b, a \rangle \in \text{converse}(r)$
 <proof>

lemma *converseD*: $\langle a, b \rangle \in \text{converse}(r) \implies \langle b, a \rangle \in r$
 <proof>

lemma *converseE* [elim!]:

$$\begin{aligned} & [| \text{ } yx \in \text{converse}(r); \\ & \quad !!x \ y. [| \text{ } yx = \langle y, x \rangle; \langle x, y \rangle \in r \text{ } |] \implies P \text{ } |] \\ & \implies P \end{aligned}$$

 <proof>

lemma *converse-converse*: $r \subseteq \text{Sigma}(A, B) \implies \text{converse}(\text{converse}(r)) = r$
 <proof>

lemma *converse-type*: $r \subseteq A * B \implies \text{converse}(r) \subseteq B * A$
 <proof>

lemma *converse-prod* [simp]: $\text{converse}(A * B) = B * A$
 <proof>

lemma *converse-empty* [simp]: $\text{converse}(0) = 0$
 <proof>

lemma *converse-subset-iff*:

$$A \subseteq \text{Sigma}(X, Y) \implies \text{converse}(A) \subseteq \text{converse}(B) \iff A \subseteq B$$

 <proof>

4.3 Finite Set Constructions Using cons

lemma *cons-subsetI*: $[| \text{ } a \in C; B \subseteq C \text{ } |] \implies \text{cons}(a, B) \subseteq C$
 <proof>

lemma *subset-consI*: $B \subseteq \text{cons}(a, B)$
 <proof>

lemma *cons-subset-iff* [iff]: $\text{cons}(a, B) \subseteq C \iff a \in C \ \& \ B \subseteq C$
 <proof>

lemmas *cons-subsetE* = *cons-subset-iff* [THEN iffD1, THEN conjE, standard]

lemma *subset-empty-iff*: $A \subseteq 0 \iff A = 0$
 <proof>

lemma *subset-cons-iff*: $C \subseteq \text{cons}(a, B) \leftrightarrow C \subseteq B \mid (a \in C \ \& \ C - \{a\} \subseteq B)$
 $\langle \text{proof} \rangle$

lemma *cons-eq*: $\{a\} \cup B = \text{cons}(a, B)$
 $\langle \text{proof} \rangle$

lemma *cons-commute*: $\text{cons}(a, \text{cons}(b, C)) = \text{cons}(b, \text{cons}(a, C))$
 $\langle \text{proof} \rangle$

lemma *cons-absorb*: $a: B \implies \text{cons}(a, B) = B$
 $\langle \text{proof} \rangle$

lemma *cons-Diff*: $a: B \implies \text{cons}(a, B - \{a\}) = B$
 $\langle \text{proof} \rangle$

lemma *Diff-cons-eq*: $\text{cons}(a, B) - C = (\text{if } a \in C \text{ then } B - C \text{ else } \text{cons}(a, B - C))$
 $\langle \text{proof} \rangle$

lemma *equal-singleton* [rule-format]: $[\mid a: C; \ \forall y \in C. y = b \mid] \implies C = \{b\}$
 $\langle \text{proof} \rangle$

lemma [simp]: $\text{cons}(a, \text{cons}(a, B)) = \text{cons}(a, B)$
 $\langle \text{proof} \rangle$

lemma *singleton-subsetI*: $a \in C \implies \{a\} \subseteq C$
 $\langle \text{proof} \rangle$

lemma *singleton-subsetD*: $\{a\} \subseteq C \implies a \in C$
 $\langle \text{proof} \rangle$

lemma *subset-succI*: $i \subseteq \text{succ}(i)$
 $\langle \text{proof} \rangle$

lemma *succ-subsetI*: $[\mid i \in j; \ i \subseteq j \mid] \implies \text{succ}(i) \subseteq j$
 $\langle \text{proof} \rangle$

lemma *succ-subsetE*:
 $[\mid \text{succ}(i) \subseteq j; \ [\mid i \in j; \ i \subseteq j \mid] \implies P \mid] \implies P$
 $\langle \text{proof} \rangle$

lemma *succ-subset-iff*: $\text{succ}(a) \subseteq B \leftrightarrow (a \subseteq B \ \& \ a \in B)$
 $\langle \text{proof} \rangle$

4.4 Binary Intersection

lemma *Int-subset-iff*: $C \subseteq A \text{ Int } B \leftrightarrow C \subseteq A \ \& \ C \subseteq B$
<proof>

lemma *Int-lower1*: $A \text{ Int } B \subseteq A$
<proof>

lemma *Int-lower2*: $A \text{ Int } B \subseteq B$
<proof>

lemma *Int-greatest*: $[| C \subseteq A; C \subseteq B |] ==> C \subseteq A \text{ Int } B$
<proof>

lemma *Int-cons*: $\text{cons}(a, B) \text{ Int } C \subseteq \text{cons}(a, B \text{ Int } C)$
<proof>

lemma *Int-absorb [simp]*: $A \text{ Int } A = A$
<proof>

lemma *Int-left-absorb*: $A \text{ Int } (A \text{ Int } B) = A \text{ Int } B$
<proof>

lemma *Int-commute*: $A \text{ Int } B = B \text{ Int } A$
<proof>

lemma *Int-left-commute*: $A \text{ Int } (B \text{ Int } C) = B \text{ Int } (A \text{ Int } C)$
<proof>

lemma *Int-assoc*: $(A \text{ Int } B) \text{ Int } C = A \text{ Int } (B \text{ Int } C)$
<proof>

lemmas *Int-ac= Int-assoc Int-left-absorb Int-commute Int-left-commute*

lemma *Int-absorb1*: $B \subseteq A ==> A \cap B = B$
<proof>

lemma *Int-absorb2*: $A \subseteq B ==> A \cap B = A$
<proof>

lemma *Int-Un-distrib*: $A \text{ Int } (B \text{ Un } C) = (A \text{ Int } B) \text{ Un } (A \text{ Int } C)$
<proof>

lemma *Int-Un-distrib2*: $(B \text{ Un } C) \text{ Int } A = (B \text{ Int } A) \text{ Un } (C \text{ Int } A)$
<proof>

lemma *subset-Int-iff*: $A \subseteq B \leftrightarrow A \text{ Int } B = A$
<proof>

lemma *subset-Int-iff2*: $A \subseteq B \iff B \text{ Int } A = A$
 $\langle \text{proof} \rangle$

lemma *Int-Diff-eq*: $C \subseteq A \implies (A - B) \text{ Int } C = C - B$
 $\langle \text{proof} \rangle$

lemma *Int-cons-left*:
 $\text{cons}(a, A) \text{ Int } B = (\text{if } a \in B \text{ then } \text{cons}(a, A \text{ Int } B) \text{ else } A \text{ Int } B)$
 $\langle \text{proof} \rangle$

lemma *Int-cons-right*:
 $A \text{ Int } \text{cons}(a, B) = (\text{if } a \in A \text{ then } \text{cons}(a, A \text{ Int } B) \text{ else } A \text{ Int } B)$
 $\langle \text{proof} \rangle$

lemma *cons-Int-distrib*: $\text{cons}(x, A \cap B) = \text{cons}(x, A) \cap \text{cons}(x, B)$
 $\langle \text{proof} \rangle$

4.5 Binary Union

lemma *Un-subset-iff*: $A \cup B \subseteq C \iff A \subseteq C \ \& \ B \subseteq C$
 $\langle \text{proof} \rangle$

lemma *Un-upper1*: $A \subseteq A \cup B$
 $\langle \text{proof} \rangle$

lemma *Un-upper2*: $B \subseteq A \cup B$
 $\langle \text{proof} \rangle$

lemma *Un-least*: $[\mid A \subseteq C; \ B \subseteq C \mid] \implies A \cup B \subseteq C$
 $\langle \text{proof} \rangle$

lemma *Un-cons*: $\text{cons}(a, B) \cup C = \text{cons}(a, B \cup C)$
 $\langle \text{proof} \rangle$

lemma *Un-absorb [simp]*: $A \cup A = A$
 $\langle \text{proof} \rangle$

lemma *Un-left-absorb*: $A \cup (A \cup B) = A \cup B$
 $\langle \text{proof} \rangle$

lemma *Un-commute*: $A \cup B = B \cup A$
 $\langle \text{proof} \rangle$

lemma *Un-left-commute*: $A \cup (B \cup C) = B \cup (A \cup C)$
 $\langle \text{proof} \rangle$

lemma *Un-assoc*: $(A \cup B) \cup C = A \cup (B \cup C)$
 $\langle \text{proof} \rangle$

lemmas $Un-ac = Un-assoc \ Un-left-absorb \ Un-commute \ Un-left-commute$

lemma $Un-absorb1: A \subseteq B ==> A \cup B = B$
 $\langle proof \rangle$

lemma $Un-absorb2: B \subseteq A ==> A \cup B = A$
 $\langle proof \rangle$

lemma $Un-Int-distrib: (A \ Int \ B) \ Un \ C = (A \ Un \ C) \ Int \ (B \ Un \ C)$
 $\langle proof \rangle$

lemma $subset-Un-iff: A \subseteq B <-> A \ Un \ B = B$
 $\langle proof \rangle$

lemma $subset-Un-iff2: A \subseteq B <-> B \ Un \ A = B$
 $\langle proof \rangle$

lemma $Un-empty \ [iff]: (A \ Un \ B = 0) <-> (A = 0 \ \& \ B = 0)$
 $\langle proof \rangle$

lemma $Un-eq-Union: A \ Un \ B = Union(\{A, B\})$
 $\langle proof \rangle$

4.6 Set Difference

lemma $Diff-subset: A - B \subseteq A$
 $\langle proof \rangle$

lemma $Diff-contains: [| \ C \subseteq A; \ C \ Int \ B = 0 \ |] ==> C \subseteq A - B$
 $\langle proof \rangle$

lemma $subset-Diff-cons-iff: B \subseteq A - cons(c, C) <-> B \subseteq A - C \ \& \ c \sim: B$
 $\langle proof \rangle$

lemma $Diff-cancel: A - A = 0$
 $\langle proof \rangle$

lemma $Diff-triv: A \ Int \ B = 0 ==> A - B = A$
 $\langle proof \rangle$

lemma $empty-Diff \ [simp]: 0 - A = 0$
 $\langle proof \rangle$

lemma $Diff-0 \ [simp]: A - 0 = A$
 $\langle proof \rangle$

lemma $Diff-eq-0-iff: A - B = 0 <-> A \subseteq B$
 $\langle proof \rangle$

lemma *Diff-cons*: $A - \text{cons}(a, B) = A - B - \{a\}$
 $\langle \text{proof} \rangle$

lemma *Diff-cons2*: $A - \text{cons}(a, B) = A - \{a\} - B$
 $\langle \text{proof} \rangle$

lemma *Diff-disjoint*: $A \text{ Int } (B - A) = 0$
 $\langle \text{proof} \rangle$

lemma *Diff-partition*: $A \subseteq B \implies A \text{ Un } (B - A) = B$
 $\langle \text{proof} \rangle$

lemma *subset-Un-Diff*: $A \subseteq B \text{ Un } (A - B)$
 $\langle \text{proof} \rangle$

lemma *double-complement*: $[A \subseteq B; B \subseteq C] \implies B - (C - A) = A$
 $\langle \text{proof} \rangle$

lemma *double-complement-Un*: $(A \text{ Un } B) - (B - A) = A$
 $\langle \text{proof} \rangle$

lemma *Un-Int-crazy*:
 $(A \text{ Int } B) \text{ Un } (B \text{ Int } C) \text{ Un } (C \text{ Int } A) = (A \text{ Un } B) \text{ Int } (B \text{ Un } C) \text{ Int } (C \text{ Un } A)$
 $\langle \text{proof} \rangle$

lemma *Diff-Un*: $A - (B \text{ Un } C) = (A - B) \text{ Int } (A - C)$
 $\langle \text{proof} \rangle$

lemma *Diff-Int*: $A - (B \text{ Int } C) = (A - B) \text{ Un } (A - C)$
 $\langle \text{proof} \rangle$

lemma *Un-Diff*: $(A \text{ Un } B) - C = (A - C) \text{ Un } (B - C)$
 $\langle \text{proof} \rangle$

lemma *Int-Diff*: $(A \text{ Int } B) - C = A \text{ Int } (B - C)$
 $\langle \text{proof} \rangle$

lemma *Diff-Int-distrib*: $C \text{ Int } (A - B) = (C \text{ Int } A) - (C \text{ Int } B)$
 $\langle \text{proof} \rangle$

lemma *Diff-Int-distrib2*: $(A - B) \text{ Int } C = (A \text{ Int } C) - (B \text{ Int } C)$
 $\langle \text{proof} \rangle$

lemma *Un-Int-assoc-iff*: $(A \text{ Int } B) \text{ Un } C = A \text{ Int } (B \text{ Un } C) \iff C \subseteq A$
 $\langle \text{proof} \rangle$

4.7 Big Union and Intersection

lemma *Union-subset-iff*: $Union(A) \subseteq C \leftrightarrow (\forall x \in A. x \subseteq C)$
 $\langle proof \rangle$

lemma *Union-upper*: $B \in A \implies B \subseteq Union(A)$
 $\langle proof \rangle$

lemma *Union-least*: $[\mid \forall x. x \in A \implies x \subseteq C \mid] \implies Union(A) \subseteq C$
 $\langle proof \rangle$

lemma *Union-cons* [simp]: $Union(cons(a, B)) = a \cup Union(B)$
 $\langle proof \rangle$

lemma *Union-Un-distrib*: $Union(A \cup B) = Union(A) \cup Union(B)$
 $\langle proof \rangle$

lemma *Union-Int-subset*: $Union(A \cap B) \subseteq Union(A) \cap Union(B)$
 $\langle proof \rangle$

lemma *Union-disjoint*: $Union(C) \cap A = 0 \leftrightarrow (\forall B \in C. B \cap A = 0)$
 $\langle proof \rangle$

lemma *Union-empty-iff*: $Union(A) = 0 \leftrightarrow (\forall B \in A. B = 0)$
 $\langle proof \rangle$

lemma *Int-Union2*: $Union(B) \cap A = (\bigcup C \in B. C \cap A)$
 $\langle proof \rangle$

lemma *Inter-subset-iff*: $A \neq 0 \implies C \subseteq Inter(A) \leftrightarrow (\forall x \in A. C \subseteq x)$
 $\langle proof \rangle$

lemma *Inter-lower*: $B \in A \implies Inter(A) \subseteq B$
 $\langle proof \rangle$

lemma *Inter-greatest*: $[\mid A \neq 0; \forall x. x \in A \implies C \subseteq x \mid] \implies C \subseteq Inter(A)$
 $\langle proof \rangle$

lemma *INT-lower*: $x \in A \implies (\bigcap x \in A. B(x)) \subseteq B(x)$
 $\langle proof \rangle$

lemma *INT-greatest*: $[\mid A \neq 0; \forall x. x \in A \implies C \subseteq B(x) \mid] \implies C \subseteq (\bigcap x \in A. B(x))$
 $\langle proof \rangle$

lemma *Inter-0* [simp]: $Inter(0) = 0$

$\langle proof \rangle$

lemma *Inter-Un-subset*:

$\llbracket z \in A; z \in B \rrbracket \implies Inter(A) \cup Inter(B) \subseteq Inter(A \text{ Int } B)$
 $\langle proof \rangle$

lemma *Inter-Un-distrib*:

$\llbracket A \neq 0; B \neq 0 \rrbracket \implies Inter(A \cup B) = Inter(A) \text{ Int } Inter(B)$
 $\langle proof \rangle$

lemma *Union-singleton*: $Union(\{b\}) = b$

$\langle proof \rangle$

lemma *Inter-singleton*: $Inter(\{b\}) = b$

$\langle proof \rangle$

lemma *Inter-cons* [simp]:

$Inter(cons(a, B)) = (\text{if } B=0 \text{ then } a \text{ else } a \text{ Int } Inter(B))$
 $\langle proof \rangle$

4.8 Unions and Intersections of Families

lemma *subset-UN-iff-eq*: $A \subseteq (\bigcup i \in I. B(i)) \iff A = (\bigcup i \in I. A \text{ Int } B(i))$

$\langle proof \rangle$

lemma *UN-subset-iff*: $(\bigcup x \in A. B(x)) \subseteq C \iff (\forall x \in A. B(x) \subseteq C)$

$\langle proof \rangle$

lemma *UN-upper*: $x \in A \implies B(x) \subseteq (\bigcup x \in A. B(x))$

$\langle proof \rangle$

lemma *UN-least*: $\llbracket \exists x. x \in A \implies B(x) \subseteq C \rrbracket \implies (\bigcup x \in A. B(x)) \subseteq C$

$\langle proof \rangle$

lemma *Union-eq-UN*: $Union(A) = (\bigcup x \in A. x)$

$\langle proof \rangle$

lemma *Inter-eq-INT*: $Inter(A) = (\bigcap x \in A. x)$

$\langle proof \rangle$

lemma *UN-0* [simp]: $(\bigcup i \in 0. A(i)) = 0$

$\langle proof \rangle$

lemma *UN-singleton*: $(\bigcup x \in A. \{x\}) = A$

$\langle proof \rangle$

lemma *UN-Un*: $(\bigcup i \in A \cup B. C(i)) = (\bigcup i \in A. C(i)) \cup (\bigcup i \in B. C(i))$

$\langle proof \rangle$

lemma *INT-Un*: $(\bigcap_{i \in I} \text{Un } J. A(i)) =$
 (if $I=0$ *then* $\bigcap_{j \in J. A(j)}$
 else if $J=0$ *then* $\bigcap_{i \in I. A(i)}$
 else $((\bigcap_{i \in I. A(i)) \text{Int } (\bigcap_{j \in J. A(j))})$
 $\langle \text{proof} \rangle$

lemma *UN-UN-flatten*: $(\bigcup x \in (\bigcup_{y \in A. B(y)). C(x)) = (\bigcup_{y \in A. \bigcup_{x \in B(y). C(x)})$
 $\langle \text{proof} \rangle$

lemma *Int-UN-distrib*: $B \text{Int } (\bigcup_{i \in I. A(i)) = (\bigcup_{i \in I. B \text{Int } A(i)}$
 $\langle \text{proof} \rangle$

lemma *Un-INT-distrib*: $I \neq 0 \implies B \text{Un } (\bigcap_{i \in I. A(i)) = (\bigcap_{i \in I. B \text{Un } A(i)}$
 $\langle \text{proof} \rangle$

lemma *Int-UN-distrib2*:
 $(\bigcup_{i \in I. A(i)) \text{Int } (\bigcup_{j \in J. B(j)) = (\bigcup_{i \in I. \bigcup_{j \in J. A(i) \text{Int } B(j)}$
 $\langle \text{proof} \rangle$

lemma *Un-INT-distrib2*: $[I \neq 0; J \neq 0] \implies$
 $(\bigcap_{i \in I. A(i)) \text{Un } (\bigcap_{j \in J. B(j)) = (\bigcap_{i \in I. \bigcap_{j \in J. A(i) \text{Un } B(j)}$
 $\langle \text{proof} \rangle$

lemma *UN-constant [simp]*: $(\bigcup_{y \in A. c) = (\text{if } A=0 \text{ then } 0 \text{ else } c)$
 $\langle \text{proof} \rangle$

lemma *INT-constant [simp]*: $(\bigcap_{y \in A. c) = (\text{if } A=0 \text{ then } 0 \text{ else } c)$
 $\langle \text{proof} \rangle$

lemma *UN-RepFun [simp]*: $(\bigcup_{y \in \text{RepFun}(A, f). B(y)) = (\bigcup_{x \in A. B(f(x)))$
 $\langle \text{proof} \rangle$

lemma *INT-RepFun [simp]*: $(\bigcap_{x \in \text{RepFun}(A, f). B(x)) = (\bigcap_{a \in A. B(f(a)))$
 $\langle \text{proof} \rangle$

lemma *INT-Union-eq*:
 $0 \sim: A \implies (\bigcap_{x \in \text{Union}(A). B(x)) = (\bigcap_{y \in A. \bigcap_{x \in y. B(x)})$
 $\langle \text{proof} \rangle$

lemma *INT-UN-eq*:
 $(\forall x \in A. B(x) \sim 0)$
 $\implies (\bigcap_{z \in (\bigcup_{x \in A. B(x)). C(z)) = (\bigcap_{x \in A. \bigcap_{z \in B(x). C(z)})$
 $\langle \text{proof} \rangle$

lemma *UN-Un-distrib*:

$$(\bigcup_{i \in I}. A(i) \text{ Un } B(i)) = (\bigcup_{i \in I}. A(i)) \text{ Un } (\bigcup_{i \in I}. B(i))$$

<proof>

lemma *INT-Int-distrib*:

$$I \neq 0 \implies (\bigcap_{i \in I}. A(i) \text{ Int } B(i)) = (\bigcap_{i \in I}. A(i)) \text{ Int } (\bigcap_{i \in I}. B(i))$$

<proof>

lemma *UN-Int-subset*:

$$(\bigcup_{z \in I \text{ Int } J}. A(z)) \subseteq (\bigcup_{z \in I}. A(z)) \text{ Int } (\bigcup_{z \in J}. A(z))$$

<proof>

lemma *Diff-UN*: $I \neq 0 \implies B - (\bigcup_{i \in I}. A(i)) = (\bigcap_{i \in I}. B - A(i))$

<proof>

lemma *Diff-INT*: $I \neq 0 \implies B - (\bigcap_{i \in I}. A(i)) = (\bigcup_{i \in I}. B - A(i))$

<proof>

lemma *Sigma-cons1*: $\text{Sigma}(\text{cons}(a, B), C) = (\{a\} * C(a)) \text{ Un } \text{Sigma}(B, C)$

<proof>

lemma *Sigma-cons2*: $A * \text{cons}(b, B) = A * \{b\} \text{ Un } A * B$

<proof>

lemma *Sigma-succ1*: $\text{Sigma}(\text{succ}(A), B) = (\{A\} * B(A)) \text{ Un } \text{Sigma}(A, B)$

<proof>

lemma *Sigma-succ2*: $A * \text{succ}(B) = A * \{B\} \text{ Un } A * B$

<proof>

lemma *SUM-UN-distrib1*:

$$(\sum x \in (\bigcup_{y \in A}. C(y)). B(x)) = (\bigcup_{y \in A}. \sum x \in C(y). B(x))$$

<proof>

lemma *SUM-UN-distrib2*:

$$(\sum i \in I. \bigcup_{j \in J}. C(i, j)) = (\bigcup_{j \in J}. \sum i \in I. C(i, j))$$

<proof>

lemma *SUM-Un-distrib1*:

$$(\sum i \in I \text{ Un } J. C(i)) = (\sum i \in I. C(i)) \text{ Un } (\sum j \in J. C(j))$$

<proof>

lemma *SUM-Un-distrib2*:

$$(\Sigma i \in I. A(i) \text{ Un } B(i)) = (\Sigma i \in I. A(i)) \text{ Un } (\Sigma i \in I. B(i))$$

<proof>

lemma *prod-Un-distrib2*: $I * (A \text{ Un } B) = I * A \text{ Un } I * B$

<proof>

lemma *SUM-Int-distrib1*:

$$(\Sigma i \in I \text{ Int } J. C(i)) = (\Sigma i \in I. C(i)) \text{ Int } (\Sigma j \in J. C(j))$$

<proof>

lemma *SUM-Int-distrib2*:

$$(\Sigma i \in I. A(i) \text{ Int } B(i)) = (\Sigma i \in I. A(i)) \text{ Int } (\Sigma i \in I. B(i))$$

<proof>

lemma *prod-Int-distrib2*: $I * (A \text{ Int } B) = I * A \text{ Int } I * B$

<proof>

lemma *SUM-eq-UN*: $(\Sigma i \in I. A(i)) = (\bigcup i \in I. \{i\} * A(i))$

<proof>

lemma *times-subset-iff*:

$$(A' * B' \subseteq A * B) \leftrightarrow (A' = 0 \mid B' = 0 \mid (A' \subseteq A) \ \& \ (B' \subseteq B))$$

<proof>

lemma *Int-Sigma-eq*:

$$(\Sigma x \in A'. B'(x)) \text{ Int } (\Sigma x \in A. B(x)) = (\Sigma x \in A' \text{ Int } A. B'(x)) \text{ Int } B(x)$$

<proof>

lemma *domain-iff*: $a: \text{domain}(r) \leftrightarrow (EX y. \langle a, y \rangle \in r)$

<proof>

lemma *domainI [intro]*: $\langle a, b \rangle \in r \implies a: \text{domain}(r)$

<proof>

lemma *domainE [elim!]*:

$$[\mid a \in \text{domain}(r); \ !y. \langle a, y \rangle \in r \implies P \mid] \implies P$$

<proof>

lemma *domain-subset*: $\text{domain}(\text{Sigma}(A, B)) \subseteq A$

<proof>

lemma *domain-of-prod*: $b \in B \implies \text{domain}(A * B) = A$

$\langle \text{proof} \rangle$

lemma *domain-0* [simp]: $\text{domain}(0) = 0$
 $\langle \text{proof} \rangle$

lemma *domain-cons* [simp]: $\text{domain}(\text{cons}(\langle a, b \rangle, r)) = \text{cons}(a, \text{domain}(r))$
 $\langle \text{proof} \rangle$

lemma *domain-Un-eq* [simp]: $\text{domain}(A \text{ Un } B) = \text{domain}(A) \text{ Un } \text{domain}(B)$
 $\langle \text{proof} \rangle$

lemma *domain-Int-subset*: $\text{domain}(A \text{ Int } B) \subseteq \text{domain}(A) \text{ Int } \text{domain}(B)$
 $\langle \text{proof} \rangle$

lemma *domain-Diff-subset*: $\text{domain}(A) - \text{domain}(B) \subseteq \text{domain}(A - B)$
 $\langle \text{proof} \rangle$

lemma *domain-UN*: $\text{domain}(\bigcup x \in A. B(x)) = (\bigcup x \in A. \text{domain}(B(x)))$
 $\langle \text{proof} \rangle$

lemma *domain-Union*: $\text{domain}(\text{Union}(A)) = (\bigcup x \in A. \text{domain}(x))$
 $\langle \text{proof} \rangle$

lemma *rangeI* [intro]: $\langle a, b \rangle \in r \implies b \in \text{range}(r)$
 $\langle \text{proof} \rangle$

lemma *rangeE* [elim!]: $[\![\ b \in \text{range}(r); \ !x. \langle x, b \rangle \in r \implies P \]\!] \implies P$
 $\langle \text{proof} \rangle$

lemma *range-subset*: $\text{range}(A * B) \subseteq B$
 $\langle \text{proof} \rangle$

lemma *range-of-prod*: $a \in A \implies \text{range}(A * B) = B$
 $\langle \text{proof} \rangle$

lemma *range-0* [simp]: $\text{range}(0) = 0$
 $\langle \text{proof} \rangle$

lemma *range-cons* [simp]: $\text{range}(\text{cons}(\langle a, b \rangle, r)) = \text{cons}(b, \text{range}(r))$
 $\langle \text{proof} \rangle$

lemma *range-Un-eq* [simp]: $\text{range}(A \text{ Un } B) = \text{range}(A) \text{ Un } \text{range}(B)$
 $\langle \text{proof} \rangle$

lemma *range-Int-subset*: $\text{range}(A \text{ Int } B) \subseteq \text{range}(A) \text{ Int } \text{range}(B)$
 $\langle \text{proof} \rangle$

lemma *range-Diff-subset*: $\text{range}(A) - \text{range}(B) \subseteq \text{range}(A - B)$
 $\langle \text{proof} \rangle$

lemma *domain-converse* [simp]: $\text{domain}(\text{converse}(r)) = \text{range}(r)$
 $\langle \text{proof} \rangle$

lemma *range-converse* [simp]: $\text{range}(\text{converse}(r)) = \text{domain}(r)$
 $\langle \text{proof} \rangle$

lemma *fieldI1*: $\langle a, b \rangle \in r \implies a \in \text{field}(r)$
 $\langle \text{proof} \rangle$

lemma *fieldI2*: $\langle a, b \rangle \in r \implies b \in \text{field}(r)$
 $\langle \text{proof} \rangle$

lemma *fieldCI* [intro]:
 $(\sim \langle c, a \rangle \in r \implies \langle a, b \rangle \in r) \implies a \in \text{field}(r)$
 $\langle \text{proof} \rangle$

lemma *fieldE* [elim!]:
 $\llbracket a \in \text{field}(r);$
 $\quad \llbracket x. \langle a, x \rangle \in r \implies P;$
 $\quad \llbracket x. \langle x, a \rangle \in r \implies P \quad \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *field-subset*: $\text{field}(A * B) \subseteq A \cup B$
 $\langle \text{proof} \rangle$

lemma *domain-subset-field*: $\text{domain}(r) \subseteq \text{field}(r)$
 $\langle \text{proof} \rangle$

lemma *range-subset-field*: $\text{range}(r) \subseteq \text{field}(r)$
 $\langle \text{proof} \rangle$

lemma *domain-times-range*: $r \subseteq \text{Sigma}(A, B) \implies r \subseteq \text{domain}(r) * \text{range}(r)$
 $\langle \text{proof} \rangle$

lemma *field-times-field*: $r \subseteq \text{Sigma}(A, B) \implies r \subseteq \text{field}(r) * \text{field}(r)$
 $\langle \text{proof} \rangle$

lemma *relation-field-times-field*: $\text{relation}(r) \implies r \subseteq \text{field}(r) * \text{field}(r)$
 $\langle \text{proof} \rangle$

lemma *field-of-prod*: $\text{field}(A * A) = A$
 $\langle \text{proof} \rangle$

lemma *field-0* [*simp*]: $\text{field}(0) = 0$

$\langle \text{proof} \rangle$

lemma *field-cons* [*simp*]: $\text{field}(\text{cons}(\langle a, b \rangle, r)) = \text{cons}(a, \text{cons}(b, \text{field}(r)))$

$\langle \text{proof} \rangle$

lemma *field-Un-eq* [*simp*]: $\text{field}(A \text{ Un } B) = \text{field}(A) \text{ Un } \text{field}(B)$

$\langle \text{proof} \rangle$

lemma *field-Int-subset*: $\text{field}(A \text{ Int } B) \subseteq \text{field}(A) \text{ Int } \text{field}(B)$

$\langle \text{proof} \rangle$

lemma *field-Diff-subset*: $\text{field}(A) - \text{field}(B) \subseteq \text{field}(A - B)$

$\langle \text{proof} \rangle$

lemma *field-converse* [*simp*]: $\text{field}(\text{converse}(r)) = \text{field}(r)$

$\langle \text{proof} \rangle$

lemma *rel-Union*: $(\forall x \in S. \exists X A B. x \subseteq A * B) ==>$

$\text{Union}(S) \subseteq \text{domain}(\text{Union}(S)) * \text{range}(\text{Union}(S))$

$\langle \text{proof} \rangle$

lemma *rel-Un*: $[\mid r \subseteq A * B; s \subseteq C * D \mid] ==> (r \text{ Un } s) \subseteq (A \text{ Un } C) * (B \text{ Un } D)$

$\langle \text{proof} \rangle$

lemma *domain-Diff-eq*: $[\mid \langle a, c \rangle \in r; c \sim b \mid] ==> \text{domain}(r - \{\langle a, b \rangle\}) = \text{domain}(r)$

$\langle \text{proof} \rangle$

lemma *range-Diff-eq*: $[\mid \langle c, b \rangle \in r; c \sim a \mid] ==> \text{range}(r - \{\langle a, b \rangle\}) = \text{range}(r)$

$\langle \text{proof} \rangle$

4.9 Image of a Set under a Function or Relation

lemma *image-iff*: $b \in r''A \iff (\exists x \in A. \langle x, b \rangle \in r)$

$\langle \text{proof} \rangle$

lemma *image-singleton-iff*: $b \in r''\{a\} \iff \langle a, b \rangle \in r$

$\langle \text{proof} \rangle$

lemma *imageI* [*intro*]: $[\mid \langle a, b \rangle \in r; a \in A \mid] ==> b \in r''A$

$\langle \text{proof} \rangle$

lemma *imageE* [*elim!*]:

$[\mid b \in r''A; !!x. [\mid \langle x, b \rangle \in r; x \in A \mid] ==> P \mid] ==> P$

$\langle \text{proof} \rangle$

lemma *image-subset*: $r \subseteq A*B \implies r''C \subseteq B$

<proof>

lemma *image-0 [simp]*: $r''0 = 0$

<proof>

lemma *image-Un [simp]*: $r''(A \text{ Un } B) = (r''A) \text{ Un } (r''B)$

<proof>

lemma *image-UN*: $r''(\bigcup_{x \in A}. B(x)) = (\bigcup_{x \in A}. r''B(x))$

<proof>

lemma *Collect-image-eq*:

$\{z \in \text{Sigma}(A,B). P(z)\}''C = (\bigcup_{x \in A}. \{y \in B(x). x \in C \ \& \ P(\langle x,y \rangle)\})$

<proof>

lemma *image-Int-subset*: $r''(A \text{ Int } B) \subseteq (r''A) \text{ Int } (r''B)$

<proof>

lemma *image-Int-square-subset*: $(r \text{ Int } A*A)''B \subseteq (r''B) \text{ Int } A$

<proof>

lemma *image-Int-square*: $B \subseteq A \implies (r \text{ Int } A*A)''B = (r''B) \text{ Int } A$

<proof>

lemma *image-0-left [simp]*: $0''A = 0$

<proof>

lemma *image-Un-left*: $(r \text{ Un } s)''A = (r''A) \text{ Un } (s''A)$

<proof>

lemma *image-Int-subset-left*: $(r \text{ Int } s)''A \subseteq (r''A) \text{ Int } (s''A)$

<proof>

4.10 Inverse Image of a Set under a Function or Relation

lemma *vimage-iff*:

$a \in r-''B \iff (\exists y \in B. \langle a,y \rangle \in r)$

<proof>

lemma *vimage-singleton-iff*: $a \in r-''\{b\} \iff \langle a,b \rangle \in r$

<proof>

lemma *vimageI [intro]*: $[\langle a,b \rangle \in r; b \in B] \implies a \in r-''B$

<proof>

lemma *vimageE* [*elim!*]:

$\llbracket a: r - \text{``}B; !!x. \llbracket \langle a, x \rangle \in r; x \in B \rrbracket \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *vimage-subset*: $r \subseteq A * B \implies r - \text{``}C \subseteq A$

$\langle \text{proof} \rangle$

lemma *vimage-0* [*simp*]: $r - \text{``}0 = 0$

$\langle \text{proof} \rangle$

lemma *vimage-Un* [*simp*]: $r - \text{``}(A \text{ Un } B) = (r - \text{``}A) \text{ Un } (r - \text{``}B)$

$\langle \text{proof} \rangle$

lemma *vimage-Int-subset*: $r - \text{``}(A \text{ Int } B) \subseteq (r - \text{``}A) \text{ Int } (r - \text{``}B)$

$\langle \text{proof} \rangle$

lemma *vimage-eq-UN*: $f - \text{``}B = (\bigcup_{y \in B}. f - \text{``}\{y\})$

$\langle \text{proof} \rangle$

lemma *function-vimage-Int*:

$\text{function}(f) \implies f - \text{``}(A \text{ Int } B) = (f - \text{``}A) \text{ Int } (f - \text{``}B)$
 $\langle \text{proof} \rangle$

lemma *function-vimage-Diff*: $\text{function}(f) \implies f - \text{``}(A - B) = (f - \text{``}A) - (f - \text{``}B)$

$\langle \text{proof} \rangle$

lemma *function-image-vimage*: $\text{function}(f) \implies f \text{ `` } (f - \text{``} A) \subseteq A$

$\langle \text{proof} \rangle$

lemma *vimage-Int-square-subset*: $(r \text{ Int } A * A) - \text{``}B \subseteq (r - \text{``}B) \text{ Int } A$

$\langle \text{proof} \rangle$

lemma *vimage-Int-square*: $B \subseteq A \implies (r \text{ Int } A * A) - \text{``}B = (r - \text{``}B) \text{ Int } A$

$\langle \text{proof} \rangle$

lemma *vimage-0-left* [*simp*]: $0 - \text{``}A = 0$

$\langle \text{proof} \rangle$

lemma *vimage-Un-left*: $(r \text{ Un } s) - \text{``}A = (r - \text{``}A) \text{ Un } (s - \text{``}A)$

$\langle \text{proof} \rangle$

lemma *vimage-Int-subset-left*: $(r \text{ Int } s) - \text{``}A \subseteq (r - \text{``}A) \text{ Int } (s - \text{``}A)$

$\langle \text{proof} \rangle$

lemma *converse-Un* [simp]: $\text{converse}(A \text{ Un } B) = \text{converse}(A) \text{ Un } \text{converse}(B)$
 ⟨proof⟩

lemma *converse-Int* [simp]: $\text{converse}(A \text{ Int } B) = \text{converse}(A) \text{ Int } \text{converse}(B)$
 ⟨proof⟩

lemma *converse-Diff* [simp]: $\text{converse}(A - B) = \text{converse}(A) - \text{converse}(B)$
 ⟨proof⟩

lemma *converse-UN* [simp]: $\text{converse}(\bigcup x \in A. B(x)) = (\bigcup x \in A. \text{converse}(B(x)))$
 ⟨proof⟩

lemma *converse-INT* [simp]:
 $\text{converse}(\bigcap x \in A. B(x)) = (\bigcap x \in A. \text{converse}(B(x)))$
 ⟨proof⟩

4.11 Powerset Operator

lemma *Pow-0* [simp]: $\text{Pow}(0) = \{0\}$
 ⟨proof⟩

lemma *Pow-insert*: $\text{Pow}(\text{cons}(a, A)) = \text{Pow}(A) \text{ Un } \{\text{cons}(a, X) \mid X \in \text{Pow}(A)\}$
 ⟨proof⟩

lemma *Un-Pow-subset*: $\text{Pow}(A) \text{ Un } \text{Pow}(B) \subseteq \text{Pow}(A \text{ Un } B)$
 ⟨proof⟩

lemma *UN-Pow-subset*: $(\bigcup x \in A. \text{Pow}(B(x))) \subseteq \text{Pow}(\bigcup x \in A. B(x))$
 ⟨proof⟩

lemma *subset-Pow-Union*: $A \subseteq \text{Pow}(\text{Union}(A))$
 ⟨proof⟩

lemma *Union-Pow-eq* [simp]: $\text{Union}(\text{Pow}(A)) = A$
 ⟨proof⟩

lemma *Union-Pow-iff*: $\text{Union}(A) \in \text{Pow}(B) \iff A \in \text{Pow}(\text{Pow}(B))$
 ⟨proof⟩

lemma *Pow-Int-eq* [simp]: $\text{Pow}(A \text{ Int } B) = \text{Pow}(A) \text{ Int } \text{Pow}(B)$
 ⟨proof⟩

lemma *Pow-INT-eq*: $A \neq 0 \implies \text{Pow}(\bigcap x \in A. B(x)) = (\bigcap x \in A. \text{Pow}(B(x)))$
 ⟨proof⟩

4.12 RepFun

lemma *RepFun-subset*: $[[\text{!!}x. x \in A \implies f(x) \in B]] \implies \{f(x). x \in A\} \subseteq B$
 $\langle \text{proof} \rangle$

lemma *RepFun-eq-0-iff* [simp]: $\{f(x). x \in A\} = 0 \iff A = 0$
 $\langle \text{proof} \rangle$

lemma *RepFun-constant* [simp]: $\{c. x \in A\} = (\text{if } A = 0 \text{ then } 0 \text{ else } \{c\})$
 $\langle \text{proof} \rangle$

4.13 Collect

lemma *Collect-subset*: $\text{Collect}(A, P) \subseteq A$
 $\langle \text{proof} \rangle$

lemma *Collect-Un*: $\text{Collect}(A \text{ Un } B, P) = \text{Collect}(A, P) \text{ Un } \text{Collect}(B, P)$
 $\langle \text{proof} \rangle$

lemma *Collect-Int*: $\text{Collect}(A \text{ Int } B, P) = \text{Collect}(A, P) \text{ Int } \text{Collect}(B, P)$
 $\langle \text{proof} \rangle$

lemma *Collect-Diff*: $\text{Collect}(A - B, P) = \text{Collect}(A, P) - \text{Collect}(B, P)$
 $\langle \text{proof} \rangle$

lemma *Collect-cons*: $\{x \in \text{cons}(a, B). P(x)\} =$
 $(\text{if } P(a) \text{ then } \text{cons}(a, \{x \in B. P(x)\}) \text{ else } \{x \in B. P(x)\})$
 $\langle \text{proof} \rangle$

lemma *Int-Collect-self-eq*: $A \text{ Int } \text{Collect}(A, P) = \text{Collect}(A, P)$
 $\langle \text{proof} \rangle$

lemma *Collect-Collect-eq* [simp]:
 $\text{Collect}(\text{Collect}(A, P), Q) = \text{Collect}(A, \%x. P(x) \ \& \ Q(x))$
 $\langle \text{proof} \rangle$

lemma *Collect-Int-Collect-eq*:
 $\text{Collect}(A, P) \text{ Int } \text{Collect}(A, Q) = \text{Collect}(A, \%x. P(x) \ \& \ Q(x))$
 $\langle \text{proof} \rangle$

lemma *Collect-Union-eq* [simp]:
 $\text{Collect}(\bigcup x \in A. B(x), P) = (\bigcup x \in A. \text{Collect}(B(x), P))$
 $\langle \text{proof} \rangle$

lemma *Collect-Int-left*: $\{x \in A. P(x)\} \text{ Int } B = \{x \in A \text{ Int } B. P(x)\}$
 $\langle \text{proof} \rangle$

lemma *Collect-Int-right*: $A \text{ Int } \{x \in B. P(x)\} = \{x \in A \text{ Int } B. P(x)\}$
 $\langle \text{proof} \rangle$

lemma *Collect-disj-eq*: $\{x \in A. P(x) \mid Q(x)\} = \text{Collect}(A, P) \text{ Un } \text{Collect}(A, Q)$
 $\langle \text{proof} \rangle$

lemma *Collect-conj-eq*: $\{x \in A. P(x) \ \& \ Q(x)\} = \text{Collect}(A, P) \text{ Int } \text{Collect}(A, Q)$
 $\langle \text{proof} \rangle$

lemmas *subset-SIs* = *subset-refl cons-subsetI subset-consI*
Union-least UN-least Un-least
Inter-greatest Int-greatest RepFun-subset
Un-upper1 Un-upper2 Int-lower1 Int-lower2

$\langle ML \rangle$

end

5 Fixedpt: Least and Greatest Fixed Points; the Knaster-Tarski Theorem

theory *Fixedpt* **imports** *equalities* **begin**

definition

bnd-mono :: $[i, i \Rightarrow i] \Rightarrow o$ **where**
bnd-mono(*D*, *h*) == $h(D) \leq D \ \& \ (\text{ALL } W \ X. W \leq X \longrightarrow X \leq D \longrightarrow h(W) \leq h(X))$

definition

lfp :: $[i, i \Rightarrow i] \Rightarrow i$ **where**
lfp(*D*, *h*) == $\text{Inter}(\{X: \text{Pow}(D). h(X) \leq X\})$

definition

gfp :: $[i, i \Rightarrow i] \Rightarrow i$ **where**
gfp(*D*, *h*) == $\text{Union}(\{X: \text{Pow}(D). X \leq h(X)\})$

The theorem is proved in the lattice of subsets of *D*, namely *Pow*(*D*), with *Inter* as the greatest lower bound.

5.1 Monotone Operators

lemma *bnd-monoI*:

$[\mid h(D) \leq D;$
 $\quad \text{!! } W \ X. [\mid W \leq D; X \leq D; W \leq X] \implies h(W) \leq h(X)$
 $\quad \mid] \implies \text{bnd-mono}(D, h)$
 $\langle \text{proof} \rangle$

lemma *bnd-monoD1*: $\text{bnd-mono}(D, h) \implies h(D) \leq D$

$\langle proof \rangle$

lemma *bnd-monoD2*: $[\mid bnd\text{-}mono(D,h); \quad W \leq X; \quad X \leq D \mid] \implies h(W) \leq h(X)$
 $\langle proof \rangle$

lemma *bnd-mono-subset*:
 $[\mid bnd\text{-}mono(D,h); \quad X \leq D \mid] \implies h(X) \leq D$
 $\langle proof \rangle$

lemma *bnd-mono-Un*:
 $[\mid bnd\text{-}mono(D,h); \quad A \leq D; \quad B \leq D \mid] \implies h(A) \text{ Un } h(B) \leq h(A \text{ Un } B)$
 $\langle proof \rangle$

lemma *bnd-mono-UN*:
 $[\mid bnd\text{-}mono(D,h); \quad \forall i \in I. A(i) \leq D \mid]$
 $\implies (\bigcup i \in I. h(A(i))) \leq h(\bigcup i \in I. A(i))$
 $\langle proof \rangle$

lemma *bnd-mono-Int*:
 $[\mid bnd\text{-}mono(D,h); \quad A \leq D; \quad B \leq D \mid] \implies h(A \text{ Int } B) \leq h(A) \text{ Int } h(B)$
 $\langle proof \rangle$

5.2 Proof of Knaster-Tarski Theorem using *lfp*

lemma *lfp-lowerbound*:
 $[\mid h(D) \leq A; \quad A \leq D \mid] \implies lfp(D,h) \leq A$
 $\langle proof \rangle$

lemma *lfp-subset*: $lfp(D,h) \leq D$
 $\langle proof \rangle$

lemma *def-lfp-subset*: $A == lfp(D,h) \implies A \leq D$
 $\langle proof \rangle$

lemma *lfp-greatest*:
 $[\mid h(D) \leq D; \quad !!X. [\mid h(X) \leq X; \quad X \leq D \mid] \implies A \leq X \mid] \implies A \leq lfp(D,h)$
 $\langle proof \rangle$

lemma *lfp-lemma1*:
 $[\mid bnd\text{-}mono(D,h); \quad h(A) \leq A; \quad A \leq D \mid] \implies h(lfp(D,h)) \leq A$
 $\langle proof \rangle$

lemma *lfp-lemma2*: $bnd\text{-}mono(D,h) \implies h(lfp(D,h)) \leq lfp(D,h)$

$\langle proof \rangle$

lemma *lfp-lemma3*:

$bnd\text{-}mono(D, h) ==> lfp(D, h) \leq h(lfp(D, h))$

$\langle proof \rangle$

lemma *lfp-unfold*: $bnd\text{-}mono(D, h) ==> lfp(D, h) = h(lfp(D, h))$

$\langle proof \rangle$

lemma *def-lfp-unfold*:

$[| A == lfp(D, h); \ bnd\text{-}mono(D, h) |] ==> A = h(A)$

$\langle proof \rangle$

5.3 General Induction Rule for Least Fixedpoints

lemma *Collect-is-pre-fixedpt*:

$[| \ bnd\text{-}mono(D, h); \ !!x. x : h(Collect(lfp(D, h), P)) ==> P(x) \ |]$
 $==> h(Collect(lfp(D, h), P)) \leq Collect(lfp(D, h), P)$

$\langle proof \rangle$

lemma *induct*:

$[| \ bnd\text{-}mono(D, h); \ a : lfp(D, h);$
 $\quad !!x. x : h(Collect(lfp(D, h), P)) ==> P(x)$
 $\quad |] ==> P(a)$

$\langle proof \rangle$

lemma *def-induct*:

$[| \ A == lfp(D, h); \ bnd\text{-}mono(D, h); \ a:A;$
 $\quad !!x. x : h(Collect(A, P)) ==> P(x)$
 $\quad |] ==> P(a)$

$\langle proof \rangle$

lemma *lfp-Int-lowerbound*:

$[| \ h(D \text{ Int } A) \leq A; \ bnd\text{-}mono(D, h) \ |] ==> lfp(D, h) \leq A$

$\langle proof \rangle$

lemma *lfp-mono*:

assumes *hmono*: $bnd\text{-}mono(D, h)$

and *imono*: $bnd\text{-}mono(E, i)$

and *subhi*: $!!X. X \leq D ==> h(X) \leq i(X)$

shows $lfp(D, h) \leq lfp(E, i)$

$\langle proof \rangle$

lemma *lfp-mono2*:

$\llbracket i(D) \leq D; \forall X. X \leq D \implies h(X) \leq i(X) \rrbracket \implies \text{lfp}(D, h) \leq \text{lfp}(D, i)$
 $\langle \text{proof} \rangle$

lemma *lfp-cong*:

$\llbracket D = D'; \forall X. X \leq D' \implies h(X) = h'(X) \rrbracket \implies \text{lfp}(D, h) = \text{lfp}(D', h')$
 $\langle \text{proof} \rangle$

5.4 Proof of Knaster-Tarski Theorem using *gfp*

lemma *gfp-upperbound*: $\llbracket A \leq h(A); A \leq D \rrbracket \implies A \leq \text{gfp}(D, h)$
 $\langle \text{proof} \rangle$

lemma *gfp-subset*: $\text{gfp}(D, h) \leq D$
 $\langle \text{proof} \rangle$

lemma *def-gfp-subset*: $A = \text{gfp}(D, h) \implies A \leq D$
 $\langle \text{proof} \rangle$

lemma *gfp-least*:

$\llbracket \text{bnd-mono}(D, h); \forall X. \llbracket X \leq h(X); X \leq D \rrbracket \implies X \leq A \rrbracket \implies$
 $\text{gfp}(D, h) \leq A$
 $\langle \text{proof} \rangle$

lemma *gfp-lemma1*:

$\llbracket \text{bnd-mono}(D, h); A \leq h(A); A \leq D \rrbracket \implies A \leq h(\text{gfp}(D, h))$
 $\langle \text{proof} \rangle$

lemma *gfp-lemma2*: $\text{bnd-mono}(D, h) \implies \text{gfp}(D, h) \leq h(\text{gfp}(D, h))$
 $\langle \text{proof} \rangle$

lemma *gfp-lemma3*:

$\text{bnd-mono}(D, h) \implies h(\text{gfp}(D, h)) \leq \text{gfp}(D, h)$
 $\langle \text{proof} \rangle$

lemma *gfp-unfold*: $\text{bnd-mono}(D, h) \implies \text{gfp}(D, h) = h(\text{gfp}(D, h))$
 $\langle \text{proof} \rangle$

lemma *def-gfp-unfold*:

$\llbracket A = \text{gfp}(D, h); \text{bnd-mono}(D, h) \rrbracket \implies A = h(A)$
 $\langle \text{proof} \rangle$

5.5 Coinduction Rules for Greatest Fixed Points

lemma *weak-coinduct*: $\llbracket a : X; X \leq h(X); X \leq D \rrbracket \implies a : \text{gfp}(D, h)$
 $\langle \text{proof} \rangle$

lemma *coinduct-lemma*:

$$\begin{aligned} & \llbracket X \leq h(X \text{ Un } \text{gfp}(D, h)); X \leq D; \text{bnd-mono}(D, h) \rrbracket \implies \\ & X \text{ Un } \text{gfp}(D, h) \leq h(X \text{ Un } \text{gfp}(D, h)) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *coinduct*:

$$\begin{aligned} & \llbracket \text{bnd-mono}(D, h); a : X; X \leq h(X \text{ Un } \text{gfp}(D, h)); X \leq D \rrbracket \\ & \implies a : \text{gfp}(D, h) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *def-coinduct*:

$$\begin{aligned} & \llbracket A == \text{gfp}(D, h); \text{bnd-mono}(D, h); a : X; X \leq h(X \text{ Un } A); X \leq D \rrbracket \\ & \implies \\ & a : A \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *def-Collect-coinduct*:

$$\begin{aligned} & \llbracket A == \text{gfp}(D, \%w. \text{Collect}(D, P(w))); \text{bnd-mono}(D, \%w. \text{Collect}(D, P(w))); \\ & a : X; X \leq D; !!z. z : X \implies P(X \text{ Un } A, z) \rrbracket \implies \\ & a : A \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *gfp-mono*:

$$\begin{aligned} & \llbracket \text{bnd-mono}(D, h); D \leq E; \\ & !!X. X \leq D \implies h(X) \leq i(X) \rrbracket \implies \text{gfp}(D, h) \leq \text{gfp}(E, i) \\ & \langle \text{proof} \rangle \end{aligned}$$

end

6 Bool: Booleans in Zermelo-Fraenkel Set Theory

theory *Bool* **imports** *pair* **begin**

abbreviation

$$\begin{aligned} & \text{one} \ (1) \ \mathbf{where} \\ & 1 == \text{succ}(0) \end{aligned}$$

abbreviation

$$\begin{aligned} & \text{two} \ (2) \ \mathbf{where} \\ & 2 == \text{succ}(1) \end{aligned}$$

2 is equal to bool, but is used as a number rather than a type.

definition *bool* == $\{0, 1\}$

definition $\text{cond}(b,c,d) == \text{if}(b=1,c,d)$

definition $\text{not}(b) == \text{cond}(b,0,1)$

definition

$\text{and} \quad :: [i,i] ==> i \quad (\text{infixl and } 70) \quad \text{where}$
 $a \text{ and } b == \text{cond}(a,b,0)$

definition

$\text{or} \quad :: [i,i] ==> i \quad (\text{infixl or } 65) \quad \text{where}$
 $a \text{ or } b == \text{cond}(a,1,b)$

definition

$\text{xor} \quad :: [i,i] ==> i \quad (\text{infixl xor } 65) \quad \text{where}$
 $a \text{ xor } b == \text{cond}(a,\text{not}(b),b)$

lemmas $\text{bool-defs} = \text{bool-def cond-def}$

lemma $\text{singleton-0}: \{0\} = 1$
 $\langle \text{proof} \rangle$

lemma $\text{bool-1I} \text{ [simp,TC]}: 1 : \text{bool}$
 $\langle \text{proof} \rangle$

lemma $\text{bool-0I} \text{ [simp,TC]}: 0 : \text{bool}$
 $\langle \text{proof} \rangle$

lemma $\text{one-not-0}: 1 \sim 0$
 $\langle \text{proof} \rangle$

lemmas $\text{one-neq-0} = \text{one-not-0} \text{ [THEN notE, standard]}$

lemma boolE :
 $\llbracket c : \text{bool}; \quad c=1 ==> P; \quad c=0 ==> P \rrbracket ==> P$
 $\langle \text{proof} \rangle$

lemma $\text{cond-1} \text{ [simp]}: \text{cond}(1,c,d) = c$
 $\langle \text{proof} \rangle$

lemma $\text{cond-0} \text{ [simp]}: \text{cond}(0,c,d) = d$
 $\langle \text{proof} \rangle$

lemma *cond-type* [TC]: [| *b*: bool; *c*: A(1); *d*: A(0) |] ==> *cond*(*b*,*c*,*d*): A(*b*)
 <proof>

lemma *cond-simple-type*: [| *b*: bool; *c*: A; *d*: A |] ==> *cond*(*b*,*c*,*d*): A
 <proof>

lemma *def-cond-1*: [| !!*b*. *j*(*b*)==cond(*b*,*c*,*d*) |] ==> *j*(1) = *c*
 <proof>

lemma *def-cond-0*: [| !!*b*. *j*(*b*)==cond(*b*,*c*,*d*) |] ==> *j*(0) = *d*
 <proof>

lemmas *not-1* = *not-def* [THEN *def-cond-1*, *standard*, *simp*]
lemmas *not-0* = *not-def* [THEN *def-cond-0*, *standard*, *simp*]

lemmas *and-1* = *and-def* [THEN *def-cond-1*, *standard*, *simp*]
lemmas *and-0* = *and-def* [THEN *def-cond-0*, *standard*, *simp*]

lemmas *or-1* = *or-def* [THEN *def-cond-1*, *standard*, *simp*]
lemmas *or-0* = *or-def* [THEN *def-cond-0*, *standard*, *simp*]

lemmas *xor-1* = *xor-def* [THEN *def-cond-1*, *standard*, *simp*]
lemmas *xor-0* = *xor-def* [THEN *def-cond-0*, *standard*, *simp*]

lemma *not-type* [TC]: *a*:bool ==> *not*(*a*) : bool
 <proof>

lemma *and-type* [TC]: [| *a*:bool; *b*:bool |] ==> *a and b* : bool
 <proof>

lemma *or-type* [TC]: [| *a*:bool; *b*:bool |] ==> *a or b* : bool
 <proof>

lemma *xor-type* [TC]: [| *a*:bool; *b*:bool |] ==> *a xor b* : bool
 <proof>

lemmas *bool-typechecks* = *bool-1I bool-0I cond-type not-type and-type*
or-type xor-type

6.1 Laws About 'not'

lemma *not-not* [*simp*]: *a*:bool ==> *not*(*not*(*a*)) = *a*
 <proof>

lemma *not-and* [*simp*]: *a*:bool ==> *not*(*a and b*) = *not*(*a*) *or not*(*b*)
 <proof>

lemma *not-or* [simp]: $a:\text{bool} \implies \text{not}(a \text{ or } b) = \text{not}(a) \text{ and } \text{not}(b)$
 $\langle \text{proof} \rangle$

6.2 Laws About 'and'

lemma *and-absorb* [simp]: $a:\text{bool} \implies a \text{ and } a = a$
 $\langle \text{proof} \rangle$

lemma *and-commute*: $[| a:\text{bool}; b:\text{bool} |] \implies a \text{ and } b = b \text{ and } a$
 $\langle \text{proof} \rangle$

lemma *and-assoc*: $a:\text{bool} \implies (a \text{ and } b) \text{ and } c = a \text{ and } (b \text{ and } c)$
 $\langle \text{proof} \rangle$

lemma *and-or-distrib*: $[| a:\text{bool}; b:\text{bool}; c:\text{bool} |] \implies$
 $(a \text{ or } b) \text{ and } c = (a \text{ and } c) \text{ or } (b \text{ and } c)$
 $\langle \text{proof} \rangle$

6.3 Laws About 'or'

lemma *or-absorb* [simp]: $a:\text{bool} \implies a \text{ or } a = a$
 $\langle \text{proof} \rangle$

lemma *or-commute*: $[| a:\text{bool}; b:\text{bool} |] \implies a \text{ or } b = b \text{ or } a$
 $\langle \text{proof} \rangle$

lemma *or-assoc*: $a:\text{bool} \implies (a \text{ or } b) \text{ or } c = a \text{ or } (b \text{ or } c)$
 $\langle \text{proof} \rangle$

lemma *or-and-distrib*: $[| a:\text{bool}; b:\text{bool}; c:\text{bool} |] \implies$
 $(a \text{ and } b) \text{ or } c = (a \text{ or } c) \text{ and } (b \text{ or } c)$
 $\langle \text{proof} \rangle$

definition

$\text{bool-of-o} :: o \Rightarrow i$ **where**
 $\text{bool-of-o}(P) == (\text{if } P \text{ then } 1 \text{ else } 0)$

lemma [simp]: $\text{bool-of-o}(\text{True}) = 1$
 $\langle \text{proof} \rangle$

lemma [simp]: $\text{bool-of-o}(\text{False}) = 0$
 $\langle \text{proof} \rangle$

lemma [simp,TC]: $\text{bool-of-o}(P) \in \text{bool}$
 $\langle \text{proof} \rangle$

lemma [simp]: $(\text{bool-of-o}(P) = 1) <-> P$
 $\langle \text{proof} \rangle$

lemma *[simp]*: $(\text{bool-of-o}(P) = 0) <-> \sim P$
 $\langle \text{proof} \rangle$

$\langle ML \rangle$

end

7 Sum: Disjoint Sums

theory *Sum* **imports** *Bool equalities* **begin**

And the "Part" primitive for simultaneous recursive type definitions

global

definition *sum* :: $[i, i] \Rightarrow i$ (**infixr** + 65) **where**
 $A + B == \{0\} * A \text{ Un } \{1\} * B$

definition *Inl* :: $i \Rightarrow i$ **where**
 $\text{Inl}(a) == <0, a>$

definition *Inr* :: $i \Rightarrow i$ **where**
 $\text{Inr}(b) == <1, b>$

definition *case* :: $[i \Rightarrow i, i \Rightarrow i, i] \Rightarrow i$ **where**
 $\text{case}(c, d) == (\%<y, z>. \text{cond}(y, d(z), c(z)))$

definition *Part* :: $[i, i \Rightarrow i] \Rightarrow i$ **where**
 $\text{Part}(A, h) == \{x: A. \text{EX } z. x = h(z)\}$

local

7.1 Rules for the *Part* Primitive

lemma *Part-iff*:
 $a : \text{Part}(A, h) <-> a : A \ \& \ (\text{EX } y. a = h(y))$
 $\langle \text{proof} \rangle$

lemma *Part-eqI* [*intro*]:
 $[\mid a : A; a = h(b) \mid] ==> a : \text{Part}(A, h)$
 $\langle \text{proof} \rangle$

lemmas *PartI* = *refl* [*THEN* [2] *Part-eqI*]

lemma *PartE* [*elim!*]:
 $[\mid a : \text{Part}(A, h); !!z. [\mid a : A; a = h(z) \mid] ==> P \mid] ==> P$
 $\langle \text{proof} \rangle$

lemma *Part-subset*: $Part(A, h) \leq A$
 $\langle proof \rangle$

7.2 Rules for Disjoint Sums

lemmas *sum-defs* = *sum-def Inl-def Inr-def case-def*

lemma *Sigma-bool*: $Sigma(bool, C) = C(0) + C(1)$
 $\langle proof \rangle$

lemma *InlI* [*intro!*, *simp*, *TC*]: $a : A \implies Inl(a) : A+B$
 $\langle proof \rangle$

lemma *InrI* [*intro!*, *simp*, *TC*]: $b : B \implies Inr(b) : A+B$
 $\langle proof \rangle$

lemma *sumE* [*elim!*]:

$$\begin{aligned} & [[u : A+B; \\ & \quad !!x. [[x:A; \quad u=Inl(x)]] \implies P; \\ & \quad !!y. [[y:B; \quad u=Inr(y)]] \implies P \\ &]] \implies P \end{aligned}$$

 $\langle proof \rangle$

lemma *Inl-iff* [*iff*]: $Inl(a)=Inl(b) \iff a=b$
 $\langle proof \rangle$

lemma *Inr-iff* [*iff*]: $Inr(a)=Inr(b) \iff a=b$
 $\langle proof \rangle$

lemma *Inl-Inr-iff* [*simp*]: $Inl(a)=Inr(b) \iff False$
 $\langle proof \rangle$

lemma *Inr-Inl-iff* [*simp*]: $Inr(b)=Inl(a) \iff False$
 $\langle proof \rangle$

lemma *sum-empty* [*simp*]: $0+0 = 0$
 $\langle proof \rangle$

lemmas *Inl-inject* = *Inl-iff* [*THEN iffD1, standard*]
lemmas *Inr-inject* = *Inr-iff* [*THEN iffD1, standard*]

lemmas $Inl-neq-Inr = Inl-Inr-iff$ $[THEN\ iffD1,\ THEN\ FalseE,\ elim!]$
lemmas $Inr-neq-Inl = Inr-Inl-iff$ $[THEN\ iffD1,\ THEN\ FalseE,\ elim!]$

lemma $InlD$: $Inl(a): A+B ==> a: A$
 $\langle proof \rangle$

lemma $InrD$: $Inr(b): A+B ==> b: B$
 $\langle proof \rangle$

lemma $sum-iff$: $u: A+B <-> (EX\ x.\ x:A \ \&\ u=Inl(x)) \mid (EX\ y.\ y:B \ \&\ u=Inr(y))$
 $\langle proof \rangle$

lemma $Inl-in-sum-iff$ $[simp]$: $(Inl(x) \in A+B) <-> (x \in A)$
 $\langle proof \rangle$

lemma $Inr-in-sum-iff$ $[simp]$: $(Inr(y) \in A+B) <-> (y \in B)$
 $\langle proof \rangle$

lemma $sum-subset-iff$: $A+B <= C+D <-> A<=C \ \&\ B<=D$
 $\langle proof \rangle$

lemma $sum-equal-iff$: $A+B = C+D <-> A=C \ \&\ B=D$
 $\langle proof \rangle$

lemma $sum-eq-2-times$: $A+A = 2*A$
 $\langle proof \rangle$

7.3 The Eliminator: *case*

lemma $case-Inl$ $[simp]$: $case(c, d, Inl(a)) = c(a)$
 $\langle proof \rangle$

lemma $case-Inr$ $[simp]$: $case(c, d, Inr(b)) = d(b)$
 $\langle proof \rangle$

lemma $case-type$ $[TC]$:

$$\begin{aligned} &[[\ u: A+B; \\ &\quad !!x.\ x: A ==> c(x): C(Inl(x)); \\ &\quad !!y.\ y: B ==> d(y): C(Inr(y)) \\ &]] ==> case(c,d,u) : C(u) \end{aligned}$$
 $\langle proof \rangle$

lemma $expand-case$: $u: A+B ==>$

$$\begin{aligned} &R(case(c,d,u)) <-> \\ &((ALL\ x:A.\ u = Inl(x) --> R(c(x))) \ \& \\ &(ALL\ y:B.\ u = Inr(y) --> R(d(y)))) \end{aligned}$$
 $\langle proof \rangle$

lemma *case-cong*:

```

  [| z: A+B;
    !!x. x:A ==> c(x)=c'(x);
    !!y. y:B ==> d(y)=d'(y)
  |] ==> case(c,d,z) = case(c',d',z)
<proof>

```

lemma *case-case*: $z: A+B \implies$

```

  case(c, d, case(%x. Inl(c'(x)), %y. Inr(d'(y)), z)) =
  case(%x. c(c'(x)), %y. d(d'(y)), z)
<proof>

```

7.4 More Rules for $Part(A, h)$

lemma *Part-mono*: $A \leq B \implies Part(A, h) \leq Part(B, h)$

<proof>

lemma *Part-Collect*: $Part(Collect(A, P), h) = Collect(Part(A, h), P)$

<proof>

lemmas *Part-CollectE* =

Part-Collect [THEN equalityD1, THEN subsetD, THEN CollectE, standard]

lemma *Part-Inl*: $Part(A+B, Inl) = \{Inl(x). x: A\}$

<proof>

lemma *Part-Inr*: $Part(A+B, Inr) = \{Inr(y). y: B\}$

<proof>

lemma *PartD1*: $a : Part(A, h) \implies a : A$

<proof>

lemma *Part-id*: $Part(A, \%x. x) = A$

<proof>

lemma *Part-Inr2*: $Part(A+B, \%x. Inr(h(x))) = \{Inr(y). y: Part(B, h)\}$

<proof>

lemma *Part-sum-equality*: $C \leq A+B \implies Part(C, Inl) \cup Part(C, Inr) = C$

<proof>

end

8 func: Functions, Function Spaces, Lambda-Abstraction

theory *func* **imports** *equalities Sum* **begin**

8.1 The Pi Operator: Dependent Function Space

lemma *subset-Sigma-imp-relation*: $r \leq \text{Sigma}(A,B) \implies \text{relation}(r)$
 $\langle \text{proof} \rangle$

lemma *relation-converse-converse* [simp]:
 $\text{relation}(r) \implies \text{converse}(\text{converse}(r)) = r$
 $\langle \text{proof} \rangle$

lemma *relation-restrict* [simp]: $\text{relation}(\text{restrict}(r,A))$
 $\langle \text{proof} \rangle$

lemma *Pi-iff*:
 $f: \text{Pi}(A,B) \iff \text{function}(f) \ \& \ f \leq \text{Sigma}(A,B) \ \& \ A \leq \text{domain}(f)$
 $\langle \text{proof} \rangle$

lemma *Pi-iff-old*:
 $f: \text{Pi}(A,B) \iff f \leq \text{Sigma}(A,B) \ \& \ (\text{ALL } x:A. \text{EX! } y. \langle x,y \rangle : f)$
 $\langle \text{proof} \rangle$

lemma *fun-is-function*: $f: \text{Pi}(A,B) \implies \text{function}(f)$
 $\langle \text{proof} \rangle$

lemma *function-imp-Pi*:
 $[\text{function}(f); \text{relation}(f)] \implies f \in \text{domain}(f) \rightarrow \text{range}(f)$
 $\langle \text{proof} \rangle$

lemma *functionI*:
 $[\text{!!}x \ y \ y'. [\langle x,y \rangle : r; \langle x,y' \rangle : r] \implies y=y'] \implies \text{function}(r)$
 $\langle \text{proof} \rangle$

lemma *fun-is-rel*: $f: \text{Pi}(A,B) \implies f \leq \text{Sigma}(A,B)$
 $\langle \text{proof} \rangle$

lemma *Pi-cong*:
 $[\text{A}=\text{A}'; \text{!!}x. x:\text{A}' \implies B(x)=B'(x)] \implies \text{Pi}(A,B) = \text{Pi}(A',B')$
 $\langle \text{proof} \rangle$

lemma *fun-weaken-type*: $[\text{f}: \text{A} \rightarrow \text{B}; \text{B} \leq \text{D}] \implies \text{f}: \text{A} \rightarrow \text{D}$
 $\langle \text{proof} \rangle$

8.2 Function Application

lemma *apply-equality2*: $[\langle a,b \rangle : f; \langle a,c \rangle : f; f: \text{Pi}(A,B)] \implies b=c$
 $\langle \text{proof} \rangle$

lemma *function-apply-equality*: $[[\langle a, b \rangle : f; \text{function}(f)]] \implies f'a = b$
 $\langle \text{proof} \rangle$

lemma *apply-equality*: $[[\langle a, b \rangle : f; f : \text{Pi}(A, B)]] \implies f'a = b$
 $\langle \text{proof} \rangle$

lemma *apply-0*: $a \sim : \text{domain}(f) \implies f'a = 0$
 $\langle \text{proof} \rangle$

lemma *Pi-memberD*: $[[f : \text{Pi}(A, B); c : f]] \implies \text{EX } x : A. c = \langle x, f'x \rangle$
 $\langle \text{proof} \rangle$

lemma *function-apply-Pair*: $[[\text{function}(f); a : \text{domain}(f)]] \implies \langle a, f'a \rangle : f$
 $\langle \text{proof} \rangle$

lemma *apply-Pair*: $[[f : \text{Pi}(A, B); a : A]] \implies \langle a, f'a \rangle : f$
 $\langle \text{proof} \rangle$

lemma *apply-type* [TC]: $[[f : \text{Pi}(A, B); a : A]] \implies f'a : B(a)$
 $\langle \text{proof} \rangle$

lemma *apply-funtype*: $[[f : A \multimap B; a : A]] \implies f'a : B$
 $\langle \text{proof} \rangle$

lemma *apply-iff*: $f : \text{Pi}(A, B) \implies \langle a, b \rangle : f \iff a : A \ \& \ f'a = b$
 $\langle \text{proof} \rangle$

lemma *Pi-type*: $[[f : \text{Pi}(A, C); !!x. x : A \implies f'x : B(x)]] \implies f : \text{Pi}(A, B)$
 $\langle \text{proof} \rangle$

lemma *Pi-Collect-iff*:
 $(f : \text{Pi}(A, \%x. \{y : B(x). P(x, y)\}))$
 $\iff f : \text{Pi}(A, B) \ \& \ (\text{ALL } x : A. P(x, f'x))$
 $\langle \text{proof} \rangle$

lemma *Pi-weaken-type*:
 $[[f : \text{Pi}(A, B); !!x. x : A \implies B(x) \leq C(x)]] \implies f : \text{Pi}(A, C)$
 $\langle \text{proof} \rangle$

lemma *domain-type*: $[[\langle a, b \rangle : f; f : \text{Pi}(A, B)]] \implies a : A$

$\langle proof \rangle$

lemma *range-type*: $[| <a,b> : f; f : Pi(A,B) |] ==> b : B(a)$
 $\langle proof \rangle$

lemma *Pair-mem-PiD*: $[| <a,b> : f; f : Pi(A,B) |] ==> a:A \& b:B(a) \& f'a = b$
 $\langle proof \rangle$

8.3 Lambda Abstraction

lemma *lamI*: $a:A ==> <a,b(a)> : (lam\ x:A. b(x))$
 $\langle proof \rangle$

lemma *lamE*:
 $[| p : (lam\ x:A. b(x)); !!x. [x:A; p=<x,b(x)>] |] ==> P$
 $\langle proof \rangle$

lemma *lamD*: $[| <a,c> : (lam\ x:A. b(x)) |] ==> c = b(a)$
 $\langle proof \rangle$

lemma *lam-type [TC]*:
 $[| !!x. x:A ==> b(x) : B(x) |] ==> (lam\ x:A. b(x)) : Pi(A,B)$
 $\langle proof \rangle$

lemma *lam-funtype*: $(lam\ x:A. b(x)) : A -> \{b(x). x:A\}$
 $\langle proof \rangle$

lemma *function-lam*: *function* $(lam\ x:A. b(x))$
 $\langle proof \rangle$

lemma *relation-lam*: *relation* $(lam\ x:A. b(x))$
 $\langle proof \rangle$

lemma *beta-if [simp]*: $(lam\ x:A. b(x)) \text{ ' } a = (if\ a : A\ then\ b(a)\ else\ 0)$
 $\langle proof \rangle$

lemma *beta*: $a : A ==> (lam\ x:A. b(x)) \text{ ' } a = b(a)$
 $\langle proof \rangle$

lemma *lam-empty [simp]*: $(lam\ x:0. b(x)) = 0$
 $\langle proof \rangle$

lemma *domain-lam [simp]*: $domain(Lambda(A,b)) = A$
 $\langle proof \rangle$

lemma *lam-cong [cong]*:
 $[| A=A'; !!x. x:A' ==> b(x)=b'(x) |] ==> Lambda(A,b) = Lambda(A',b')$

$\langle proof \rangle$

lemma *lam-theI*:

$(!!x. x:A ==> EX! y. Q(x,y)) ==> EX f. ALL x:A. Q(x, f'x)$

$\langle proof \rangle$

lemma *lam-eqE*: $[(lam x:A. f(x)) = (lam x:A. g(x)); a:A] ==> f(a)=g(a)$

$\langle proof \rangle$

lemma *Pi-empty1* [simp]: $Pi(0,A) = \{0\}$

$\langle proof \rangle$

lemma *singleton-fun* [simp]: $\{<a,b>\} : \{a\} -> \{b\}$

$\langle proof \rangle$

lemma *Pi-empty2* [simp]: $(A->0) = (if A=0 then \{0\} else 0)$

$\langle proof \rangle$

lemma *fun-space-empty-iff* [iff]: $(A->X)=0 \longleftrightarrow X=0 \ \& \ (A \neq 0)$

$\langle proof \rangle$

8.4 Extensionality

lemma *fun-subset*:

$[[f : Pi(A,B); g : Pi(C,D); A<=C; !!x. x:A ==> f'x = g'x]] ==> f<=g$

$\langle proof \rangle$

lemma *fun-extension*:

$[[f : Pi(A,B); g : Pi(A,D); !!x. x:A ==> f'x = g'x]] ==> f=g$

$\langle proof \rangle$

lemma *eta* [simp]: $f : Pi(A,B) ==> (lam x:A. f'x) = f$

$\langle proof \rangle$

lemma *fun-extension-iff*:

$[[f:Pi(A,B); g:Pi(A,C)]] ==> (ALL a:A. f'a = g'a) <-> f=g$

$\langle proof \rangle$

lemma *fun-subset-eq*: $[[f:Pi(A,B); g:Pi(A,C)]] ==> f <= g <-> (f = g)$

$\langle proof \rangle$

lemma *Pi-lamE*:
 assumes *major*: $f: Pi(A,B)$
 and *minor*: $!!b. [| ALL x:A. b(x):B(x); f = (lam x:A. b(x)) |] ==> P$
 shows P
 $\langle proof \rangle$

8.5 Images of Functions

lemma *image-lam*: $C \leq A ==> (lam x:A. b(x)) \text{ `` } C = \{b(x). x:C\}$
 $\langle proof \rangle$

lemma *Repfun-function-if*:
 $function(f)$
 $==> \{f'x. x:C\} = (if C \leq domain(f) then f''C else cons(0,f''C))$
 $\langle proof \rangle$

lemma *image-function*:
 $[| function(f); C \leq domain(f) |] ==> f''C = \{f'x. x:C\}$
 $\langle proof \rangle$

lemma *image-fun*: $[| f : Pi(A,B); C \leq A |] ==> f''C = \{f'x. x:C\}$
 $\langle proof \rangle$

lemma *image-eq-UN*:
 assumes $f: f \in Pi(A,B)$ $C \subseteq A$ shows $f''C = (\bigcup x \in C. \{f'x\})$
 $\langle proof \rangle$

lemma *Pi-image-cons*:
 $[| f: Pi(A,B); x: A |] ==> f \text{ `` } cons(x,y) = cons(f'x, f''y)$
 $\langle proof \rangle$

8.6 Properties of $restrict(f, A)$

lemma *restrict-subset*: $restrict(f,A) \leq f$
 $\langle proof \rangle$

lemma *function-restrictI*:
 $function(f) ==> function(restrict(f,A))$
 $\langle proof \rangle$

lemma *restrict-type2*: $[| f: Pi(C,B); A \leq C |] ==> restrict(f,A) : Pi(A,B)$
 $\langle proof \rangle$

lemma *restrict*: $restrict(f,A) \text{ `` } a = (if a : A then f'a else 0)$
 $\langle proof \rangle$

lemma *restrict-empty [simp]*: $restrict(f,0) = 0$
 $\langle proof \rangle$

lemma *restrict-iff*: $z \in \text{restrict}(r, A) \longleftrightarrow z \in r \ \& \ (\exists x \in A. \exists y. z = \langle x, y \rangle)$
 $\langle \text{proof} \rangle$

lemma *restrict-restrict* [simp]:
 $\text{restrict}(\text{restrict}(r, A), B) = \text{restrict}(r, A \text{ Int } B)$
 $\langle \text{proof} \rangle$

lemma *domain-restrict* [simp]: $\text{domain}(\text{restrict}(f, C)) = \text{domain}(f) \text{ Int } C$
 $\langle \text{proof} \rangle$

lemma *restrict-idem*: $f \leq \text{Sigma}(A, B) \implies \text{restrict}(f, A) = f$
 $\langle \text{proof} \rangle$

lemma *domain-restrict-idem*:
 $[\text{domain}(r) \leq A; \text{relation}(r)] \implies \text{restrict}(r, A) = r$
 $\langle \text{proof} \rangle$

lemma *domain-restrict-lam* [simp]: $\text{domain}(\text{restrict}(\text{Lambda}(A, f), C)) = A \text{ Int } C$
 $\langle \text{proof} \rangle$

lemma *restrict-if* [simp]: $\text{restrict}(f, A) \text{ ` } a = (\text{if } a : A \text{ then } f \text{ ` } a \text{ else } 0)$
 $\langle \text{proof} \rangle$

lemma *restrict-lam-eq*:
 $A \leq C \implies \text{restrict}(\text{lam } x:C. b(x), A) = (\text{lam } x:A. b(x))$
 $\langle \text{proof} \rangle$

lemma *fun-cons-restrict-eq*:
 $f : \text{cons}(a, b) \rightarrow B \implies f = \text{cons}(\text{ ` } a \text{ ` }, \text{restrict}(f, b))$
 $\langle \text{proof} \rangle$

8.7 Unions of Functions

lemma *function-Union*:
 $[\text{ALL } x:S. \text{function}(x);$
 $\text{ALL } x:S. \text{ALL } y:S. x \leq y \mid y \leq x]$
 $\implies \text{function}(\text{Union}(S))$
 $\langle \text{proof} \rangle$

lemma *fun-Union*:
 $[\text{ALL } f:S. \text{EX } C D. f:C \rightarrow D;$
 $\text{ALL } f:S. \text{ALL } y:S. f \leq y \mid y \leq f]$ \implies
 $\text{Union}(S) : \text{domain}(\text{Union}(S)) \rightarrow \text{range}(\text{Union}(S))$
 $\langle \text{proof} \rangle$

lemma *gen-relation-Union* [rule-format]:
 $\forall f \in F. \text{relation}(f) \implies \text{relation}(\text{Union}(F))$

$\langle proof \rangle$

lemmas $Un\text{-}rls = Un\text{-}subset\text{-}iff\ SUM\text{-}Un\text{-}distrib1\ prod\text{-}Un\text{-}distrib2$
 $subset\text{-}trans\ [OF - Un\text{-}upper1]$
 $subset\text{-}trans\ [OF - Un\text{-}upper2]$

lemma $fun\text{-}disjoint\text{-}Un$:
 $[[f: A \rightarrow B; g: C \rightarrow D; A\ Int\ C = 0\]]$
 $\implies (f\ Un\ g) : (A\ Un\ C) \rightarrow (B\ Un\ D)$

$\langle proof \rangle$

lemma $fun\text{-}disjoint\text{-}apply1$: $a \notin domain(g) \implies (f\ Un\ g)'a = f'a$
 $\langle proof \rangle$

lemma $fun\text{-}disjoint\text{-}apply2$: $c \notin domain(f) \implies (f\ Un\ g)'c = g'c$
 $\langle proof \rangle$

8.8 Domain and Range of a Function or Relation

lemma $domain\text{-}of\text{-}fun$: $f : Pi(A,B) \implies domain(f)=A$
 $\langle proof \rangle$

lemma $apply\text{-}rangeI$: $[[f : Pi(A,B); a: A\]] \implies f'a : range(f)$
 $\langle proof \rangle$

lemma $range\text{-}of\text{-}fun$: $f : Pi(A,B) \implies f : A \rightarrow range(f)$
 $\langle proof \rangle$

8.9 Extensions of Functions

lemma $fun\text{-}extend$:
 $[[f: A \rightarrow B; c \sim: A\]] \implies cons(<c,b>,f) : cons(c,A) \rightarrow cons(b,B)$
 $\langle proof \rangle$

lemma $fun\text{-}extend3$:
 $[[f: A \rightarrow B; c \sim: A; b: B\]] \implies cons(<c,b>,f) : cons(c,A) \rightarrow B$
 $\langle proof \rangle$

lemma $extend\text{-}apply$:
 $c \sim: domain(f) \implies cons(<c,b>,f)'a = (if\ a=c\ then\ b\ else\ f'a)$
 $\langle proof \rangle$

lemma $fun\text{-}extend\text{-}apply\ [simp]$:
 $[[f: A \rightarrow B; c \sim: A\]] \implies cons(<c,b>,f)'a = (if\ a=c\ then\ b\ else\ f'a)$
 $\langle proof \rangle$

lemmas *singleton-apply* = *apply-equality* [*OF singletonI singleton-fun, simp*]

lemma *cons-fun-eq*:

$c \sim: A \implies \text{cons}(c, A) \rightarrow B = (\bigcup f \in A \rightarrow B. \bigcup b \in B. \{\text{cons}(\langle c, b \rangle, f)\})$
 $\langle \text{proof} \rangle$

lemma *succ-fun-eq*: $\text{succ}(n) \rightarrow B = (\bigcup f \in n \rightarrow B. \bigcup b \in B. \{\text{cons}(\langle n, b \rangle, f)\})$
 $\langle \text{proof} \rangle$

8.10 Function Updates

definition

$\text{update} :: [i, i, i] \Rightarrow i$ **where**
 $\text{update}(f, a, b) == \text{lam } x: \text{cons}(a, \text{domain}(f)). \text{if}(x=a, b, f'x)$

nonterminals

updbinds updbind

syntax

$\text{-updbind} :: [i, i] \Rightarrow \text{updbind} \quad ((\text{-} := / \text{-}))$
 $\quad \quad \quad :: \text{updbind} \Rightarrow \text{updbinds} \quad (\text{-})$
 $\text{-updbinds} :: [\text{updbind}, \text{updbinds}] \Rightarrow \text{updbinds} \quad (\text{-} / \text{-})$
 $\text{-Update} :: [i, \text{updbinds}] \Rightarrow i \quad (\text{-}'((\text{-})') [900, 0] 900)$

translations

$\text{-Update } (f, \text{-updbinds}(b, bs)) == \text{-Update } (\text{-Update}(f, b), bs)$
 $f(x:=y) == \text{CONST } \text{update}(f, x, y)$

lemma *update-apply* [*simp*]: $f(x:=y) \text{ ' } z = (\text{if } z=x \text{ then } y \text{ else } f'z)$
 $\langle \text{proof} \rangle$

lemma *update-idem*: $[\text{if } x = y; f: \text{Pi}(A, B); x: A] \implies f(x:=y) = f$
 $\langle \text{proof} \rangle$

declare *refl* [*THEN update-idem, simp*]

lemma *domain-update* [*simp*]: $\text{domain}(f(x:=y)) = \text{cons}(x, \text{domain}(f))$
 $\langle \text{proof} \rangle$

lemma *update-type*: $[\text{if } f: \text{Pi}(A, B); x: A; y: B(x)] \implies f(x:=y) : \text{Pi}(A, B)$
 $\langle \text{proof} \rangle$

8.11 Monotonicity Theorems

8.11.1 Replacement in its Various Forms

lemma *Replace-mono*: $A \leq B \implies \text{Replace}(A, P) \leq \text{Replace}(B, P)$
 $\langle \text{proof} \rangle$

lemma *RepFun-mono*: $A \leq B \implies \{f(x). x:A\} \leq \{f(x). x:B\}$
 $\langle \text{proof} \rangle$

lemma *Pow-mono*: $A \leq B \implies \text{Pow}(A) \leq \text{Pow}(B)$
 $\langle \text{proof} \rangle$

lemma *Union-mono*: $A \leq B \implies \text{Union}(A) \leq \text{Union}(B)$
 $\langle \text{proof} \rangle$

lemma *UN-mono*:
 $[\mid A \leq C; \ !\!x. x:A \implies B(x) \leq D(x) \mid] \implies (\bigcup x \in A. B(x)) \leq (\bigcup x \in C. D(x))$
 $\langle \text{proof} \rangle$

lemma *Inter-anti-mono*: $[\mid A \leq B; \ A \neq 0 \mid] \implies \text{Inter}(B) \leq \text{Inter}(A)$
 $\langle \text{proof} \rangle$

lemma *cons-mono*: $C \leq D \implies \text{cons}(a, C) \leq \text{cons}(a, D)$
 $\langle \text{proof} \rangle$

lemma *Un-mono*: $[\mid A \leq C; \ B \leq D \mid] \implies A \text{ Un } B \leq C \text{ Un } D$
 $\langle \text{proof} \rangle$

lemma *Int-mono*: $[\mid A \leq C; \ B \leq D \mid] \implies A \text{ Int } B \leq C \text{ Int } D$
 $\langle \text{proof} \rangle$

lemma *Diff-mono*: $[\mid A \leq C; \ D \leq B \mid] \implies A - B \leq C - D$
 $\langle \text{proof} \rangle$

8.11.2 Standard Products, Sums and Function Spaces

lemma *Sigma-mono* [rule-format]:
 $[\mid A \leq C; \ !\!x. x:A \multimap B(x) \leq D(x) \mid] \implies \text{Sigma}(A, B) \leq \text{Sigma}(C, D)$
 $\langle \text{proof} \rangle$

lemma *sum-mono*: $[\mid A \leq C; \ B \leq D \mid] \implies A + B \leq C + D$
 $\langle \text{proof} \rangle$

lemma *Pi-mono*: $B \leq C \implies A \multimap B \leq A \multimap C$
 $\langle \text{proof} \rangle$

lemma *lam-mono*: $A \leq B \implies \text{Lambda}(A, c) \leq \text{Lambda}(B, c)$
 $\langle \text{proof} \rangle$

8.11.3 Converse, Domain, Range, Field

lemma *converse-mono*: $r \leq s \implies \text{converse}(r) \leq \text{converse}(s)$
 $\langle \text{proof} \rangle$

lemma *domain-mono*: $r \leq s \implies \text{domain}(r) \leq \text{domain}(s)$
 $\langle \text{proof} \rangle$

lemmas *domain-rel-subset* = *subset-trans* [OF *domain-mono domain-subset*]

lemma *range-mono*: $r \leq s \implies \text{range}(r) \leq \text{range}(s)$
 $\langle \text{proof} \rangle$

lemmas *range-rel-subset* = *subset-trans* [OF *range-mono range-subset*]

lemma *field-mono*: $r \leq s \implies \text{field}(r) \leq \text{field}(s)$
 $\langle \text{proof} \rangle$

lemma *field-rel-subset*: $r \leq A * A \implies \text{field}(r) \leq A$
 $\langle \text{proof} \rangle$

8.11.4 Images

lemma *image-pair-mono*:
 $[[\text{!! } x \ y. \langle x, y \rangle : r \implies \langle x, y \rangle : s; \ A \leq B]] \implies r''A \leq s''B$
 $\langle \text{proof} \rangle$

lemma *vimage-pair-mono*:
 $[[\text{!! } x \ y. \langle x, y \rangle : r \implies \langle x, y \rangle : s; \ A \leq B]] \implies r^{-''}A \leq s^{-''}B$
 $\langle \text{proof} \rangle$

lemma *image-mono*: $[[r \leq s; \ A \leq B]] \implies r''A \leq s''B$
 $\langle \text{proof} \rangle$

lemma *vimage-mono*: $[[r \leq s; \ A \leq B]] \implies r^{-''}A \leq s^{-''}B$
 $\langle \text{proof} \rangle$

lemma *Collect-mono*:
 $[[A \leq B; \ \text{!!}x. x:A \implies P(x) \dashrightarrow Q(x)]] \implies \text{Collect}(A, P) \leq \text{Collect}(B, Q)$
 $\langle \text{proof} \rangle$

lemmas *basic-monos* = *subset-refl imp-refl disj-mono conj-mono ex-mono*
Collect-mono Part-mono in-mono

lemma *bex-image-simp*:

$[| f : Pi(X, Y); A \subseteq X |] ==> (EX x : f^{\prime\prime}A. P(x)) <-> (EX x:A. P(f^{\prime}x))$
 $\langle proof \rangle$

lemma *ball-image-simp*:

$[| f : Pi(X, Y); A \subseteq X |] ==> (ALL x : f^{\prime\prime}A. P(x)) <-> (ALL x:A. P(f^{\prime}x))$
 $\langle proof \rangle$

end

9 QPair: Quine-Inspired Ordered Pairs and Disjoint Sums

theory *QPair* **imports** *Sum func* **begin**

For non-well-founded data structures in ZF. Does not precisely follow Quine's construction. Thanks to Thomas Forster for suggesting this approach!

W. V. Quine, On Ordered Pairs and Relations, in Selected Logic Papers, 1966.

definition

$QPair :: [i, i] ==> i$ $(<(-;/ -)>)$ **where**
 $<a;b> == a+b$

definition

$qfst :: i ==> i$ **where**
 $qfst(p) == THE a. EX b. p=<a;b>$

definition

$qsnd :: i ==> i$ **where**
 $qsnd(p) == THE b. EX a. p=<a;b>$

definition

$qsplit :: [[i, i] ==> 'a, i] ==> 'a::\}$ **where**
 $qsplit(c,p) == c(qfst(p), qsnd(p))$

definition

$qconverse :: i ==> i$ **where**
 $qconverse(r) == \{z. w:r, EX x y. w=<x;y> \ \& \ z=<y;x>\}$

definition

$QSigma :: [i, i ==> i] ==> i$ **where**
 $QSigma(A,B) == \bigcup_{x \in A} \bigcup_{y \in B(x)} \{<x;y>\}$

syntax

$-QSUM :: [idt, i, i] ==> i$ $((\exists QSUM \text{ :-./ -}) 10)$

translations

$QSUM\ x:A. B \Rightarrow CONST\ QSigma(A, \%x. B)$

abbreviation

$qprod\ (\text{infixr } <*> 80)\ \text{where}$
 $A <*> B == QSigma(A, \%-. B)$

definition

$qsum\ :: [i,i]=>i\ (\text{infixr } <+> 65)\ \text{where}$
 $A <+> B == (\{0\} <*> A) \cup (\{1\} <*> B)$

definition

$QInl\ :: i=>i\ \text{where}$
 $QInl(a) == <0;a>$

definition

$QInr\ :: i=>i\ \text{where}$
 $QInr(b) == <1;b>$

definition

$qcase\ :: [i=>i, i=>i, i]=>i\ \text{where}$
 $qcase(c,d) == qsplit(\%y\ z. cond(y, d(z), c(z)))$

9.1 Quine ordered pairing

lemma $QPair\text{-}empty\ [simp]: <0;0> = 0$
 $\langle proof \rangle$

lemma $QPair\text{-}iff\ [simp]: <a;b> = <c;d> \Leftrightarrow a=c \ \& \ b=d$
 $\langle proof \rangle$

lemmas $QPair\text{-}inject = QPair\text{-}iff\ [THEN\ iffD1, THEN\ conjE, standard, elim!]$

lemma $QPair\text{-}inject1: <a;b> = <c;d> \Rightarrow a=c$
 $\langle proof \rangle$

lemma $QPair\text{-}inject2: <a;b> = <c;d> \Rightarrow b=d$
 $\langle proof \rangle$

9.1.1 QSigma: Disjoint union of a family of sets Generalizes Cartesian product

lemma $QSigmaI\ [intro!]: [\ a:A;\ b:B(a)\] \Rightarrow <a;b> : QSigma(A,B)$
 $\langle proof \rangle$

lemma $QSigmaE\ [elim!]:$

$[\ c: QSigma(A,B);$
 $!!x\ y. [\ x:A;\ y:B(x);\ c=<x;y>\] \Rightarrow P$

$\llbracket \rrbracket \implies P$
 $\langle proof \rangle$

lemma *QSigmaE2* [elim!]:
 $\llbracket \langle a; b \rangle : QSigma(A, B); \llbracket a : A; b : B(a) \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *QSigmaD1*: $\langle a; b \rangle : QSigma(A, B) \implies a : A$
 $\langle proof \rangle$

lemma *QSigmaD2*: $\langle a; b \rangle : QSigma(A, B) \implies b : B(a)$
 $\langle proof \rangle$

lemma *QSigma-cong*:
 $\llbracket A = A'; \llbracket !x. x : A' \implies B(x) = B'(x) \rrbracket \implies$
 $QSigma(A, B) = QSigma(A', B')$
 $\langle proof \rangle$

lemma *QSigma-empty1* [simp]: $QSigma(0, B) = 0$
 $\langle proof \rangle$

lemma *QSigma-empty2* [simp]: $A < * \rangle 0 = 0$
 $\langle proof \rangle$

9.1.2 Projections: qfst, qsnd

lemma *qfst-conv* [simp]: $qfst(\langle a; b \rangle) = a$
 $\langle proof \rangle$

lemma *qsnd-conv* [simp]: $qsnd(\langle a; b \rangle) = b$
 $\langle proof \rangle$

lemma *qfst-type* [TC]: $p : QSigma(A, B) \implies qfst(p) : A$
 $\langle proof \rangle$

lemma *qsnd-type* [TC]: $p : QSigma(A, B) \implies qsnd(p) : B(qfst(p))$
 $\langle proof \rangle$

lemma *QPair-qfst-qsnd-eq*: $a : QSigma(A, B) \implies \langle qfst(a); qsnd(a) \rangle = a$
 $\langle proof \rangle$

9.1.3 Eliminator: qsplit

lemma *qsplit* [simp]: $qsplit(\%x y. c(x, y), \langle a; b \rangle) == c(a, b)$
 $\langle proof \rangle$

lemma *qsplit-type* [elim!]:
 $\llbracket p : QSigma(A, B);$
 $\llbracket !x y. \llbracket x : A; y : B(x) \rrbracket \implies c(x, y) : C(\langle x; y \rangle) \rrbracket$

$\llbracket \rrbracket \implies \text{qsplit}(\%x\ y. c(x,y), p) : C(p)$
 $\langle \text{proof} \rangle$

lemma *expand-qsplit*:

$u : A <*> B \implies R(\text{qsplit}(c,u)) <-> (\text{ALL } x:A. \text{ALL } y:B. u = <x;y> \dashv\dashv R(c(x,y)))$
 $\langle \text{proof} \rangle$

9.1.4 qsplit for predicates: result type o

lemma *qsplitI*: $R(a,b) \implies \text{qsplit}(R, <a;b>)$
 $\langle \text{proof} \rangle$

lemma *qsplitE*:

$\llbracket \text{qsplit}(R,z); z:QSigma(A,B);$
 $\text{!!}x\ y. \llbracket z = <x;y>; R(x,y) \rrbracket \implies P$
 $\llbracket \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *qsplitD*: $\text{qsplit}(R, <a;b>) \implies R(a,b)$
 $\langle \text{proof} \rangle$

9.1.5 qconverse

lemma *qconverseI* [*intro!*]: $<a;b>:r \implies <b;a>:qconverse(r)$
 $\langle \text{proof} \rangle$

lemma *qconverseD* [*elim!*]: $<a;b> : qconverse(r) \implies <b;a> : r$
 $\langle \text{proof} \rangle$

lemma *qconverseE* [*elim!*]:

$\llbracket yx : qconverse(r);$
 $\text{!!}x\ y. \llbracket yx = <y;x>; <x;y>:r \rrbracket \implies P$
 $\llbracket \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *qconverse-qconverse*: $r <= QSigma(A,B) \implies qconverse(qconverse(r)) = r$
 $\langle \text{proof} \rangle$

lemma *qconverse-type*: $r <= A <*> B \implies qconverse(r) <= B <*> A$
 $\langle \text{proof} \rangle$

lemma *qconverse-prod*: $qconverse(A <*> B) = B <*> A$
 $\langle \text{proof} \rangle$

lemma *qconverse-empty*: $qconverse(0) = 0$
 $\langle \text{proof} \rangle$

9.2 The Quine-inspired notion of disjoint sum

lemmas *qsum-defs* = *qsum-def* *QInl-def* *QInr-def* *qcase-def*

lemma *QInlI* [*intro!*]: $a : A \implies QInl(a) : A <+> B$
 $\langle proof \rangle$

lemma *QInrI* [*intro!*]: $b : B \implies QInr(b) : A <+> B$
 $\langle proof \rangle$

lemma *qsumE* [*elim!*]:

$$\begin{aligned} & [| u : A <+> B; \\ & \quad !!x. [| x:A; u=QInl(x) |] \implies P; \\ & \quad !!y. [| y:B; u=QInr(y) |] \implies P \\ & |] \implies P \end{aligned}$$

 $\langle proof \rangle$

lemma *QInl-iff* [*iff*]: $QInl(a)=QInl(b) <-> a=b$
 $\langle proof \rangle$

lemma *QInr-iff* [*iff*]: $QInr(a)=QInr(b) <-> a=b$
 $\langle proof \rangle$

lemma *QInl-QInr-iff* [*simp*]: $QInl(a)=QInr(b) <-> False$
 $\langle proof \rangle$

lemma *QInr-QInl-iff* [*simp*]: $QInr(b)=QInl(a) <-> False$
 $\langle proof \rangle$

lemma *qsum-empty* [*simp*]: $0 <+> 0 = 0$
 $\langle proof \rangle$

lemmas *QInl-inject* = *QInl-iff* [*THEN iffD1, standard*]
lemmas *QInr-inject* = *QInr-iff* [*THEN iffD1, standard*]
lemmas *QInl-neq-QInr* = *QInl-QInr-iff* [*THEN iffD1, THEN FalseE, elim!*]
lemmas *QInr-neq-QInl* = *QInr-QInl-iff* [*THEN iffD1, THEN FalseE, elim!*]

lemma *QInlD*: $QInl(a) : A <+> B \implies a : A$
 $\langle proof \rangle$

lemma *QInrD*: $QInr(b) : A <+> B \implies b : B$

$\langle proof \rangle$

lemma *qsum-iff*:

$u: A <+> B <-> (EX\ x. x:A \ \&\ u=QInl(x)) \mid (EX\ y. y:B \ \&\ u=QInr(y))$
 $\langle proof \rangle$

lemma *qsum-subset-iff*: $A <+> B \leq C <+> D <-> A \leq C \ \&\ B \leq D$

$\langle proof \rangle$

lemma *qsum-equal-iff*: $A <+> B = C <+> D <-> A=C \ \&\ B=D$

$\langle proof \rangle$

9.2.1 Eliminator – qcase

lemma *qcase-QInl* [*simp*]: $qcase(c, d, QInl(a)) = c(a)$

$\langle proof \rangle$

lemma *qcase-QInr* [*simp*]: $qcase(c, d, QInr(b)) = d(b)$

$\langle proof \rangle$

lemma *qcase-type*:

$$\begin{aligned} & [\mid u: A <+> B; \\ & \quad !!x. x: A ==> c(x): C(QInl(x)); \\ & \quad !!y. y: B ==> d(y): C(QInr(y)) \\ &] ==> qcase(c,d,u) : C(u) \end{aligned}$$

 $\langle proof \rangle$

lemma *Part-QInl*: $Part(A <+> B, QInl) = \{QInl(x). x: A\}$

$\langle proof \rangle$

lemma *Part-QInr*: $Part(A <+> B, QInr) = \{QInr(y). y: B\}$

$\langle proof \rangle$

lemma *Part-QInr2*: $Part(A <+> B, \%x. QInr(h(x))) = \{QInr(y). y: Part(B,h)\}$

$\langle proof \rangle$

lemma *Part-qsum-equality*: $C \leq A <+> B ==> Part(C, QInl) \cup Part(C, QInr) = C$

$\langle proof \rangle$

9.2.2 Monotonicity

lemma *QPair-mono*: $[\mid a \leq c; \ b \leq d \] ==> \langle a;b \rangle \leq \langle c;d \rangle$

$\langle proof \rangle$

lemma *QSigma-mono* [rule-format]:

$$[\mid A \leq C; \text{ ALL } x:A. B(x) \leq D(x) \mid] \implies QSigma(A,B) \leq QSigma(C,D)$$
 $\langle proof \rangle$

lemma *QInl-mono*: $a \leq b \implies QInl(a) \leq QInl(b)$
 $\langle proof \rangle$

lemma *QInr-mono*: $a \leq b \implies QInr(a) \leq QInr(b)$
 $\langle proof \rangle$

lemma *qsum-mono*: $[\mid A \leq C; B \leq D \mid] \implies A <+> B \leq C <+> D$
 $\langle proof \rangle$

end

10 Perm: Injections, Surjections, Bijections, Composition

theory *Perm* **imports** *func* **begin**

definition

$comp \quad :: [i,i] \Rightarrow i \quad (\text{infixr } O \ 60) \quad \textbf{where}$
 $r \ O \ s == \{xz : domain(s)*range(r) \}.$
 $EX \ x \ y \ z. \ xz = \langle x,z \rangle \ \& \ \langle x,y \rangle : s \ \& \ \langle y,z \rangle : r \}$

definition

$id \quad :: i \Rightarrow i \quad \textbf{where}$
 $id(A) == (lam \ x:A. \ x)$

definition

$inj \quad :: [i,i] \Rightarrow i \quad \textbf{where}$
 $inj(A,B) == \{ f: A \rightarrow B. \text{ ALL } w:A. \text{ ALL } x:A. f'w = f'x \implies w = x \}$

definition

$surj \quad :: [i,i] \Rightarrow i \quad \textbf{where}$
 $surj(A,B) == \{ f: A \rightarrow B \ . \text{ ALL } y:B. \text{ EX } x:A. f'x = y \}$

definition

$bij \quad :: [i,i] \Rightarrow i \quad \textbf{where}$
 $bij(A,B) == inj(A,B) \ Int \ surj(A,B)$

10.1 Surjections

lemma *surj-is-fun*: $f: \text{surj}(A,B) \implies f: A \multimap B$
 $\langle \text{proof} \rangle$

lemma *fun-is-surj*: $f: \text{Pi}(A,B) \implies f: \text{surj}(A, \text{range}(f))$
 $\langle \text{proof} \rangle$

lemma *surj-range*: $f: \text{surj}(A,B) \implies \text{range}(f) = B$
 $\langle \text{proof} \rangle$

lemma *f-imp-surjective*:
 $\llbracket f: A \multimap B; \forall y. y: B \implies d(y): A; \forall y. y: B \implies f(d(y)) = y \rrbracket$
 $\implies f: \text{surj}(A,B)$
 $\langle \text{proof} \rangle$

lemma *lam-surjective*:
 $\llbracket \forall x. x: A \implies c(x): B;$
 $\forall y. y: B \implies d(y): A;$
 $\forall y. y: B \implies c(d(y)) = y$
 $\rrbracket \implies (\text{lam } x:A. c(x)) : \text{surj}(A,B)$
 $\langle \text{proof} \rangle$

lemma *cantor-surj*: $f \sim: \text{surj}(A, \text{Pow}(A))$
 $\langle \text{proof} \rangle$

10.2 Injections

lemma *inj-is-fun*: $f: \text{inj}(A,B) \implies f: A \multimap B$
 $\langle \text{proof} \rangle$

lemma *inj-equality*:
 $\llbracket \langle a, b \rangle : f; \langle c, b \rangle : f; f: \text{inj}(A,B) \rrbracket \implies a = c$
 $\langle \text{proof} \rangle$

lemma *inj-apply-equality*: $\llbracket f: \text{inj}(A,B); f'a = f'b; a:A; b:A \rrbracket \implies a = b$
 $\langle \text{proof} \rangle$

lemma *f-imp-injective*: $\llbracket f: A \multimap B; \text{ALL } x:A. d(f'x) = x \rrbracket \implies f: \text{inj}(A,B)$
 $\langle \text{proof} \rangle$

lemma *lam-injective*:
 $\llbracket \forall x. x: A \implies c(x): B;$

$$\begin{aligned} & \text{!!}x. x:A ==> d(c(x)) = x \text{]} \\ & ==> (\text{lam } x:A. c(x)) : \text{inj}(A,B) \\ & \langle \text{proof} \rangle \end{aligned}$$

10.3 Bijections

lemma *bij-is-inj*: $f: \text{bij}(A,B) ==> f: \text{inj}(A,B)$
 $\langle \text{proof} \rangle$

lemma *bij-is-surj*: $f: \text{bij}(A,B) ==> f: \text{surj}(A,B)$
 $\langle \text{proof} \rangle$

lemmas *bij-is-fun* = *bij-is-inj* [*THEN inj-is-fun, standard*]

lemma *lam-bijective*:

$$\begin{aligned} & [\text{!!}x. x:A ==> c(x): B; \\ & \quad \text{!!}y. y:B ==> d(y): A; \\ & \quad \text{!!}x. x:A ==> d(c(x)) = x; \\ & \quad \text{!!}y. y:B ==> c(d(y)) = y \\ &] ==> (\text{lam } x:A. c(x)) : \text{bij}(A,B) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *RepFun-bijective*: (*ALL* $y : x. \text{EX! } y'. f(y') = f(y)$)
 $==> (\text{lam } z:\{f(y). y:x\}. \text{THE } y. f(y) = z) : \text{bij}(\{f(y). y:x\}, x)$
 $\langle \text{proof} \rangle$

10.4 Identity Function

lemma *idI* [*intro!*]: $a:A ==> \langle a,a \rangle : \text{id}(A)$
 $\langle \text{proof} \rangle$

lemma *idE* [*elim!*]: $[p: \text{id}(A); \text{!!}x.[x:A; p=\langle x,x \rangle] ==> P] ==> P$
 $\langle \text{proof} \rangle$

lemma *id-type*: $\text{id}(A) : A \multimap A$
 $\langle \text{proof} \rangle$

lemma *id-conv* [*simp*]: $x:A ==> \text{id}(A) 'x = x$
 $\langle \text{proof} \rangle$

lemma *id-mono*: $A \leq B ==> \text{id}(A) \leq \text{id}(B)$
 $\langle \text{proof} \rangle$

lemma *id-subset-inj*: $A \leq B ==> \text{id}(A): \text{inj}(A,B)$
 $\langle \text{proof} \rangle$

lemmas *id-inj* = *subset-refl* [*THEN id-subset-inj, standard*]

lemma *id-surj*: $\text{id}(A): \text{surj}(A,A)$

$\langle proof \rangle$

lemma *id-bij*: $id(A): bij(A,A)$
 $\langle proof \rangle$

lemma *subset-iff-id*: $A \leq B \iff id(A) : A \rightarrow B$
 $\langle proof \rangle$

id as the identity relation

lemma *id-iff [simp]*: $\langle x,y \rangle \in id(A) \iff x=y \ \& \ y \in A$
 $\langle proof \rangle$

10.5 Converse of a Function

lemma *inj-converse-fun*: $f: inj(A,B) \implies converse(f) : range(f) \rightarrow A$
 $\langle proof \rangle$

The premises are equivalent to saying that *f* is injective...

lemma *left-inverse-lemma*:
 $[[f: A \rightarrow B; \ converse(f): C \rightarrow A; \ a: A]] \implies converse(f) ' (f'a) = a$
 $\langle proof \rangle$

lemma *left-inverse [simp]*: $[[f: inj(A,B); \ a: A]] \implies converse(f) ' (f'a) = a$
 $\langle proof \rangle$

lemma *left-inverse-eq*:
 $[[f \in inj(A,B); \ f ' x = y; \ x \in A]] \implies converse(f) ' y = x$
 $\langle proof \rangle$

lemmas *left-inverse-bij = bij-is-inj [THEN left-inverse, standard]*

lemma *right-inverse-lemma*:
 $[[f: A \rightarrow B; \ converse(f): C \rightarrow A; \ b: C]] \implies f ' (converse(f) ' b) = b$
 $\langle proof \rangle$

lemma *right-inverse [simp]*:
 $[[f: inj(A,B); \ b: range(f)]] \implies f ' (converse(f) ' b) = b$
 $\langle proof \rangle$

lemma *right-inverse-bij*: $[[f: bij(A,B); \ b: B]] \implies f ' (converse(f) ' b) = b$
 $\langle proof \rangle$

10.6 Converses of Injections, Surjections, Bijections

lemma *inj-converse-inj*: $f: inj(A,B) \implies converse(f): inj(range(f), A)$
 $\langle proof \rangle$

lemma *inj-converse-surj*: $f: inj(A,B) \implies converse(f): surj(range(f), A)$

$\langle proof \rangle$

lemma *bij-converse-bij* [TC]: $f: \text{bij}(A,B) \implies \text{converse}(f): \text{bij}(B,A)$
 $\langle proof \rangle$

10.7 Composition of Two Relations

lemma *compI* [intro]: $[\![\langle a,b \rangle : s; \langle b,c \rangle : r]\!] \implies \langle a,c \rangle : r \circ s$
 $\langle proof \rangle$

lemma *compE* [elim!]:
 $[\![xz : r \circ s;$
 $!!x \ y \ z. [\![xz = \langle x,z \rangle; \langle x,y \rangle : s; \langle y,z \rangle : r]\!] \implies P]\!] \implies P$
 $\langle proof \rangle$

lemma *compEpair*:
 $[\![\langle a,c \rangle : r \circ s;$
 $!!y. [\![\langle a,y \rangle : s; \langle y,c \rangle : r]\!] \implies P]\!] \implies P$
 $\langle proof \rangle$

lemma *converse-comp*: $\text{converse}(R \circ S) = \text{converse}(S) \circ \text{converse}(R)$
 $\langle proof \rangle$

10.8 Domain and Range – see Suppes, Section 3.1

lemma *range-comp*: $\text{range}(r \circ s) \leq \text{range}(r)$
 $\langle proof \rangle$

lemma *range-comp-eq*: $\text{domain}(r) \leq \text{range}(s) \implies \text{range}(r \circ s) = \text{range}(r)$
 $\langle proof \rangle$

lemma *domain-comp*: $\text{domain}(r \circ s) \leq \text{domain}(s)$
 $\langle proof \rangle$

lemma *domain-comp-eq*: $\text{range}(s) \leq \text{domain}(r) \implies \text{domain}(r \circ s) = \text{domain}(s)$
 $\langle proof \rangle$

lemma *image-comp*: $(r \circ s)^{''}A = r^{''}(s^{''}A)$
 $\langle proof \rangle$

10.9 Other Results

lemma *comp-mono*: $[\![r' \leq r; s' \leq s]\!] \implies (r' \circ s') \leq (r \circ s)$
 $\langle proof \rangle$

lemma *comp-rel*: $[[s \leq A * B; r \leq B * C]] \implies (r \circ s) \leq A * C$
 $\langle proof \rangle$

lemma *comp-assoc*: $(r \circ s) \circ t = r \circ (s \circ t)$
 $\langle proof \rangle$

lemma *left-comp-id*: $r \leq A * B \implies id(B) \circ r = r$
 $\langle proof \rangle$

lemma *right-comp-id*: $r \leq A * B \implies r \circ id(A) = r$
 $\langle proof \rangle$

10.10 Composition Preserves Functions, Injections, and Surjections

lemma *comp-function*: $[[function(g); function(f)]] \implies function(f \circ g)$
 $\langle proof \rangle$

lemma *comp-fun*: $[[g: A \multimap B; f: B \multimap C]] \implies (f \circ g) : A \multimap C$
 $\langle proof \rangle$

lemma *comp-fun-apply* [simp]:
 $[[g: A \multimap B; a:A]] \implies (f \circ g) 'a = f '(g 'a)$
 $\langle proof \rangle$

lemma *comp-lam*:
 $[[!!x. x:A \implies b(x): B]]$
 $\implies (lam y:B. c(y)) \circ (lam x:A. b(x)) = (lam x:A. c(b(x)))$
 $\langle proof \rangle$

lemma *comp-inj*:
 $[[g: inj(A,B); f: inj(B,C)]] \implies (f \circ g) : inj(A,C)$
 $\langle proof \rangle$

lemma *comp-surj*:
 $[[g: surj(A,B); f: surj(B,C)]] \implies (f \circ g) : surj(A,C)$
 $\langle proof \rangle$

lemma *comp-bij*:
 $[[g: bij(A,B); f: bij(B,C)]] \implies (f \circ g) : bij(A,C)$
 $\langle proof \rangle$

10.11 Dual Properties of *inj* and *surj*

Useful for proofs from D Pastre. Automatic theorem proving in set theory. Artificial Intelligence, 10:1–27, 1978.

lemma *comp-mem-injD1*:

$$[[(f \circ g): inj(A,C); \ g: A \rightarrow B; \ f: B \rightarrow C]] ==> g: inj(A,B)$$

<proof>

lemma *comp-mem-injD2*:

$$[[(f \circ g): inj(A,C); \ g: surj(A,B); \ f: B \rightarrow C]] ==> f: inj(B,C)$$

<proof>

lemma *comp-mem-surjD1*:

$$[[(f \circ g): surj(A,C); \ g: A \rightarrow B; \ f: B \rightarrow C]] ==> f: surj(B,C)$$

<proof>

lemma *comp-mem-surjD2*:

$$[[(f \circ g): surj(A,C); \ g: A \rightarrow B; \ f: inj(B,C)]] ==> g: surj(A,B)$$

<proof>

10.11.1 Inverses of Composition

lemma *left-comp-inverse*: $f: inj(A,B) ==> converse(f) \circ f = id(A)$
<proof>

lemma *right-comp-inverse*:

$f: surj(A,B) ==> f \circ converse(f) = id(B)$
<proof>

10.11.2 Proving that a Function is a Bijection

lemma *comp-eq-id-iff*:

$$[[f: A \rightarrow B; \ g: B \rightarrow A]] ==> f \circ g = id(B) \leftrightarrow (ALL\ y:B. f'(g'y)=y)$$

<proof>

lemma *fg-imp-bijective*:

$$[[f: A \rightarrow B; \ g: B \rightarrow A; \ f \circ g = id(B); \ g \circ f = id(A)]] ==> f: bij(A,B)$$

<proof>

lemma *nilpotent-imp-bijective*: $[[f: A \rightarrow A; \ f \circ f = id(A)]] ==> f: bij(A,A)$
<proof>

lemma *invertible-imp-bijective*:

$$[[converse(f): B \rightarrow A; \ f: A \rightarrow B]] ==> f: bij(A,B)$$

<proof>

10.11.3 Unions of Functions

See similar theorems in `func.thy`

lemma *inj-disjoint-Un*:

$$\begin{aligned} & [[f: \text{inj}(A,B); \quad g: \text{inj}(C,D); \quad B \text{ Int } D = 0 \]] \\ & \implies (\text{lam } a: A \text{ Un } C. \text{ if } a:A \text{ then } f'a \text{ else } g'a) : \text{inj}(A \text{ Un } C, B \text{ Un } D) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *surj-disjoint-Un*:

$$\begin{aligned} & [[f: \text{surj}(A,B); \quad g: \text{surj}(C,D); \quad A \text{ Int } C = 0 \]] \\ & \implies (f \text{ Un } g) : \text{surj}(A \text{ Un } C, B \text{ Un } D) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *bij-disjoint-Un*:

$$\begin{aligned} & [[f: \text{bij}(A,B); \quad g: \text{bij}(C,D); \quad A \text{ Int } C = 0; \quad B \text{ Int } D = 0 \]] \\ & \implies (f \text{ Un } g) : \text{bij}(A \text{ Un } C, B \text{ Un } D) \\ & \langle \text{proof} \rangle \end{aligned}$$

10.11.4 Restrictions as Surjections and Bijections

lemma *surj-image*:

$$f: \text{Pi}(A,B) \implies f: \text{surj}(A, f''A)$$

$\langle \text{proof} \rangle$

lemma *restrict-image [simp]*: $\text{restrict}(f,A) \text{ `` } B = f \text{ `` } (A \text{ Int } B)$

$\langle \text{proof} \rangle$

lemma *restrict-inj*:

$$[[f: \text{inj}(A,B); \quad C \leq A \]] \implies \text{restrict}(f,C): \text{inj}(C,B)$$

$\langle \text{proof} \rangle$

lemma *restrict-surj*: $[[f: \text{Pi}(A,B); \quad C \leq A \]] \implies \text{restrict}(f,C): \text{surj}(C, f''C)$

$\langle \text{proof} \rangle$

lemma *restrict-bij*:

$$[[f: \text{inj}(A,B); \quad C \leq A \]] \implies \text{restrict}(f,C): \text{bij}(C, f''C)$$

$\langle \text{proof} \rangle$

10.11.5 Lemmas for Ramsey's Theorem

lemma *inj-weaken-type*: $[[f: \text{inj}(A,B); \quad B \leq D \]] \implies f: \text{inj}(A,D)$

$\langle \text{proof} \rangle$

lemma *inj-succ-restrict*:

$$[[f: \text{inj}(\text{succ}(m), A) \]] \implies \text{restrict}(f,m) : \text{inj}(m, A - \{f'm\})$$

$\langle \text{proof} \rangle$

```

lemma inj-extend:
  [|  $f: inj(A,B); a \sim A; b \sim B$  |]
  ==>  $cons(<a,b>,f) : inj(cons(a,A), cons(b,B))$ 
  <proof>

end

```

11 Tranc1: Relations: Their General Properties and Transitive Closure

```

theory Tranc1 imports Fixedpt Perm begin

```

```

definition
  refl    :: [ $i,i$ ]==> $o$  where
     $refl(A,r) == (ALL x: A. <x,x> : r)$ 

```

```

definition
  irrefl  :: [ $i,i$ ]==> $o$  where
     $irrefl(A,r) == ALL x: A. <x,x> \sim: r$ 

```

```

definition
  sym     ::  $i==>o$  where
     $sym(r) == ALL x y. <x,y>: r \longrightarrow <y,x>: r$ 

```

```

definition
  asym    ::  $i==>o$  where
     $asym(r) == ALL x y. <x,y>: r \longrightarrow \sim <y,x>: r$ 

```

```

definition
  antisym ::  $i==>o$  where
     $antisym(r) == ALL x y. <x,y>: r \longrightarrow <y,x>: r \longrightarrow x=y$ 

```

```

definition
  trans   ::  $i==>o$  where
     $trans(r) == ALL x y z. <x,y>: r \longrightarrow <y,z>: r \longrightarrow <x,z>: r$ 

```

```

definition
  trans-on :: [ $i,i$ ]==> $o$  (trans[-]'(-')) where
     $trans[A](r) == ALL x:A. ALL y:A. ALL z:A. <x,y>: r \longrightarrow <y,z>: r \longrightarrow <x,z>: r$ 

```

```

definition
  rtranc1 ::  $i==>i$  ((-^*) [100] 100) where
     $r^{\wedge *} == lfp(field(r)*field(r), \%s. id(field(r)) \ Un (r \ O \ s))$ 

```

```

definition
  tranc1  ::  $i==>i$  ((-^+) [100] 100) where

```

$$r^{\wedge}+ == r \ O \ r^{\wedge}*$$

definition

equiv :: $[i,i] \Rightarrow o$ **where**
equiv(A,r) == $r \leq A*A \ \& \ refl(A,r) \ \& \ sym(r) \ \& \ trans(r)$

11.1 General properties of relations

11.1.1 irreflexivity

lemma *irreflI*:

$[[\ !x. x:A \Rightarrow \langle x,x \rangle \sim : r \]] \Rightarrow irrefl(A,r)$
 $\langle proof \rangle$

lemma *irreflE*: $[[irrefl(A,r); x:A \]] \Rightarrow \langle x,x \rangle \sim : r$
 $\langle proof \rangle$

11.1.2 symmetry

lemma *symI*:

$[[\ !x \ y. \langle x,y \rangle : r \Rightarrow \langle y,x \rangle : r \]] \Rightarrow sym(r)$
 $\langle proof \rangle$

lemma *symE*: $[[sym(r); \langle x,y \rangle : r \]] \Rightarrow \langle y,x \rangle : r$
 $\langle proof \rangle$

11.1.3 antisymmetry

lemma *antisymI*:

$[[\ !x \ y. [\langle x,y \rangle : r; \langle y,x \rangle : r \] \Rightarrow x=y \]] \Rightarrow antisym(r)$
 $\langle proof \rangle$

lemma *antisymE*: $[[antisym(r); \langle x,y \rangle : r; \langle y,x \rangle : r \]] \Rightarrow x=y$
 $\langle proof \rangle$

11.1.4 transitivity

lemma *transD*: $[[trans(r); \langle a,b \rangle : r; \langle b,c \rangle : r \]] \Rightarrow \langle a,c \rangle : r$
 $\langle proof \rangle$

lemma *trans-onD*:

$[[trans[A](r); \langle a,b \rangle : r; \langle b,c \rangle : r; a:A; b:A; c:A \]] \Rightarrow \langle a,c \rangle : r$
 $\langle proof \rangle$

lemma *trans-imp-trans-on*: $trans(r) \Rightarrow trans[A](r)$
 $\langle proof \rangle$

lemma *trans-on-imp-trans*: $[[trans[A](r); r \leq A*A \]] \Rightarrow trans(r)$
 $\langle proof \rangle$

11.2 Transitive closure of a relation

lemma *rtrancl-bnd-mono*:

$bnd\text{-}mono(field(r)*field(r), \%s. id(field(r)) \cup n (r \circ s))$
 $\langle proof \rangle$

lemma *rtrancl-mono*: $r \leq s \implies r^* \leq s^*$

$\langle proof \rangle$

lemmas *rtrancl-unfold* =

rtrancl-bnd-mono [THEN *rtrancl-def* [THEN *def-lfp-unfold*], *standard*]

lemmas *rtrancl-type* = *rtrancl-def* [THEN *def-lfp-subset*, *standard*]

lemma *relation-rtrancl*: $relation(r^*)$

$\langle proof \rangle$

lemma *rtrancl-refl*: $[| a: field(r) |] \implies \langle a, a \rangle : r^*$

$\langle proof \rangle$

lemma *rtrancl-into-rtrancl*: $[| \langle a, b \rangle : r^*; \langle b, c \rangle : r |] \implies \langle a, c \rangle : r^*$

$\langle proof \rangle$

lemma *r-into-rtrancl*: $\langle a, b \rangle : r \implies \langle a, b \rangle : r^*$

$\langle proof \rangle$

lemma *r-subset-rtrancl*: $relation(r) \implies r \leq r^*$

$\langle proof \rangle$

lemma *rtrancl-field*: $field(r^*) = field(r)$

$\langle proof \rangle$

lemma *rtrancl-full-induct* [*case-names initial step, consumes 1*]:

$[| \langle a, b \rangle : r^*;$
 $!!x. x: field(r) \implies P(\langle x, x \rangle);$
 $!!x\ y\ z. [| P(\langle x, y \rangle); \langle x, y \rangle : r^*; \langle y, z \rangle : r |] \implies P(\langle x, z \rangle) |]$
 $\implies P(\langle a, b \rangle)$
 $\langle proof \rangle$

lemma *rtrancI-induct* [*case-names initial step, induct set: rtrancI*]:

[[$\langle a, b \rangle : r^*$;
 $P(a)$;
 $!!y\ z. [\langle a, y \rangle : r^*; \langle y, z \rangle : r; P(y)] \implies P(z)$
 $]] \implies P(b)$

$\langle proof \rangle$

lemma *trans-rtrancI*: $trans(r^*)$

$\langle proof \rangle$

lemmas *rtrancI-trans* = *trans-rtrancI* [*THEN transD, standard*]

lemma *rtrancIE*:

[[$\langle a, b \rangle : r^*$; $(a=b) \implies P$;
 $!!y. [\langle a, y \rangle : r^*; \langle y, b \rangle : r] \implies P$]
 $\implies P$

$\langle proof \rangle$

lemma *trans-trancI*: $trans(r^+)$

$\langle proof \rangle$

lemmas *trans-on-trancI* = *trans-trancI* [*THEN trans-imp-trans-on*]

lemmas *trancI-trans* = *trans-trancI* [*THEN transD, standard*]

lemma *trancI-into-rtrancI*: $\langle a, b \rangle : r^+ \implies \langle a, b \rangle : r^*$

$\langle proof \rangle$

lemma *r-into-trancI*: $\langle a, b \rangle : r \implies \langle a, b \rangle : r^+$

$\langle proof \rangle$

lemma *r-subset-trancI*: $relation(r) \implies r \leq r^+$

$\langle proof \rangle$

lemma *rtrancI-into-trancI1*: $[\langle a, b \rangle : r^*; \langle b, c \rangle : r] \implies \langle a, c \rangle : r^+$

$\langle proof \rangle$

lemma *rtranc1-into-tranc12*:

$\llbracket \langle a, b \rangle : r; \langle b, c \rangle : r^{\wedge *} \rrbracket \implies \langle a, c \rangle : r^{\wedge +}$
 $\langle proof \rangle$

lemma *tranc1-induct* [*case-names initial step, induct set: tranc1*]:

$\llbracket \langle a, b \rangle : r^{\wedge +};$
 $\quad !!y. \llbracket \langle a, y \rangle : r \rrbracket \implies P(y);$
 $\quad !!y z. \llbracket \langle a, y \rangle : r^{\wedge +}; \langle y, z \rangle : r; P(y) \rrbracket \implies P(z)$
 $\rrbracket \implies P(b)$
 $\langle proof \rangle$

lemma *tranc1E*:

$\llbracket \langle a, b \rangle : r^{\wedge +};$
 $\quad \langle a, b \rangle : r \implies P;$
 $\quad !!y. \llbracket \langle a, y \rangle : r^{\wedge +}; \langle y, b \rangle : r \rrbracket \implies P$
 $\rrbracket \implies P$
 $\langle proof \rangle$

lemma *tranc1-type*: $r^{\wedge +} \leq \text{field}(r) * \text{field}(r)$
 $\langle proof \rangle$

lemma *relation-tranc1*: $\text{relation}(r^{\wedge +})$
 $\langle proof \rangle$

lemma *tranc1-subset-times*: $r \subseteq A * A \implies r^{\wedge +} \subseteq A * A$
 $\langle proof \rangle$

lemma *tranc1-mono*: $r \leq s \implies r^{\wedge +} \leq s^{\wedge +}$
 $\langle proof \rangle$

lemma *tranc1-eq-r*: $\llbracket \text{relation}(r); \text{trans}(r) \rrbracket \implies r^{\wedge +} = r$
 $\langle proof \rangle$

lemma *rtranc1-idemp* [*simp*]: $(r^{\wedge *})^{\wedge *} = r^{\wedge *}$
 $\langle proof \rangle$

lemma *rtranc1-subset*: $\llbracket R \leq S; S \leq R^{\wedge *} \rrbracket \implies S^{\wedge *} = R^{\wedge *}$
 $\langle proof \rangle$

lemma *rtranc1-Un-rtranc1*:

$\llbracket \text{relation}(r); \text{relation}(s) \rrbracket \implies (r^{\wedge *} \text{ Un } s^{\wedge *})^{\wedge *} = (r \text{ Un } s)^{\wedge *}$

$\langle proof \rangle$

lemma *rtrancl-converseD*: $\langle x, y \rangle : converse(r)^* \implies \langle x, y \rangle : converse(r^*)$
 $\langle proof \rangle$

lemma *rtrancl-converseI*: $\langle x, y \rangle : converse(r^*) \implies \langle x, y \rangle : converse(r)^*$
 $\langle proof \rangle$

lemma *rtrancl-converse*: $converse(r)^* = converse(r^*)$
 $\langle proof \rangle$

lemma *trancl-converseD*: $\langle a, b \rangle : converse(r)^+ \implies \langle a, b \rangle : converse(r^+)$
 $\langle proof \rangle$

lemma *trancl-converseI*: $\langle x, y \rangle : converse(r^+) \implies \langle x, y \rangle : converse(r)^+$
 $\langle proof \rangle$

lemma *trancl-converse*: $converse(r)^+ = converse(r^+)$
 $\langle proof \rangle$

lemma *converse-trancl-induct* [*case-names initial step, consumes 1*]:

$$\begin{aligned} & [[\langle a, b \rangle : r^+; !!y. \langle y, b \rangle : r \implies P(y); \\ & \quad !!y z. [[\langle y, z \rangle : r; \langle z, b \rangle : r^+; P(z)]] \implies P(y)]] \\ & \implies P(a) \end{aligned}$$

 $\langle proof \rangle$

end

12 WF: Well-Founded Recursion

theory *WF* **imports** *Trancl* **begin**

definition

$wf \quad :: i \Rightarrow o \quad \mathbf{where}$

$wf(r) == ALL\ Z. Z=0 \mid (EX\ x:Z. ALL\ y. \langle y, x \rangle : r \longrightarrow \sim y:Z)$

definition

$wf-on \quad :: [i, i] \Rightarrow o \quad (wf[-]'\langle - \rangle) \quad \mathbf{where}$

$wf-on(A, r) == wf(r\ Int\ A * A)$

definition

$is-recfun \quad :: [i, i, [i,i]=>i, i] => o \text{ where}$
 $is-recfun(r,a,H,f) == (f = (lam x: r-'\{a\}. H(x, restrict(f, r-'\{x\}))))$

definition

$the-recfun \quad :: [i, i, [i,i]=>i] => i \text{ where}$
 $the-recfun(r,a,H) == (THE f. is-recfun(r,a,H,f))$

definition

$wftrec \quad :: [i, i, [i,i]=>i] => i \text{ where}$
 $wftrec(r,a,H) == H(a, the-recfun(r,a,H))$

definition

$wfrec \quad :: [i, i, [i,i]=>i] => i \text{ where}$
 $wfrec(r,a,H) == wftrec(r^+ +, a, \%x f. H(x, restrict(f,r-'\{x\})))$

definition

$wfrec-on \quad :: [i, i, i, [i,i]=>i] => i \quad (wfrec[-]'(-,-,-)) \text{ where}$
 $wfrec[A](r,a,H) == wfrec(r \text{ Int } A * A, a, H)$

12.1 Well-Founded Relations**12.1.1 Equivalences between wf and $wf-on$**

lemma $wf-imp-wf-on$: $wf(r) ==> wf[A](r)$
 $\langle proof \rangle$

lemma $wf-on-imp-wf$: $[wf[A](r); r <= A * A] ==> wf(r)$
 $\langle proof \rangle$

lemma $wf-on-field-imp-wf$: $wf[field(r)](r) ==> wf(r)$
 $\langle proof \rangle$

lemma $wf-iff-wf-on-field$: $wf(r) <-> wf[field(r)](r)$
 $\langle proof \rangle$

lemma $wf-on-subset-A$: $[wf[A](r); B <= A] ==> wf[B](r)$
 $\langle proof \rangle$

lemma $wf-on-subset-r$: $[wf[A](r); s <= r] ==> wf[A](s)$
 $\langle proof \rangle$

lemma $wf-subset$: $[wf(s); r <= s] ==> wf(r)$
 $\langle proof \rangle$

12.1.2 Introduction Rules for $wf-on$

If every non-empty subset of A has an r -minimal element then we have $wf[A](r)$.

lemma *wf-onI*:
assumes *prem*: $!!Z\ u. [| Z \leq A; \ u:Z; \ \text{ALL } x:Z. \ \text{EX } y:Z. \ \langle y, x \rangle : r |] \implies \text{False}$
shows $\text{wf}[A](r)$
 $\langle \text{proof} \rangle$

If r allows well-founded induction over A then we have $\text{wf}[A](r)$. Premise is equivalent to $\bigwedge B. \forall x \in A. (\forall y. \langle y, x \rangle \in r \longrightarrow y \in B) \longrightarrow x \in B \implies A \subseteq B$

lemma *wf-onI2*:
assumes *prem*: $!!y\ B. [| \text{ALL } x:A. (\text{ALL } y:A. \ \langle y, x \rangle : r \longrightarrow y:B) \longrightarrow x:B; \ y:A |]$
 $\implies y:B$
shows $\text{wf}[A](r)$
 $\langle \text{proof} \rangle$

12.1.3 Well-founded Induction

Consider the least z in $\text{domain}(r)$ such that $P(z)$ does not hold...

lemma *wf-induct* [*induct set*: *wf*]:
 $[| \text{wf}(r);$
 $\quad !!x. [| \text{ALL } y. \ \langle y, x \rangle : r \longrightarrow P(y) |] \implies P(x) |]$
 $\implies P(a)$
 $\langle \text{proof} \rangle$

lemmas *wf-induct-rule* = *wf-induct* [*rule-format*, *induct set*: *wf*]

The form of this rule is designed to match *wfI*

lemma *wf-induct2*:
 $[| \text{wf}(r); \ a:A; \ \text{field}(r) \leq A;$
 $\quad !!x. [| x:A; \ \text{ALL } y. \ \langle y, x \rangle : r \longrightarrow P(y) |] \implies P(x) |]$
 $\implies P(a)$
 $\langle \text{proof} \rangle$

lemma *field-Int-square*: $\text{field}(r \text{ Int } A * A) \leq A$
 $\langle \text{proof} \rangle$

lemma *wf-on-induct* [*consumes 2*, *induct set*: *wf-on*]:
 $[| \text{wf}[A](r); \ a:A;$
 $\quad !!x. [| x:A; \ \text{ALL } y:A. \ \langle y, x \rangle : r \longrightarrow P(y) |] \implies P(x)$
 $|] \implies P(a)$
 $\langle \text{proof} \rangle$

lemmas *wf-on-induct-rule* =
wf-on-induct [*rule-format*, *consumes 2*, *induct set*: *wf-on*]

If r allows well-founded induction then we have $\text{wf}(r)$.

lemma *wfI*:
 $[| \text{field}(r) \leq A;$

$$\begin{aligned} & !!y B. [| ALL x:A. (ALL y:A. <y,x>:r \dashrightarrow y:B) \dashrightarrow x:B; y:A|] \\ & \quad \quad \quad ==> y:B [|] \\ & ==> wf(r) \\ \langle proof \rangle \end{aligned}$$

12.2 Basic Properties of Well-Founded Relations

lemma *wf-not-refl*: $wf(r) ==> <a,a> \sim: r$
 $\langle proof \rangle$

lemma *wf-not-sym* [*rule-format*]: $wf(r) ==> ALL x. <a,x>:r \dashrightarrow <x,a> \sim: r$
 $\langle proof \rangle$

lemmas *wf-asy* = *wf-not-sym* [*THEN* *swap*, *standard*]

lemma *wf-on-not-refl*: $[| wf[A](r); a:A |] ==> <a,a> \sim: r$
 $\langle proof \rangle$

lemma *wf-on-not-sym* [*rule-format*]:
 $[| wf[A](r); a:A |] ==> ALL b:A. <a,b>:r \dashrightarrow <b,a> \sim: r$
 $\langle proof \rangle$

lemma *wf-on-asy*:
 $[| wf[A](r); \sim Z ==> <a,b> : r;$
 $\quad <b,a> \sim: r ==> Z; \sim Z ==> a : A; \sim Z ==> b : A |] ==> Z$
 $\langle proof \rangle$

lemma *wf-on-chain3*:
 $[| wf[A](r); <a,b>:r; <b,c>:r; <c,a>:r; a:A; b:A; c:A |] ==> P$
 $\langle proof \rangle$

transitive closure of a WF relation is WF provided A is downward closed

lemma *wf-on-trancl*:
 $[| wf[A](r); r - "A \leq A |] ==> wf[A](r^+)$
 $\langle proof \rangle$

lemma *wf-trancl*: $wf(r) ==> wf(r^+)$
 $\langle proof \rangle$

$r - "$ $\{a\}$ is the set of everything under a in r

lemmas *underI* = *vimage-singleton-iff* [*THEN* *iffD2*, *standard*]

lemmas *underD* = *vimage-singleton-iff* [*THEN* *iffD1*, *standard*]

12.3 The Predicate *is-recfun*

lemma *is-recfun-type*: $is-recfun(r,a,H,f) ==> f: r - "\{a\} \rightarrow range(f)$

$\langle \text{proof} \rangle$

lemmas *is-recfun-imp-function* = *is-recfun-type* [THEN *fun-is-function*]

lemma *apply-recfun*:

$[[\text{is-recfun}(r, a, H, f); \langle x, a \rangle : r]] \implies f'x = H(x, \text{restrict}(f, r - \{\{x\}\}))$
 $\langle \text{proof} \rangle$

lemma *is-recfun-equal* [rule-format]:

$[[\text{wf}(r); \text{trans}(r); \text{is-recfun}(r, a, H, f); \text{is-recfun}(r, b, H, g)]]$
 $\implies \langle x, a \rangle : r \dashv\dashv \langle x, b \rangle : r \dashv\dashv f'x = g'x$
 $\langle \text{proof} \rangle$

lemma *is-recfun-cut*:

$[[\text{wf}(r); \text{trans}(r);$
 $\text{is-recfun}(r, a, H, f); \text{is-recfun}(r, b, H, g); \langle b, a \rangle : r]]$
 $\implies \text{restrict}(f, r - \{\{b\}\}) = g$
 $\langle \text{proof} \rangle$

12.4 Recursion: Main Existence Lemma

lemma *is-recfun-functional*:

$[[\text{wf}(r); \text{trans}(r); \text{is-recfun}(r, a, H, f); \text{is-recfun}(r, a, H, g)]] \implies f = g$
 $\langle \text{proof} \rangle$

lemma *the-recfun-eq*:

$[[\text{is-recfun}(r, a, H, f); \text{wf}(r); \text{trans}(r)]] \implies \text{the-recfun}(r, a, H) = f$
 $\langle \text{proof} \rangle$

lemma *is-the-recfun*:

$[[\text{is-recfun}(r, a, H, f); \text{wf}(r); \text{trans}(r)]]$
 $\implies \text{is-recfun}(r, a, H, \text{the-recfun}(r, a, H))$
 $\langle \text{proof} \rangle$

lemma *unfold-the-recfun*:

$[[\text{wf}(r); \text{trans}(r)]] \implies \text{is-recfun}(r, a, H, \text{the-recfun}(r, a, H))$
 $\langle \text{proof} \rangle$

12.5 Unfolding *wftrec*(*r*, *a*, *H*)

lemma *the-recfun-cut*:

$[[\text{wf}(r); \text{trans}(r); \langle b, a \rangle : r]]$
 $\implies \text{restrict}(\text{the-recfun}(r, a, H), r - \{\{b\}\}) = \text{the-recfun}(r, b, H)$
 $\langle \text{proof} \rangle$

lemma *wftrec*:

$[[\text{wf}(r); \text{trans}(r)]] \implies$
 $\text{wftrec}(r, a, H) = H(a, \text{lam } x: r - \{\{a\}\}. \text{wftrec}(r, x, H))$

$\langle proof \rangle$

12.5.1 Removal of the Premise $trans(r)$

lemma *wfrec*:

$wf(r) \implies wfrec(r, a, H) = H(a, \text{lam } x:r - \{\{a\}. wfrec(r, x, H)\})$
 $\langle proof \rangle$

lemma *def-wfrec*:

$[\text{!}x. h(x) = wfrec(r, x, H); wf(r)] \implies$
 $h(a) = H(a, \text{lam } x: r - \{\{a\}. h(x)\})$
 $\langle proof \rangle$

lemma *wfrec-type*:

$[wf(r); a:A; field(r) \leq A;$
 $\text{!}x\ u. [x: A; u: Pi(r - \{\{x\}, B)] \implies H(x, u) : B(x)$
 $] \implies wfrec(r, a, H) : B(a)$
 $\langle proof \rangle$

lemma *wfrec-on*:

$[wf[A](r); a: A] \implies$
 $wfrec[A](r, a, H) = H(a, \text{lam } x: (r - \{\{a\}\} Int A. wfrec[A](r, x, H))$
 $\langle proof \rangle$

Minimal-element characterization of well-foundedness

lemma *wf-eq-minimal*:

$wf(r) <-> (ALL\ Q\ x. x:Q \dashrightarrow (EX\ z:Q. ALL\ y. <y, z>:r \dashrightarrow y \sim :Q))$
 $\langle proof \rangle$

end

13 Ordinal: Transitive Sets and Ordinals

theory *Ordinal* **imports** *WF Bool equalities* **begin**

definition

$Memrel \quad :: i \Rightarrow i \text{ where}$
 $Memrel(A) \quad == \{z: A * A . EX\ x\ y. z = <x, y> \ \& \ x : y \}$

definition

$Transset \quad :: i \Rightarrow o \text{ where}$
 $Transset(i) == ALL\ x:i. x \leq i$

definition

$Ord \quad :: i \Rightarrow o \text{ where}$
 $Ord(i) \quad == Transset(i) \ \& \ (ALL\ x:i. Transset(x))$

definition

$lt \quad :: [i,i] \Rightarrow o \text{ (infixl } < 50) \quad \text{where}$
 $i < j \quad == i:j \ \& \ Ord(j)$

definition

$Limit \quad :: i \Rightarrow o \text{ where}$
 $Limit(i) \quad == Ord(i) \ \& \ 0 < i \ \& \ (ALL \ y. \ y < i \ \longrightarrow \ succ(y) < i)$

abbreviation

$le \text{ (infixl } le \ 50) \text{ where}$
 $x \ le \ y == x < succ(y)$

notation (*xsymbols*)

$le \text{ (infixl } \leq 50)$

notation (*HTML output*)

$le \text{ (infixl } \leq 50)$

13.1 Rules for Transset**13.1.1 Three Neat Characterisations of Transset**

lemma *Transset-iff-Pow*: $Transset(A) <-> A \leq Pow(A)$
 $\langle proof \rangle$

lemma *Transset-iff-Union-succ*: $Transset(A) <-> Union(succ(A)) = A$
 $\langle proof \rangle$

lemma *Transset-iff-Union-subset*: $Transset(A) <-> Union(A) \leq A$
 $\langle proof \rangle$

13.1.2 Consequences of Downwards Closure

lemma *Transset-doubleton-D*:
 $[| \ Transset(C); \{a,b\}: C \ |] \Rightarrow a:C \ \& \ b: C$
 $\langle proof \rangle$

lemma *Transset-Pair-D*:
 $[| \ Transset(C); <a,b>: C \ |] \Rightarrow a:C \ \& \ b: C$
 $\langle proof \rangle$

lemma *Transset-includes-domain*:
 $[| \ Transset(C); A*B \leq C; b: B \ |] \Rightarrow A \leq C$
 $\langle proof \rangle$

lemma *Transset-includes-range*:
 $[| \ Transset(C); A*B \leq C; a: A \ |] \Rightarrow B \leq C$
 $\langle proof \rangle$

13.1.3 Closure Properties

lemma *Transset-0*: $\text{Transset}(0)$

$\langle \text{proof} \rangle$

lemma *Transset-Un*:

$[\text{Transset}(i); \text{Transset}(j)] \implies \text{Transset}(i \text{ Un } j)$

$\langle \text{proof} \rangle$

lemma *Transset-Int*:

$[\text{Transset}(i); \text{Transset}(j)] \implies \text{Transset}(i \text{ Int } j)$

$\langle \text{proof} \rangle$

lemma *Transset-succ*: $\text{Transset}(i) \implies \text{Transset}(\text{succ}(i))$

$\langle \text{proof} \rangle$

lemma *Transset-Pow*: $\text{Transset}(i) \implies \text{Transset}(\text{Pow}(i))$

$\langle \text{proof} \rangle$

lemma *Transset-Union*: $\text{Transset}(A) \implies \text{Transset}(\text{Union}(A))$

$\langle \text{proof} \rangle$

lemma *Transset-Union-family*:

$[\![\forall i. i:A \implies \text{Transset}(i)]\!] \implies \text{Transset}(\text{Union}(A))$

$\langle \text{proof} \rangle$

lemma *Transset-Inter-family*:

$[\![\forall i. i:A \implies \text{Transset}(i)]\!] \implies \text{Transset}(\text{Inter}(A))$

$\langle \text{proof} \rangle$

lemma *Transset-UN*:

$(\forall x. x \in A \implies \text{Transset}(B(x))) \implies \text{Transset}(\bigcup_{x \in A} B(x))$

$\langle \text{proof} \rangle$

lemma *Transset-INT*:

$(\forall x. x \in A \implies \text{Transset}(B(x))) \implies \text{Transset}(\bigcap_{x \in A} B(x))$

$\langle \text{proof} \rangle$

13.2 Lemmas for Ordinals

lemma *OrdI*:

$[\text{Transset}(i); \forall x. x:i \implies \text{Transset}(x)] \implies \text{Ord}(i)$

$\langle \text{proof} \rangle$

lemma *Ord-is-Transset*: $\text{Ord}(i) \implies \text{Transset}(i)$

$\langle \text{proof} \rangle$

lemma *Ord-contains-Transset*:

$[\text{Ord}(i); j:i] \implies \text{Transset}(j)$

$\langle \text{proof} \rangle$

lemma *Ord-in-Ord*: $[[\text{Ord}(i); j:i]] ==> \text{Ord}(j)$
 $\langle \text{proof} \rangle$

lemma *Ord-in-Ord'*: $[[j:i; \text{Ord}(i)]] ==> \text{Ord}(j)$
 $\langle \text{proof} \rangle$

lemmas *Ord-succD* = *Ord-in-Ord* [*OF* - *succI1*]

lemma *Ord-subset-Ord*: $[[\text{Ord}(i); \text{Transset}(j); j \leq i]] ==> \text{Ord}(j)$
 $\langle \text{proof} \rangle$

lemma *OrdmemD*: $[[j:i; \text{Ord}(i)]] ==> j \leq i$
 $\langle \text{proof} \rangle$

lemma *Ord-trans*: $[[i:j; j:k; \text{Ord}(k)]] ==> i:k$
 $\langle \text{proof} \rangle$

lemma *Ord-succ-subsetI*: $[[i:j; \text{Ord}(j)]] ==> \text{succ}(i) \leq j$
 $\langle \text{proof} \rangle$

13.3 The Construction of Ordinals: 0, succ, Union

lemma *Ord-0* [*iff*, *TC*]: $\text{Ord}(0)$
 $\langle \text{proof} \rangle$

lemma *Ord-succ* [*TC*]: $\text{Ord}(i) ==> \text{Ord}(\text{succ}(i))$
 $\langle \text{proof} \rangle$

lemmas *Ord-1* = *Ord-0* [*THEN* *Ord-succ*]

lemma *Ord-succ-iff* [*iff*]: $\text{Ord}(\text{succ}(i)) <-> \text{Ord}(i)$
 $\langle \text{proof} \rangle$

lemma *Ord-Un* [*intro*, *simp*, *TC*]: $[[\text{Ord}(i); \text{Ord}(j)]] ==> \text{Ord}(i \text{ Un } j)$
 $\langle \text{proof} \rangle$

lemma *Ord-Int* [*TC*]: $[[\text{Ord}(i); \text{Ord}(j)]] ==> \text{Ord}(i \text{ Int } j)$
 $\langle \text{proof} \rangle$

lemma *ON-class*: $\sim (ALL i. i:X <-> \text{Ord}(i))$
 $\langle \text{proof} \rangle$

13.4 \leq is 'less Than' for Ordinals

lemma *ltI*: $[[i:j; \text{Ord}(j)]] ==> i < j$

$\langle proof \rangle$

lemma *ltE*:

$[[i < j; [i:j; Ord(i); Ord(j)] ==> P] ==> P$
 $\langle proof \rangle$

lemma *ltD*: $i < j ==> i:j$

$\langle proof \rangle$

lemma *not-lt0* [*simp*]: $\sim i < 0$

$\langle proof \rangle$

lemma *lt-Ord*: $j < i ==> Ord(j)$

$\langle proof \rangle$

lemma *lt-Ord2*: $j < i ==> Ord(i)$

$\langle proof \rangle$

lemmas *le-Ord2* = *lt-Ord2* [*THEN Ord-succD*]

lemmas *lt0E* = *not-lt0* [*THEN notE, elim!*]

lemma *lt-trans*: $[[i < j; j < k] ==> i < k$

$\langle proof \rangle$

lemma *lt-not-sym*: $i < j ==> \sim (j < i)$

$\langle proof \rangle$

lemmas *lt-asy* = *lt-not-sym* [*THEN swap*]

lemma *lt-irrefl* [*elim!*]: $i < i ==> P$

$\langle proof \rangle$

lemma *lt-not-refl*: $\sim i < i$

$\langle proof \rangle$

lemma *le-iff*: $i \leq j \iff i < j \mid (i=j \ \& \ Ord(j))$

$\langle proof \rangle$

lemma *leI*: $i < j ==> i \leq j$

$\langle proof \rangle$

lemma *le-eqI*: $[\mid i=j; \text{Ord}(j) \mid] \implies i \text{ le } j$
 $\langle \text{proof} \rangle$

lemmas *le-refl* = *refl* [THEN *le-eqI*]

lemma *le-refl-iff* [*iff*]: $i \text{ le } i \iff \text{Ord}(i)$
 $\langle \text{proof} \rangle$

lemma *leCI*: $(\sim (i=j \ \& \ \text{Ord}(j))) \implies i < j \implies i \text{ le } j$
 $\langle \text{proof} \rangle$

lemma *leE*:
 $[\mid i \text{ le } j; \ i < j \implies P; \mid i=j; \ \text{Ord}(j) \mid] \implies P \mid] \implies P$
 $\langle \text{proof} \rangle$

lemma *le-anti-sym*: $[\mid i \text{ le } j; \ j \text{ le } i \mid] \implies i=j$
 $\langle \text{proof} \rangle$

lemma *le0-iff* [*simp*]: $i \text{ le } 0 \iff i=0$
 $\langle \text{proof} \rangle$

lemmas *le0D* = *le0-iff* [THEN *iffD1*, *dest!*]

13.5 Natural Deduction Rules for Memrel

lemma *Memrel-iff* [*simp*]: $\langle a, b \rangle : \text{Memrel}(A) \iff a:b \ \& \ a:A \ \& \ b:A$
 $\langle \text{proof} \rangle$

lemma *MemrelI* [*intro!*]: $[\mid a: b; \ a: A; \ b: A \mid] \implies \langle a, b \rangle : \text{Memrel}(A)$
 $\langle \text{proof} \rangle$

lemma *MemrelE* [*elim!*]:
 $[\mid \langle a, b \rangle : \text{Memrel}(A);$
 $\quad [\mid a: A; \ b: A; \ a:b \mid] \implies P \mid]$
 $\implies P$
 $\langle \text{proof} \rangle$

lemma *Memrel-type*: $\text{Memrel}(A) \leq A * A$
 $\langle \text{proof} \rangle$

lemma *Memrel-mono*: $A \leq B \implies \text{Memrel}(A) \leq \text{Memrel}(B)$
 $\langle \text{proof} \rangle$

lemma *Memrel-0* [*simp*]: $\text{Memrel}(0) = 0$
 $\langle \text{proof} \rangle$

lemma *Memrel-1* [*simp*]: $\text{Memrel}(1) = 0$
 $\langle \text{proof} \rangle$

lemma *relation-Memrel*: $\text{relation}(\text{Memrel}(A))$
 $\langle \text{proof} \rangle$

lemma *wf-Memrel*: $\text{wf}(\text{Memrel}(A))$
 $\langle \text{proof} \rangle$

The premise $\text{Ord}(i)$ does not suffice.

lemma *trans-Memrel*:
 $\text{Ord}(i) \implies \text{trans}(\text{Memrel}(i))$
 $\langle \text{proof} \rangle$

However, the following premise is strong enough.

lemma *Transset-trans-Memrel*:
 $\forall j \in i. \text{Transset}(j) \implies \text{trans}(\text{Memrel}(i))$
 $\langle \text{proof} \rangle$

lemma *Transset-Memrel-iff*:
 $\text{Transset}(A) \implies \langle a, b \rangle : \text{Memrel}(A) \iff a : b \ \& \ b : A$
 $\langle \text{proof} \rangle$

13.6 Transfinite Induction

lemma *Transset-induct*:
 $\llbracket i : k; \text{Transset}(k);$
 $\quad \text{!!}x. \llbracket x : k; \text{ALL } y : x. P(y) \rrbracket \implies P(x) \rrbracket$
 $\implies P(i)$
 $\langle \text{proof} \rangle$

lemmas *Ord-induct* [consumes 2] = *Transset-induct* [OF - Ord-is-Transset]
lemmas *Ord-induct-rule* = *Ord-induct* [rule-format, consumes 2]

lemma *trans-induct* [consumes 1]:
 $\llbracket \text{Ord}(i);$
 $\quad \text{!!}x. \llbracket \text{Ord}(x); \text{ALL } y : x. P(y) \rrbracket \implies P(x) \rrbracket$
 $\implies P(i)$
 $\langle \text{proof} \rangle$

lemmas *trans-induct-rule* = *trans-induct* [rule-format, consumes 1]

13.6.1 Proving That ; is a Linear Ordering on the Ordinals

lemma *Ord-linear* [rule-format]:
 $\text{Ord}(i) \implies (\text{ALL } j. \text{Ord}(j) \dashv\vdash i : j \mid i = j \mid j : i)$
 $\langle \text{proof} \rangle$

lemma *Ord-linear-lt*:

$\llbracket \text{Ord}(i); \text{Ord}(j); i < j \implies P; i = j \implies P; j < i \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *Ord-linear2*:

$\llbracket \text{Ord}(i); \text{Ord}(j); i < j \implies P; j \text{ le } i \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *Ord-linear-le*:

$\llbracket \text{Ord}(i); \text{Ord}(j); i \text{ le } j \implies P; j \text{ le } i \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *le-imp-not-lt*: $j \text{ le } i \implies \sim i < j$

$\langle \text{proof} \rangle$

lemma *not-lt-imp-le*: $\llbracket \sim i < j; \text{Ord}(i); \text{Ord}(j) \rrbracket \implies j \text{ le } i$

$\langle \text{proof} \rangle$

13.6.2 Some Rewrite Rules for \mathfrak{j} , le

lemma *Ord-mem-iff-lt*: $\text{Ord}(j) \implies i:j < \rightarrow i < j$

$\langle \text{proof} \rangle$

lemma *not-lt-iff-le*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \sim i < j < \rightarrow j \text{ le } i$

$\langle \text{proof} \rangle$

lemma *not-le-iff-lt*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \sim i \text{ le } j < \rightarrow j < i$

$\langle \text{proof} \rangle$

lemma *Ord-0-le*: $\text{Ord}(i) \implies 0 \text{ le } i$

$\langle \text{proof} \rangle$

lemma *Ord-0-lt*: $\llbracket \text{Ord}(i); i \sim 0 \rrbracket \implies 0 < i$

$\langle \text{proof} \rangle$

lemma *Ord-0-lt-iff*: $\text{Ord}(i) \implies i \sim 0 < \rightarrow 0 < i$

$\langle \text{proof} \rangle$

13.7 Results about Less-Than or Equals

lemma *zero-le-succ-iff* [*iff*]: $0 \text{ le } \text{succ}(x) < \rightarrow \text{Ord}(x)$

$\langle \text{proof} \rangle$

lemma *subset-imp-le*: $\llbracket j \leq i; \text{Ord}(i); \text{Ord}(j) \rrbracket \implies j \text{ le } i$

$\langle \text{proof} \rangle$

lemma *le-imp-subset*: $i \text{ le } j \implies i \leq j$

$\langle proof \rangle$

lemma *le-subset-iff*: $j \text{ le } i \iff j \leq i \ \& \ \text{Ord}(i) \ \& \ \text{Ord}(j)$
 $\langle proof \rangle$

lemma *le-succ-iff*: $i \text{ le } \text{succ}(j) \iff i \text{ le } j \mid i = \text{succ}(j) \ \& \ \text{Ord}(i)$
 $\langle proof \rangle$

lemma *all-lt-imp-le*: $[\mid \text{Ord}(i); \text{Ord}(j); \forall x. x < j \implies x < i] \implies j \text{ le } i$
 $\langle proof \rangle$

13.7.1 Transitivity Laws

lemma *lt-trans1*: $[\mid i \text{ le } j; j < k] \implies i < k$
 $\langle proof \rangle$

lemma *lt-trans2*: $[\mid i < j; j \text{ le } k] \implies i < k$
 $\langle proof \rangle$

lemma *le-trans*: $[\mid i \text{ le } j; j \text{ le } k] \implies i \text{ le } k$
 $\langle proof \rangle$

lemma *succ-leI*: $i < j \implies \text{succ}(i) \text{ le } j$
 $\langle proof \rangle$

lemma *succ-leE*: $\text{succ}(i) \text{ le } j \implies i < j$
 $\langle proof \rangle$

lemma *succ-le-iff* [*iff*]: $\text{succ}(i) \text{ le } j \iff i < j$
 $\langle proof \rangle$

lemma *succ-le-imp-le*: $\text{succ}(i) \text{ le } \text{succ}(j) \implies i \text{ le } j$
 $\langle proof \rangle$

lemma *lt-subset-trans*: $[\mid i \leq j; j < k; \text{Ord}(i)] \implies i < k$
 $\langle proof \rangle$

lemma *lt-imp-0-lt*: $j < i \implies 0 < i$
 $\langle proof \rangle$

lemma *succ-lt-iff*: $\text{succ}(i) < j \iff i < j \ \& \ \text{succ}(i) \neq j$
 $\langle proof \rangle$

lemma *Ord-succ-mem-iff*: $\text{Ord}(j) \implies \text{succ}(i) \in \text{succ}(j) \iff i \in j$
 $\langle proof \rangle$

13.7.2 Union and Intersection

lemma *Un-upper1-le*: $[\text{Ord}(i); \text{Ord}(j)] \implies i \text{ le } i \text{ Un } j$
 $\langle \text{proof} \rangle$

lemma *Un-upper2-le*: $[\text{Ord}(i); \text{Ord}(j)] \implies j \text{ le } i \text{ Un } j$
 $\langle \text{proof} \rangle$

lemma *Un-least-lt*: $[i < k; j < k] \implies i \text{ Un } j < k$
 $\langle \text{proof} \rangle$

lemma *Un-least-lt-iff*: $[\text{Ord}(i); \text{Ord}(j)] \implies i \text{ Un } j < k \iff i < k \ \& \ j < k$
 $\langle \text{proof} \rangle$

lemma *Un-least-mem-iff*:
 $[\text{Ord}(i); \text{Ord}(j); \text{Ord}(k)] \implies i \text{ Un } j : k \iff i:k \ \& \ j:k$
 $\langle \text{proof} \rangle$

lemma *Int-greatest-lt*: $[i < k; j < k] \implies i \text{ Int } j < k$
 $\langle \text{proof} \rangle$

lemma *Ord-Un-if*:
 $[\text{Ord}(i); \text{Ord}(j)] \implies i \cup j = (\text{if } j < i \text{ then } i \text{ else } j)$
 $\langle \text{proof} \rangle$

lemma *succ-Un-distrib*:
 $[\text{Ord}(i); \text{Ord}(j)] \implies \text{succ}(i \cup j) = \text{succ}(i) \cup \text{succ}(j)$
 $\langle \text{proof} \rangle$

lemma *lt-Un-iff*:
 $[\text{Ord}(i); \text{Ord}(j)] \implies k < i \cup j \iff k < i \mid k < j$
 $\langle \text{proof} \rangle$

lemma *le-Un-iff*:
 $[\text{Ord}(i); \text{Ord}(j)] \implies k \leq i \cup j \iff k \leq i \mid k \leq j$
 $\langle \text{proof} \rangle$

lemma *Un-upper1-lt*: $[k < i; \text{Ord}(j)] \implies k < i \text{ Un } j$
 $\langle \text{proof} \rangle$

lemma *Un-upper2-lt*: $[k < j; \text{Ord}(i)] \implies k < i \text{ Un } j$
 $\langle \text{proof} \rangle$

lemma *Ord-Union-succ-eq*: $\text{Ord}(i) \implies \bigcup(\text{succ}(i)) = i$
 $\langle \text{proof} \rangle$

13.8 Results about Limits

lemma *Ord-Union* [intro,simp,TC]: $[\![\! \! i. i:A ==> \text{Ord}(i) \! \!]\!] ==> \text{Ord}(\text{Union}(A))$
 $\langle \text{proof} \rangle$

lemma *Ord-UN* [intro,simp,TC]:
 $[\![\! \! x. x:A ==> \text{Ord}(B(x)) \! \!]\!] ==> \text{Ord}(\bigcup_{x \in A} B(x))$
 $\langle \text{proof} \rangle$

lemma *Ord-Inter* [intro,simp,TC]:
 $[\![\! \! i. i:A ==> \text{Ord}(i) \! \!]\!] ==> \text{Ord}(\text{Inter}(A))$
 $\langle \text{proof} \rangle$

lemma *Ord-INT* [intro,simp,TC]:
 $[\![\! \! x. x:A ==> \text{Ord}(B(x)) \! \!]\!] ==> \text{Ord}(\bigcap_{x \in A} B(x))$
 $\langle \text{proof} \rangle$

lemma *UN-least-le*:
 $[\![\text{Ord}(i); \! \! x. x:A ==> b(x) \text{ le } i \! \!]\!] ==> (\bigcup_{x \in A} b(x)) \text{ le } i$
 $\langle \text{proof} \rangle$

lemma *UN-succ-least-lt*:
 $[\![j < i; \! \! x. x:A ==> b(x) < j \! \!]\!] ==> (\bigcup_{x \in A} \text{succ}(b(x))) < i$
 $\langle \text{proof} \rangle$

lemma *UN-upper-lt*:
 $[\![a \in A; i < b(a); \text{Ord}(\bigcup_{x \in A} b(x)) \! \!]\!] ==> i < (\bigcup_{x \in A} b(x))$
 $\langle \text{proof} \rangle$

lemma *UN-upper-le*:
 $[\![a: A; i \text{ le } b(a); \text{Ord}(\bigcup_{x \in A} b(x)) \! \!]\!] ==> i \text{ le } (\bigcup_{x \in A} b(x))$
 $\langle \text{proof} \rangle$

lemma *lt-Union-iff*: $\forall i \in A. \text{Ord}(i) ==> (j < \bigcup(A)) <-> (\exists i \in A. j < i)$
 $\langle \text{proof} \rangle$

lemma *Union-upper-le*:
 $[\![j: J; i \leq j; \text{Ord}(\bigcup(J)) \! \!]\!] ==> i \leq \bigcup J$
 $\langle \text{proof} \rangle$

lemma *le-implies-UN-le-UN*:
 $[\![\! \! x. x:A ==> c(x) \text{ le } d(x) \! \!]\!] ==> (\bigcup_{x \in A} c(x)) \text{ le } (\bigcup_{x \in A} d(x))$
 $\langle \text{proof} \rangle$

lemma *Ord-equality*: $\text{Ord}(i) ==> (\bigcup_{y \in i} \text{succ}(y)) = i$
 $\langle \text{proof} \rangle$

lemma *Ord-Union-subset*: $\text{Ord}(i) \implies \text{Union}(i) \leq i$
 $\langle \text{proof} \rangle$

13.9 Limit Ordinals – General Properties

lemma *Limit-Union-eq*: $\text{Limit}(i) \implies \text{Union}(i) = i$
 $\langle \text{proof} \rangle$

lemma *Limit-is-Ord*: $\text{Limit}(i) \implies \text{Ord}(i)$
 $\langle \text{proof} \rangle$

lemma *Limit-has-0*: $\text{Limit}(i) \implies 0 < i$
 $\langle \text{proof} \rangle$

lemma *Limit-nonzero*: $\text{Limit}(i) \implies i \sim 0$
 $\langle \text{proof} \rangle$

lemma *Limit-has-succ*: $[\text{Limit}(i); j < i] \implies \text{succ}(j) < i$
 $\langle \text{proof} \rangle$

lemma *Limit-succ-lt-iff* [simp]: $\text{Limit}(i) \implies \text{succ}(j) < i \iff (j < i)$
 $\langle \text{proof} \rangle$

lemma *zero-not-Limit* [iff]: $\sim \text{Limit}(0)$
 $\langle \text{proof} \rangle$

lemma *Limit-has-1*: $\text{Limit}(i) \implies 1 < i$
 $\langle \text{proof} \rangle$

lemma *increasing-LimitI*: $[\text{Limit}(l); 0 < l; \forall x \in l. \exists y \in l. x < y] \implies \text{Limit}(l)$
 $\langle \text{proof} \rangle$

lemma *non-succ-LimitI*:
 $[\text{Limit}(i); 0 < i; \forall y. \text{succ}(y) \sim i] \implies \text{Limit}(i)$
 $\langle \text{proof} \rangle$

lemma *succ-LimitE* [elim!]: $\text{Limit}(\text{succ}(i)) \implies P$
 $\langle \text{proof} \rangle$

lemma *not-succ-Limit* [simp]: $\sim \text{Limit}(\text{succ}(i))$
 $\langle \text{proof} \rangle$

lemma *Limit-le-succD*: $[\text{Limit}(i); i \leq \text{succ}(j)] \implies i \leq j$
 $\langle \text{proof} \rangle$

13.9.1 Traditional 3-Way Case Analysis on Ordinals

lemma *Ord-cases-disj*: $\text{Ord}(i) \implies i = 0 \mid (\exists x. \text{Ord}(x) \ \& \ i = \text{succ}(x)) \mid \text{Limit}(i)$
 $\langle \text{proof} \rangle$

lemma *Ord-cases*:

```

  [| Ord(i);
    i=0 ==> P;
    !!j. [| Ord(j); i=succ(j) |] ==> P;
    Limit(i) ==> P
  |] ==> P
<proof>

```

lemma *trans-induct3* [*case-names 0 succ limit, consumes 1*]:

```

  [| Ord(i);
    P(0);
    !!x. [| Ord(x); P(x) |] ==> P(succ(x));
    !!x. [| Limit(x); ALL y:x. P(y) |] ==> P(x)
  |] ==> P(i)
<proof>

```

lemmas *trans-induct3-rule* = *trans-induct3* [*rule-format, case-names 0 succ limit, consumes 1*]

A set of ordinals is either empty, contains its own union, or its union is a limit ordinal.

lemma *Ord-set-cases*:

```

  ∀ i ∈ I. Ord(i) ==> I=0 ∨ ⋃(I) ∈ I ∨ (⋃(I) ∉ I ∧ Limit(⋃(I)))
<proof>

```

If the union of a set of ordinals is a successor, then it is an element of that set.

lemma *Ord-Union-eq-succD*: [|∀ x ∈ X. Ord(x); ⋃ X = succ(j)|] ==> succ(j) ∈ X
 <proof>

lemma *Limit-Union* [*rule-format*]: [| I ≠ 0; ∀ i ∈ I. Limit(i) |] ==> Limit(⋃ I)
 <proof>

end

14 OrdQuant: Special quantifiers

theory *OrdQuant* **imports** *Ordinal* **begin**

14.1 Quantifiers and union operator for ordinals

definition

```

oall :: [i, i => o] => o where
  oall(A, P) == ALL x. x < A --> P(x)

```

definition

$oex :: [i, i => o] => o$ **where**
 $oex(A, P) == EX\ x. x < A \ \& \ P(x)$

definition

$OUnion :: [i, i => i] => i$ **where**
 $OUnion(i, B) == \{z: \bigcup_{x \in i}. B(x). Ord(i)\}$

syntax

$-oall \quad :: [idt, i, o] => o \quad ((\exists ALL \text{ -<./ -})\ 10)$
 $-oex \quad :: [idt, i, o] => o \quad ((\exists EX \text{ -<./ -})\ 10)$
 $-OUNION \quad :: [idt, i, i] => i \quad ((\exists UN \text{ -<./ -})\ 10)$

translations

$ALL\ x < a. P == CONST\ oall(a, \%x. P)$
 $EX\ x < a. P == CONST\ oex(a, \%x. P)$
 $UN\ x < a. B == CONST\ OUnion(a, \%x. B)$

syntax (*xsymbols*)

$-oall \quad :: [idt, i, o] => o \quad ((\exists \forall \text{ -<./ -})\ 10)$
 $-oex \quad :: [idt, i, o] => o \quad ((\exists \exists \text{ -<./ -})\ 10)$
 $-OUNION \quad :: [idt, i, i] => i \quad ((\exists \bigcup \text{ -<./ -})\ 10)$

syntax (*HTML output*)

$-oall \quad :: [idt, i, o] => o \quad ((\exists \forall \text{ -<./ -})\ 10)$
 $-oex \quad :: [idt, i, o] => o \quad ((\exists \exists \text{ -<./ -})\ 10)$
 $-OUNION \quad :: [idt, i, i] => i \quad ((\exists \bigcup \text{ -<./ -})\ 10)$

14.1.1 simplification of the new quantifiers

lemma [*simp*]: $(ALL\ x < 0. P(x))$
 $\langle proof \rangle$

lemma [*simp*]: $\sim (EX\ x < 0. P(x))$
 $\langle proof \rangle$

lemma [*simp*]: $(ALL\ x < succ(i). P(x)) <-> (Ord(i) \dashv\rightarrow P(i) \ \& \ (ALL\ x < i. P(x)))$
 $\langle proof \rangle$

lemma [*simp*]: $(EX\ x < succ(i). P(x)) <-> (Ord(i) \ \& \ (P(i) \mid (EX\ x < i. P(x))))$
 $\langle proof \rangle$

14.1.2 Union over ordinals

lemma *Ord-OUN* [*intro, simp*]:
 $[\mid !!x. x < A ==> Ord(B(x)) \mid] ==> Ord(\bigcup_{x < A}. B(x))$
 $\langle proof \rangle$

lemma *OUN-upper-lt*:

$$[\mid a < A; \ i < b(a); \text{Ord}(\bigcup x < A. b(x)) \mid] \implies i < (\bigcup x < A. b(x))$$

 $\langle \text{proof} \rangle$

lemma *OUN-upper-le*:

$$[\mid a < A; \ i \leq b(a); \text{Ord}(\bigcup x < A. b(x)) \mid] \implies i \leq (\bigcup x < A. b(x))$$

 $\langle \text{proof} \rangle$

lemma *Limit-OUN-eq*: $\text{Limit}(i) \implies (\bigcup x < i. x) = i$
 $\langle \text{proof} \rangle$

lemma *OUN-least*:

$$(\! \mid x. x < A \implies B(x) \subseteq C \implies (\bigcup x < A. B(x)) \subseteq C$$

 $\langle \text{proof} \rangle$

lemma *OUN-least-le*:

$$[\mid \text{Ord}(i); \ !\mid x. x < A \implies b(x) \leq i \mid] \implies (\bigcup x < A. b(x)) \leq i$$

 $\langle \text{proof} \rangle$

lemma *le-implies-OUN-le-OUN*:

$$[\mid !\mid x. x < A \implies c(x) \leq d(x) \mid] \implies (\bigcup x < A. c(x)) \leq (\bigcup x < A. d(x))$$

 $\langle \text{proof} \rangle$

lemma *OUN-UN-eq*:

$$(\! \mid x. x:A \implies \text{Ord}(B(x)) \implies (\bigcup z < (\bigcup x \in A. B(x)). C(z)) = (\bigcup x \in A. \bigcup z < B(x). C(z))$$

 $\langle \text{proof} \rangle$

lemma *OUN-Union-eq*:

$$(\! \mid x. x:X \implies \text{Ord}(x) \implies (\bigcup z < \text{Union}(X). C(z)) = (\bigcup x \in X. \bigcup z < x. C(z))$$

 $\langle \text{proof} \rangle$

lemma *atomize-oall* [*symmetric, rulify*]:

$$(\! \mid x. x < A \implies P(x) \implies \text{Trueprop} (\text{ALL } x < A. P(x))$$

 $\langle \text{proof} \rangle$

14.1.3 universal quantifier for ordinals

lemma *oallI* [*intro!*]:

$$[\mid !\mid x. x < A \implies P(x) \mid] \implies \text{ALL } x < A. P(x)$$

 $\langle \text{proof} \rangle$

lemma *ospec*: $[\mid \text{ALL } x < A. P(x); \ x < A \mid] \implies P(x)$
 $\langle \text{proof} \rangle$

lemma *oallE*:

$\llbracket \text{ALL } x < A. P(x); P(x) \implies Q; \sim x < A \implies Q \rrbracket \implies Q$
 $\langle \text{proof} \rangle$

lemma *rev-oallE* [*elim*]:

$\llbracket \text{ALL } x < A. P(x); \sim x < A \implies Q; P(x) \implies Q \rrbracket \implies Q$
 $\langle \text{proof} \rangle$

lemma *oall-simp* [*simp*]: $(\text{ALL } x < a. \text{True}) <-> \text{True}$
 $\langle \text{proof} \rangle$

lemma *oall-cong* [*cong*]:

$\llbracket a = a'; \llbracket \text{!!}x. x < a' \implies P(x) <-> P'(x) \rrbracket \implies \text{oall}(a, \%x. P(x)) <-> \text{oall}(a', \%x. P'(x))$
 $\langle \text{proof} \rangle$

14.1.4 existential quantifier for ordinals

lemma *oexI* [*intro*]:

$\llbracket P(x); x < A \rrbracket \implies \text{EX } x < A. P(x)$
 $\langle \text{proof} \rangle$

lemma *oexCI*:

$\llbracket \text{ALL } x < A. \sim P(x) \implies P(a); a < A \rrbracket \implies \text{EX } x < A. P(x)$
 $\langle \text{proof} \rangle$

lemma *oexE* [*elim!*]:

$\llbracket \text{EX } x < A. P(x); \llbracket \text{!!}x. \llbracket x < A; P(x) \rrbracket \implies Q \rrbracket \implies Q$
 $\langle \text{proof} \rangle$

lemma *oex-cong* [*cong*]:

$\llbracket a = a'; \llbracket \text{!!}x. x < a' \implies P(x) <-> P'(x) \rrbracket \implies \text{oex}(a, \%x. P(x)) <-> \text{oex}(a', \%x. P'(x))$
 $\langle \text{proof} \rangle$

14.1.5 Rules for Ordinal-Indexed Unions

lemma *OUN-I* [*intro*]: $\llbracket a < i; b : B(a) \rrbracket \implies b : (\bigcup z < i. B(z))$
 $\langle \text{proof} \rangle$

lemma *OUN-E* [*elim!*]:

$\llbracket b : (\bigcup z < i. B(z)); \llbracket \text{!!}a. \llbracket b : B(a); a < i \rrbracket \implies R \rrbracket \implies R$
 $\langle \text{proof} \rangle$

lemma *OUN-iff*: $b : (\bigcup x < i. B(x)) <-> (\text{EX } x < i. b : B(x))$
 $\langle \text{proof} \rangle$

lemma *OUN-cong* [*cong*]:

$\llbracket i=j; \text{!!}x. x<j \implies C(x)=D(x) \rrbracket \implies (\bigcup x<i. C(x)) = (\bigcup x<j. D(x))$
 $\langle \text{proof} \rangle$

lemma *lt-induct*:

$\llbracket i<k; \text{!!}x. \llbracket x<k; \text{ALL } y<x. P(y) \rrbracket \implies P(x) \rrbracket \implies P(i)$
 $\langle \text{proof} \rangle$

14.2 Quantification over a class

definition

rall $:: [i=>o, i=>o] => o$ **where**
rall(*M*, *P*) == *ALL* *x*. *M*(*x*) \longrightarrow *P*(*x*)

definition

rex $:: [i=>o, i=>o] => o$ **where**
rex(*M*, *P*) == *EX* *x*. *M*(*x*) & *P*(*x*)

syntax

-rall $:: [pttrn, i=>o, o] => o$ $((\exists \text{ALL } [-]. / -) 10)$
-rex $:: [pttrn, i=>o, o] => o$ $((\exists \text{EX } [-]. / -) 10)$

syntax (*xsymbols*)

-rall $:: [pttrn, i=>o, o] => o$ $((\exists \forall [-]. / -) 10)$
-rex $:: [pttrn, i=>o, o] => o$ $((\exists \exists [-]. / -) 10)$

syntax (*HTML output*)

-rall $:: [pttrn, i=>o, o] => o$ $((\exists \forall [-]. / -) 10)$
-rex $:: [pttrn, i=>o, o] => o$ $((\exists \exists [-]. / -) 10)$

translations

ALL *x*[*M*]. *P* == *CONST* *rall*(*M*, %*x*. *P*)
EX *x*[*M*]. *P* == *CONST* *rex*(*M*, %*x*. *P*)

14.2.1 Relativized universal quantifier

lemma *rallI* [*intro!*]: $\llbracket \text{!!}x. M(x) \implies P(x) \rrbracket \implies \text{ALL } x[M]. P(x)$
 $\langle \text{proof} \rangle$

lemma *rspec*: $\llbracket \text{ALL } x[M]. P(x); M(x) \rrbracket \implies P(x)$
 $\langle \text{proof} \rangle$

lemma *rev-rallE* [*elim*]:

$\llbracket \text{ALL } x[M]. P(x); \sim M(x) \implies Q; P(x) \implies Q \rrbracket \implies Q$
 $\langle \text{proof} \rangle$

lemma *rallE*: $\llbracket \text{ALL } x[M]. P(x); P(x) \implies Q; \sim M(x) \implies Q \rrbracket \implies Q$
 $\langle \text{proof} \rangle$

lemma *rall-triv* [*simp*]: $(ALL\ x[M].\ P) <-> ((EX\ x.\ M(x)) \dashv\vdash P)$
 $\langle proof \rangle$

lemma *rall-cong* [*cong*]:
 $(!!x.\ M(x) \implies P(x) <-> P'(x)) \implies (ALL\ x[M].\ P(x)) <-> (ALL\ x[M].\ P'(x))$
 $\langle proof \rangle$

14.2.2 Relativized existential quantifier

lemma *rexI* [*intro*]: $[| P(x); M(x) |] \implies EX\ x[M].\ P(x)$
 $\langle proof \rangle$

lemma *rev-rexI*: $[| M(x); P(x) |] \implies EX\ x[M].\ P(x)$
 $\langle proof \rangle$

lemma *rexCI*: $[| ALL\ x[M].\ \sim P(x) \implies P(a); M(a) |] \implies EX\ x[M].\ P(x)$
 $\langle proof \rangle$

lemma *rexE* [*elim!*]: $[| EX\ x[M].\ P(x); !!x.\ [| M(x); P(x) |] \implies Q |] \implies Q$
 $\langle proof \rangle$

lemma *rex-triv* [*simp*]: $(EX\ x[M].\ P) <-> ((EX\ x.\ M(x)) \& P)$
 $\langle proof \rangle$

lemma *rex-cong* [*cong*]:
 $(!!x.\ M(x) \implies P(x) <-> P'(x)) \implies (EX\ x[M].\ P(x)) <-> (EX\ x[M].\ P'(x))$
 $\langle proof \rangle$

lemma *rall-is-ball* [*simp*]: $(\forall x[\%z.\ z \in A].\ P(x)) <-> (\forall x \in A.\ P(x))$
 $\langle proof \rangle$

lemma *rex-is-bex* [*simp*]: $(\exists x[\%z.\ z \in A].\ P(x)) <-> (\exists x \in A.\ P(x))$
 $\langle proof \rangle$

lemma *atomize-rall*: $(!!x.\ M(x) \implies P(x)) == Trueprop\ (ALL\ x[M].\ P(x))$
 $\langle proof \rangle$

declare *atomize-rall* [*symmetric, rulify*]

lemma *rall-simps1*:
 $(ALL\ x[M].\ P(x) \& Q) <-> (ALL\ x[M].\ P(x)) \& ((ALL\ x[M].\ False) | Q)$
 $(ALL\ x[M].\ P(x) | Q) <-> ((ALL\ x[M].\ P(x)) | Q)$
 $(ALL\ x[M].\ P(x) \dashv\vdash Q) <-> ((EX\ x[M].\ P(x)) \dashv\vdash Q)$

$(\sim(ALL\ x[M].\ P(x))) <-> (EX\ x[M].\ \sim P(x))$
 $\langle proof \rangle$

lemma *rall-simps2*:

$(ALL\ x[M].\ P \ \&\ Q(x)) <-> ((ALL\ x[M].\ False) \mid P) \ \&\ (ALL\ x[M].\ Q(x))$
 $(ALL\ x[M].\ P \mid Q(x)) <-> (P \mid (ALL\ x[M].\ Q(x)))$
 $(ALL\ x[M].\ P \dashv\> Q(x)) <-> (P \dashv\> (ALL\ x[M].\ Q(x)))$
 $\langle proof \rangle$

lemmas *rall-simps* [simp] = *rall-simps1* *rall-simps2*

lemma *rall-conj-distrib*:

$(ALL\ x[M].\ P(x) \ \&\ Q(x)) <-> ((ALL\ x[M].\ P(x)) \ \&\ (ALL\ x[M].\ Q(x)))$
 $\langle proof \rangle$

lemma *rex-simps1*:

$(EX\ x[M].\ P(x) \ \&\ Q) <-> ((EX\ x[M].\ P(x)) \ \&\ Q)$
 $(EX\ x[M].\ P(x) \mid Q) <-> (EX\ x[M].\ P(x)) \mid ((EX\ x[M].\ True) \ \&\ Q)$
 $(EX\ x[M].\ P(x) \dashv\> Q) <-> ((ALL\ x[M].\ P(x)) \dashv\> ((EX\ x[M].\ True) \ \&\ Q))$
 $(\sim(EX\ x[M].\ P(x))) <-> (ALL\ x[M].\ \sim P(x))$
 $\langle proof \rangle$

lemma *rex-simps2*:

$(EX\ x[M].\ P \ \&\ Q(x)) <-> (P \ \&\ (EX\ x[M].\ Q(x)))$
 $(EX\ x[M].\ P \mid Q(x)) <-> ((EX\ x[M].\ True) \ \&\ P) \mid (EX\ x[M].\ Q(x))$
 $(EX\ x[M].\ P \dashv\> Q(x)) <-> (((ALL\ x[M].\ False) \mid P) \dashv\> (EX\ x[M].\ Q(x)))$
 $\langle proof \rangle$

lemmas *rex-simps* [simp] = *rex-simps1* *rex-simps2*

lemma *rex-disj-distrib*:

$(EX\ x[M].\ P(x) \mid Q(x)) <-> ((EX\ x[M].\ P(x)) \mid (EX\ x[M].\ Q(x)))$
 $\langle proof \rangle$

14.2.3 One-point rule for bounded quantifiers

lemma *rex-triv-one-point1* [simp]: $(EX\ x[M].\ x=a) <-> (M(a))$
 $\langle proof \rangle$

lemma *rex-triv-one-point2* [simp]: $(EX\ x[M].\ a=x) <-> (M(a))$
 $\langle proof \rangle$

lemma *rex-one-point1* [simp]: $(EX\ x[M].\ x=a \ \&\ P(x)) <-> (M(a) \ \&\ P(a))$
 $\langle proof \rangle$

lemma *rex-one-point2* [simp]: $(EX\ x[M].\ a=x \ \&\ P(x)) <-> (M(a) \ \&\ P(a))$
 $\langle proof \rangle$

lemma *rall-one-point1* [simp]: $(\text{ALL } x[M]. x=a \dashrightarrow P(x)) \dashv\vdash (M(a) \dashrightarrow P(a))$
 $\langle \text{proof} \rangle$

lemma *rall-one-point2* [simp]: $(\text{ALL } x[M]. a=x \dashrightarrow P(x)) \dashv\vdash (M(a) \dashrightarrow P(a))$
 $\langle \text{proof} \rangle$

14.2.4 Sets as Classes

definition

setclass :: $[i,i] \Rightarrow o$ $(\#\# \text{ } [40] \text{ } 40)$ **where**
setclass(*A*) == $\%x. x : A$

lemma *setclass-iff* [simp]: $\text{setclass}(A,x) \dashv\vdash x : A$
 $\langle \text{proof} \rangle$

lemma *rall-setclass-is-ball* [simp]: $(\forall x[\#\#A]. P(x)) \dashv\vdash (\forall x \in A. P(x))$
 $\langle \text{proof} \rangle$

lemma *rex-setclass-is-bex* [simp]: $(\exists x[\#\#A]. P(x)) \dashv\vdash (\exists x \in A. P(x))$
 $\langle \text{proof} \rangle$

$\langle ML \rangle$

Setting up the one-point-rule simproc

$\langle ML \rangle$

end

15 Nat-ZF: The Natural numbers As a Least Fixed Point

theory *Nat-ZF* **imports** *OrdQuant Bool* **begin**

definition

nat :: *i* **where**
nat == $\text{lfp}(\text{Inf}, \%X. \{0\} \text{ Un } \{\text{succ}(i). i:X\})$

definition

quasinat :: *i* \Rightarrow *o* **where**
quasinat(*n*) == $n=0 \mid (\exists m. n = \text{succ}(m))$

definition

nat-case :: $[i, i=>i, i]=>i$ **where**
nat-case(*a*,*b*,*k*) == *THE* *y*. *k*=0 & *y*=*a* | (*EX* *x*. *k*=*succ*(*x*) & *y*=*b*(*x*))

definition

nat-rec :: $[i, i, [i,i]=>i]=>i$ **where**
nat-rec(*k*,*a*,*b*) ==
wfrec(*Memrel*(*nat*), *k*, %*n* *f*. *nat-case*(*a*, %*m*. *b*(*m*, *f*'*m*), *n*))

definition

Le :: *i* **where**
Le == {<*x*,*y*>:*nat***nat*. *x* *le* *y*}

definition

Lt :: *i* **where**
Lt == {<*x*, *y*>:*nat***nat*. *x* < *y*}

definition

Ge :: *i* **where**
Ge == {<*x*,*y*>:*nat***nat*. *y* *le* *x*}

definition

Gt :: *i* **where**
Gt == {<*x*,*y*>:*nat***nat*. *y* < *x*}

definition

greater-than :: $i=>i$ **where**
greater-than(*n*) == {*i*:*nat*. *n* < *i*}

No need for a less-than operator: a natural number is its list of predecessors!

lemma *nat-bnd-mono*: *bnd-mono*(*Inf*, %*X*. {0} *Un* {*succ*(*i*). *i*:*X*})
<proof>

lemmas *nat-unfold* = *nat-bnd-mono* [*THEN* *nat-def* [*THEN* *def-lfp-unfold*], *standard*]

lemma *nat-0I* [*iff*, *TC*]: 0 : *nat*
<proof>

lemma *nat-succI* [*intro!*, *TC*]: *n* : *nat* ==> *succ*(*n*) : *nat*
<proof>

lemma *nat-1I* [*iff*, *TC*]: 1 : *nat*
<proof>

lemma *nat-2I* [*iff*, *TC*]: $2 : \text{nat}$
 $\langle \text{proof} \rangle$

lemma *bool-subset-nat*: $\text{bool} \leq \text{nat}$
 $\langle \text{proof} \rangle$

lemmas *bool-into-nat* = *bool-subset-nat* [*THEN subsetD*, *standard*]

15.1 Injectivity Properties and Induction

lemma *nat-induct* [*case-names 0 succ*, *induct set: nat*]:
 $\llbracket n : \text{nat}; P(0); \forall x. \llbracket x : \text{nat}; P(x) \rrbracket \implies P(\text{succ}(x)) \rrbracket \implies P(n)$
 $\langle \text{proof} \rangle$

lemma *natE*:
 $\llbracket n : \text{nat}; n=0 \implies P; \forall x. \llbracket x : \text{nat}; n=\text{succ}(x) \rrbracket \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *nat-into-Ord* [*simp*]: $n : \text{nat} \implies \text{Ord}(n)$
 $\langle \text{proof} \rangle$

lemmas *nat-0-le* = *nat-into-Ord* [*THEN Ord-0-le*, *standard*]

lemmas *nat-le-refl* = *nat-into-Ord* [*THEN le-refl*, *standard*]

lemma *Ord-nat* [*iff*]: $\text{Ord}(\text{nat})$
 $\langle \text{proof} \rangle$

lemma *Limit-nat* [*iff*]: $\text{Limit}(\text{nat})$
 $\langle \text{proof} \rangle$

lemma *naturals-not-limit*: $a \in \text{nat} \implies \sim \text{Limit}(a)$
 $\langle \text{proof} \rangle$

lemma *succ-natD*: $\text{succ}(i) : \text{nat} \implies i : \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-succ-iff* [*iff*]: $\text{succ}(n) : \text{nat} \iff n : \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-le-Limit*: $\text{Limit}(i) \implies \text{nat le } i$
 $\langle \text{proof} \rangle$

lemmas *succ-in-naturalD* = *Ord-trans* [*OF succI1* - *nat-into-Ord*]

lemma *lt-nat-in-nat*: $\llbracket m < n; n : \text{nat} \rrbracket \implies m : \text{nat}$

$\langle \text{proof} \rangle$

lemma *le-in-nat*: $[[\ m \text{ le } n; \ n:\text{nat} \]] \implies m:\text{nat}$
 $\langle \text{proof} \rangle$

15.2 Variations on Mathematical Induction

lemmas *complete-induct* = *Ord-induct* [*OF* - *Ord-nat*, *case-names less*, *consumes 1*]

lemmas *complete-induct-rule* =
complete-induct [*rule-format*, *case-names less*, *consumes 1*]

lemma *nat-induct-from-lemma* [*rule-format*]:
 $[[\ n:\text{nat}; \ m:\text{nat};$
 $\quad !!x. [[\ x:\text{nat}; \ m \text{ le } x; \ P(x) \]] \implies P(\text{succ}(x)) \]]$
 $\implies m \text{ le } n \dashv\dashv P(m) \dashv\dashv P(n)$
 $\langle \text{proof} \rangle$

lemma *nat-induct-from*:
 $[[\ m \text{ le } n; \ m:\text{nat}; \ n:\text{nat};$
 $\quad P(m);$
 $\quad !!x. [[\ x:\text{nat}; \ m \text{ le } x; \ P(x) \]] \implies P(\text{succ}(x)) \]]$
 $\implies P(n)$
 $\langle \text{proof} \rangle$

lemma *diff-induct* [*case-names 0 0-succ succ-succ*, *consumes 2*]:
 $[[\ m:\text{nat}; \ n:\text{nat};$
 $\quad !!x. x:\text{nat} \implies P(x,0);$
 $\quad !!y. y:\text{nat} \implies P(0,\text{succ}(y));$
 $\quad !!x\ y. [[\ x:\text{nat}; \ y:\text{nat}; \ P(x,y) \]] \implies P(\text{succ}(x),\text{succ}(y)) \]]$
 $\implies P(m,n)$
 $\langle \text{proof} \rangle$

lemma *succ-lt-induct-lemma* [*rule-format*]:
 $m:\text{nat} \implies P(m,\text{succ}(m)) \dashv\dashv (ALL\ x:\text{nat}. P(m,x) \dashv\dashv P(m,\text{succ}(x)))$
 $\dashv\dashv$
 $(ALL\ n:\text{nat}. m < n \dashv\dashv P(m,n))$
 $\langle \text{proof} \rangle$

lemma *succ-lt-induct*:
 $[[\ m < n; \ n:\text{nat};$
 $\quad P(m,\text{succ}(m));$

$$\begin{aligned} & !!x. [\mid x: \text{nat}; \ P(m,x) \mid] ==> P(m, \text{succ}(x)) \mid \\ & ==> P(m,n) \\ & \langle \text{proof} \rangle \end{aligned}$$

15.3 quasinat: to allow a case-split rule for *nat-case*

True if the argument is zero or any successor

lemma *[iff]: quasinat(0)*
 $\langle \text{proof} \rangle$

lemma *[iff]: quasinat(succ(x))*
 $\langle \text{proof} \rangle$

lemma *nat-imp-quasinat: n ∈ nat ==> quasinat(n)*
 $\langle \text{proof} \rangle$

lemma *non-nat-case: ~ quasinat(x) ==> nat-case(a,b,x) = 0*
 $\langle \text{proof} \rangle$

lemma *nat-cases-disj: k=0 | (∃ y. k = succ(y)) | ~ quasinat(k)*
 $\langle \text{proof} \rangle$

lemma *nat-cases:*

$$[\mid k=0 ==> P; \ !y. k = \text{succ}(y) ==> P; \ \sim \text{quasinat}(k) ==> P \mid] ==> P$$
 $\langle \text{proof} \rangle$

lemma *nat-case-0 [simp]: nat-case(a,b,0) = a*
 $\langle \text{proof} \rangle$

lemma *nat-case-succ [simp]: nat-case(a,b,succ(n)) = b(n)*
 $\langle \text{proof} \rangle$

lemma *nat-case-type [TC]:*

$$\begin{aligned} & [\mid n: \text{nat}; \ a: C(0); \ !m. m: \text{nat} ==> b(m): C(\text{succ}(m)) \mid] \\ & ==> \text{nat-case}(a,b,n) : C(n) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *split-nat-case:*

$$\begin{aligned} & P(\text{nat-case}(a,b,k)) <-> \\ & ((k=0 \dashrightarrow P(a)) \ \& \ (\forall x. k=\text{succ}(x) \dashrightarrow P(b(x))) \ \& \ (\sim \text{quasinat}(k) \longrightarrow \\ & P(0))) \\ & \langle \text{proof} \rangle \end{aligned}$$

15.4 Recursion on the Natural Numbers

lemma *nat-rec-0: nat-rec(0,a,b) = a*
 $\langle \text{proof} \rangle$

lemma *nat-rec-succ*: $m: \text{nat} \implies \text{nat-rec}(\text{succ}(m), a, b) = b(m, \text{nat-rec}(m, a, b))$
 $\langle \text{proof} \rangle$

lemma *Un-nat-type* [TC]: $[\mid i: \text{nat}; j: \text{nat} \mid] \implies i \text{ Un } j: \text{nat}$
 $\langle \text{proof} \rangle$

lemma *Int-nat-type* [TC]: $[\mid i: \text{nat}; j: \text{nat} \mid] \implies i \text{ Int } j: \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-nonempty* [simp]: $\text{nat} \sim = 0$
 $\langle \text{proof} \rangle$

A natural number is the set of its predecessors

lemma *nat-eq-Collect-lt*: $i \in \text{nat} \implies \{j \in \text{nat}. j < i\} = i$
 $\langle \text{proof} \rangle$

lemma *Le-iff* [iff]: $\langle x, y \rangle : \text{Le} \iff x \text{ le } y \ \& \ x : \text{nat} \ \& \ y : \text{nat}$
 $\langle \text{proof} \rangle$

end

16 Inductive-ZF: Inductive and Coinductive Definitions

theory *Inductive-ZF*
imports *Fixedpt QPair Nat-ZF*
uses
 (*ind-syntax.ML*)
 (*Tools/cartprod.ML*)
 (*Tools/ind-cases.ML*)
 (*Tools/inductive-package.ML*)
 (*Tools/induct-tacs.ML*)
 (*Tools/primrec-package.ML*)
begin

lemma *def-swap-iff*: $a == b \implies a = c \iff c = b$
 $\langle \text{proof} \rangle$

lemma *def-trans*: $f == g \implies g(a) = b \implies f(a) = b$
 $\langle \text{proof} \rangle$

lemma *refl-thin*: $!!P. a = a \implies P \implies P \ \langle \text{proof} \rangle$

$\langle ML \rangle$

end

17 Epsilon: Epsilon Induction and Recursion

theory *Epsilon* imports *Nat-ZF* begin

definition

$eclose :: i \Rightarrow i$ **where**
 $eclose(A) == \bigcup n \in nat. nat-rec(n, A, \%m r. Union(r))$

definition

$transrec :: [i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $transrec(a, H) == wfrec(Memrel(eclose(\{a\})), a, H)$

definition

$rank :: i \Rightarrow i$ **where**
 $rank(a) == transrec(a, \%x f. \bigcup y \in x. succ(f'y))$

definition

$transrec2 :: [i, i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $transrec2(k, a, b) ==$
 $transrec(k,$
 $\%i r. if(i=0, a,$
 $if(EX j. i=succ(j),$
 $b(THF j. i=succ(j), r'(THF j. i=succ(j))),$
 $\bigcup j < i. r'j)))$

definition

$recursor :: [i, [i, i] \Rightarrow i, i] \Rightarrow i$ **where**
 $recursor(a, b, k) == transrec(k, \%n f. nat-case(a, \%m. b(m, f'm), n))$

definition

$rec :: [i, i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $rec(k, a, b) == recursor(a, b, k)$

17.1 Basic Closure Properties

lemma *arg-subset-eclose*: $A \leq eclose(A)$

$\langle proof \rangle$

lemmas *arg-into-eclose* = *arg-subset-eclose* [*THEN subsetD, standard*]

lemma *Transset-eclose*: $Transset(eclose(A))$

$\langle proof \rangle$

lemmas *eclose-subset* =
Transset-eclose [*unfolded Transset-def*, *THEN bspec*, *standard*]

lemmas *ecloseD* = *eclose-subset* [*THEN subsetD*, *standard*]

lemmas *arg-in-eclose-sing* = *arg-subset-eclose* [*THEN singleton-subsetD*]
lemmas *arg-into-eclose-sing* = *arg-in-eclose-sing* [*THEN ecloseD*, *standard*]

lemmas *eclose-induct* =
Transset-induct [*OF - Transset-eclose*, *induct set: eclose*]

lemma *eps-induct*:

$$[[\text{!!}x. \text{ALL } y:x. P(y) ==> P(x)]] ==> P(a)$$
<proof>

17.2 Leastness of *eclose*

lemma *eclose-least-lemma*:

$$[[\text{Transset}(X); A \leq X; n: \text{nat}]] ==> \text{nat-rec}(n, A, \%m r. \text{Union}(r)) \leq X$$
<proof>

lemma *eclose-least*:

$$[[\text{Transset}(X); A \leq X]] ==> \text{eclose}(A) \leq X$$
<proof>

lemma *eclose-induct-down* [*consumes 1*]:

$$\begin{aligned} &[[a: \text{eclose}(b); \\ &\quad \text{!!}y. [[y: b]] ==> P(y); \\ &\quad \text{!!}y z. [[y: \text{eclose}(b); P(y); z: y]] ==> P(z) \\ &]] ==> P(a) \end{aligned}$$
<proof>

lemma *Transset-eclose-eq-arg*: $\text{Transset}(X) ==> \text{eclose}(X) = X$
<proof>

A transitive set either is empty or contains the empty set.

lemma *Transset-0-lemma* [*rule-format*]: $\text{Transset}(A) ==> x \in A \dashv\dashv 0 \in A$
<proof>

lemma *Transset-0-disj*: $\text{Transset}(A) ==> A = 0 \mid 0 \in A$
<proof>

17.3 Epsilon Recursion

lemma *mem-eclose-trans*: $[[A: \text{eclose}(B); B: \text{eclose}(C)]] ==> A: \text{eclose}(C)$

$\langle proof \rangle$

lemma *mem-eclose-sing-trans*:

$\llbracket A: \text{eclose}(\{B\}); B: \text{eclose}(\{C\}) \rrbracket \implies A: \text{eclose}(\{C\})$
 $\langle proof \rangle$

lemma *under-Memrel*: $\llbracket \text{Transset}(i); j:i \rrbracket \implies \text{Memrel}(i) - \text{“}\{j\} = j$
 $\langle proof \rangle$

lemma *lt-Memrel*: $j < i \implies \text{Memrel}(i) - \text{“}\{j\} = j$
 $\langle proof \rangle$

lemmas *under-Memrel-eclose* = *Transset-eclose* [THEN *under-Memrel*, *standard*]

lemmas *wfrec-ssubst* = *wf-Memrel* [THEN *wfrec*, THEN *ssubst*]

lemma *wfrec-eclose-eq*:

$\llbracket k: \text{eclose}(\{j\}); j: \text{eclose}(\{i\}) \rrbracket \implies$
 $\text{wfrec}(\text{Memrel}(\text{eclose}(\{i\})), k, H) = \text{wfrec}(\text{Memrel}(\text{eclose}(\{j\})), k, H)$
 $\langle proof \rangle$

lemma *wfrec-eclose-eq2*:

$k: i \implies \text{wfrec}(\text{Memrel}(\text{eclose}(\{i\})), k, H) = \text{wfrec}(\text{Memrel}(\text{eclose}(\{k\})), k, H)$
 $\langle proof \rangle$

lemma *transrec*: $\text{transrec}(a, H) = H(a, \text{lam } x:a. \text{transrec}(x, H))$
 $\langle proof \rangle$

lemma *def-transrec*:

$\llbracket !!x. f(x) == \text{transrec}(x, H) \rrbracket \implies f(a) = H(a, \text{lam } x:a. f(x))$
 $\langle proof \rangle$

lemma *transrec-type*:

$\llbracket !!x u. \llbracket x: \text{eclose}(\{a\}); u: \text{Pi}(x, B) \rrbracket \implies H(x, u) : B(x) \rrbracket$
 $\implies \text{transrec}(a, H) : B(a)$
 $\langle proof \rangle$

lemma *eclose-sing-Ord*: $\text{Ord}(i) \implies \text{eclose}(\{i\}) \leq \text{succ}(i)$
 $\langle proof \rangle$

lemma *succ-subset-eclose-sing*: $\text{succ}(i) \leq \text{eclose}(\{i\})$
 $\langle proof \rangle$

lemma *eclose-sing-Ord-eq*: $\text{Ord}(i) \implies \text{eclose}(\{i\}) = \text{succ}(i)$
 $\langle proof \rangle$

lemma *Ord-transrec-type*:
assumes *jini*: $j: i$
and *ordi*: $\text{Ord}(i)$
and *minor*: $\llbracket x: i; \ u: \text{Pi}(x, B) \rrbracket \implies H(x, u) : B(x)$
shows $\text{transrec}(j, H) : B(j)$
 $\langle \text{proof} \rangle$

17.4 Rank

lemma *rank*: $\text{rank}(a) = (\bigcup y \in a. \text{succ}(\text{rank}(y)))$
 $\langle \text{proof} \rangle$

lemma *Ord-rank* [*simp*]: $\text{Ord}(\text{rank}(a))$
 $\langle \text{proof} \rangle$

lemma *rank-of-Ord*: $\text{Ord}(i) \implies \text{rank}(i) = i$
 $\langle \text{proof} \rangle$

lemma *rank-lt*: $a < b \implies \text{rank}(a) < \text{rank}(b)$
 $\langle \text{proof} \rangle$

lemma *eclose-rank-lt*: $a: \text{eclose}(b) \implies \text{rank}(a) < \text{rank}(b)$
 $\langle \text{proof} \rangle$

lemma *rank-mono*: $a \leq b \implies \text{rank}(a) \leq \text{rank}(b)$
 $\langle \text{proof} \rangle$

lemma *rank-Pow*: $\text{rank}(\text{Pow}(a)) = \text{succ}(\text{rank}(a))$
 $\langle \text{proof} \rangle$

lemma *rank-0* [*simp*]: $\text{rank}(0) = 0$
 $\langle \text{proof} \rangle$

lemma *rank-succ* [*simp*]: $\text{rank}(\text{succ}(x)) = \text{succ}(\text{rank}(x))$
 $\langle \text{proof} \rangle$

lemma *rank-Union*: $\text{rank}(\text{Union}(A)) = (\bigcup x \in A. \text{rank}(x))$
 $\langle \text{proof} \rangle$

lemma *rank-eclose*: $\text{rank}(\text{eclose}(a)) = \text{rank}(a)$
 $\langle \text{proof} \rangle$

lemma *rank-pair1*: $\text{rank}(a) < \text{rank}(\langle a, b \rangle)$
 $\langle \text{proof} \rangle$

lemma *rank-pair2*: $\text{rank}(b) < \text{rank}(\langle a, b \rangle)$
 $\langle \text{proof} \rangle$

lemma *the-equality-if*:

$P(a) ==> (THE\ x.\ P(x)) = (if\ (EX!x.\ P(x))\ then\ a\ else\ 0)$
 $\langle proof \rangle$

lemma *rank-apply*: $[|i : domain(f); function(f)|] ==> rank(f^i) < rank(f)$
 $\langle proof \rangle$

17.5 Corollaries of Leastness

lemma *mem-eclose-subset*: $A:B ==> eclose(A) \leq eclose(B)$
 $\langle proof \rangle$

lemma *eclose-mono*: $A \leq B ==> eclose(A) \leq eclose(B)$
 $\langle proof \rangle$

lemma *eclose-idem*: $eclose(eclose(A)) = eclose(A)$
 $\langle proof \rangle$

lemma *transrec2-0* [simp]: $transrec2(0, a, b) = a$
 $\langle proof \rangle$

lemma *transrec2-succ* [simp]: $transrec2(succ(i), a, b) = b(i, transrec2(i, a, b))$
 $\langle proof \rangle$

lemma *transrec2-Limit*:

$Limit(i) ==> transrec2(i, a, b) = (\bigcup j < i. transrec2(j, a, b))$
 $\langle proof \rangle$

lemma *def-transrec2*:

$(!!x. f(x) == transrec2(x, a, b))$
 $==> f(0) = a \ \&$
 $f(succ(i)) = b(i, f(i)) \ \&$
 $(Limit(K) --> f(K) = (\bigcup j < K. f(j)))$
 $\langle proof \rangle$

lemmas *recursor-lemma* = *recursor-def* [THEN *def-transrec*, THEN *trans*]

lemma *recursor-0*: $recursor(a, b, 0) = a$
 $\langle proof \rangle$

lemma *recursor-succ*: $\text{recursor}(a, b, \text{succ}(m)) = b(m, \text{recursor}(a, b, m))$
 $\langle \text{proof} \rangle$

lemma *rec-0* [*simp*]: $\text{rec}(0, a, b) = a$
 $\langle \text{proof} \rangle$

lemma *rec-succ* [*simp*]: $\text{rec}(\text{succ}(m), a, b) = b(m, \text{rec}(m, a, b))$
 $\langle \text{proof} \rangle$

lemma *rec-type*:

$$\begin{aligned} & \llbracket n: \text{nat}; \\ & \quad a: C(0); \\ & \quad !!m\ z. \llbracket m: \text{nat};\ z: C(m) \rrbracket \implies b(m, z): C(\text{succ}(m)) \rrbracket \\ & \implies \text{rec}(n, a, b) : C(n) \end{aligned}$$

 $\langle \text{proof} \rangle$

$\langle \text{ML} \rangle$

end

18 Order: Partial and Total Orderings: Basic Definitions and Properties

theory *Order* **imports** *WF Perm* **begin**

We adopt the following convention: *ord* is used for strict orders and *order* is used for their reflexive counterparts.

definition

$$\begin{aligned} \text{part-ord} &:: [i, i] \Rightarrow o & \textbf{where} \\ \text{part-ord}(A, r) &== \text{irrefl}(A, r) \ \& \ \text{trans}[A](r) \end{aligned}$$

definition

$$\begin{aligned} \text{linear} &:: [i, i] \Rightarrow o & \textbf{where} \\ \text{linear}(A, r) &== (\text{ALL } x:A. \text{ ALL } y:A. \langle x, y \rangle : r \mid x=y \mid \langle y, x \rangle : r) \end{aligned}$$

definition

$$\begin{aligned} \text{tot-ord} &:: [i, i] \Rightarrow o & \textbf{where} \\ \text{tot-ord}(A, r) &== \text{part-ord}(A, r) \ \& \ \text{linear}(A, r) \end{aligned}$$

definition

$$\text{preorder-on}(A, r) \equiv \text{refl}(A, r) \ \& \ \text{trans}[A](r)$$

definition

$$\text{partial-order-on}(A, r) \equiv \text{preorder-on}(A, r) \ \& \ \text{antisym}(r)$$

abbreviation

$$Preorder(r) \equiv preorder-on(field(r), r)$$
abbreviation

$$Partial-order(r) \equiv partial-order-on(field(r), r)$$
definition

$$\begin{aligned} well-ord &:: [i, i] \Rightarrow o & \textbf{where} \\ well-ord(A, r) &== tot-ord(A, r) \ \& \ wf[A](r) \end{aligned}$$
definition

$$\begin{aligned} mono-map &:: [i, i, i, i] \Rightarrow i & \textbf{where} \\ mono-map(A, r, B, s) &== \\ &\{f: A \rightarrow B. \ ALL \ x:A. \ ALL \ y:A. \ \langle x, y \rangle : r \longrightarrow \langle f'x, f'y \rangle : s\} \end{aligned}$$
definition

$$\begin{aligned} ord-iso &:: [i, i, i, i] \Rightarrow i & \textbf{where} \\ ord-iso(A, r, B, s) &== \\ &\{f: bij(A, B). \ ALL \ x:A. \ ALL \ y:A. \ \langle x, y \rangle : r \longleftrightarrow \langle f'x, f'y \rangle : s\} \end{aligned}$$
definition

$$\begin{aligned} pred &:: [i, i, i] \Rightarrow i & \textbf{where} \\ pred(A, x, r) &== \{y:A. \ \langle y, x \rangle : r\} \end{aligned}$$
definition

$$\begin{aligned} ord-iso-map &:: [i, i, i, i] \Rightarrow i & \textbf{where} \\ ord-iso-map(A, r, B, s) &== \\ &\bigcup x \in A. \bigcup y \in B. \bigcup f \in ord-iso(pred(A, x, r), r, pred(B, y, s), s). \{\langle x, y \rangle\} \end{aligned}$$
definition

$$\begin{aligned} first &:: [i, i, i] \Rightarrow o & \textbf{where} \\ first(u, X, R) &== u:X \ \& \ (ALL \ v:X. \ v \sim u \longrightarrow \langle u, v \rangle : R) \end{aligned}$$
notation (*xsymbols*)
$$ord-iso \ ((\langle -, - \rangle \cong / \langle -, - \rangle) \ 51)$$
18.1 Immediate Consequences of the Definitions**lemma** *part-ord-Imp-asym*:
$$part-ord(A, r) \implies asym(r \ Int \ A * A)$$

<proof>

lemma *linearE*:
$$\begin{aligned} &[[\ linear(A, r); \ x:A; \ y:A; \\ &\quad \langle x, y \rangle : r \implies P; \ x=y \implies P; \ \langle y, x \rangle : r \implies P \] \\ &\implies P \end{aligned}$$

<proof>

lemma *well-ordI*:

$\llbracket \text{wf}[A](r); \text{linear}(A,r) \rrbracket \implies \text{well-ord}(A,r)$
 $\langle \text{proof} \rangle$

lemma *well-ord-is-wf*:

$\text{well-ord}(A,r) \implies \text{wf}[A](r)$
 $\langle \text{proof} \rangle$

lemma *well-ord-is-trans-on*:

$\text{well-ord}(A,r) \implies \text{trans}[A](r)$
 $\langle \text{proof} \rangle$

lemma *well-ord-is-linear*: $\text{well-ord}(A,r) \implies \text{linear}(A,r)$

$\langle \text{proof} \rangle$

lemma *pred-iff*: $y : \text{pred}(A,x,r) \iff \langle y,x \rangle : r \ \& \ y:A$

$\langle \text{proof} \rangle$

lemmas *predI* = *conjI* [*THEN pred-iff* [*THEN iffD2*]]

lemma *predE*: $\llbracket y : \text{pred}(A,x,r); \llbracket y:A; \langle y,x \rangle : r \rrbracket \implies P \rrbracket \implies P$

$\langle \text{proof} \rangle$

lemma *pred-subset-under*: $\text{pred}(A,x,r) \leq r - \{x\}$

$\langle \text{proof} \rangle$

lemma *pred-subset*: $\text{pred}(A,x,r) \leq A$

$\langle \text{proof} \rangle$

lemma *pred-pred-eq*:

$\text{pred}(\text{pred}(A,x,r), y, r) = \text{pred}(A,x,r) \text{ Int } \text{pred}(A,y,r)$

$\langle \text{proof} \rangle$

lemma *trans-pred-pred-eq*:

$\llbracket \text{trans}[A](r); \langle y,x \rangle : r; x:A; y:A \rrbracket$
 $\implies \text{pred}(\text{pred}(A,x,r), y, r) = \text{pred}(A,y,r)$

$\langle \text{proof} \rangle$

18.2 Restricting an Ordering's Domain

lemma *part-ord-subset*:

$\llbracket \text{part-ord}(A,r); B \leq A \rrbracket \implies \text{part-ord}(B,r)$

$\langle proof \rangle$

lemma *linear-subset*:

$\llbracket linear(A,r); B \leq A \rrbracket ==> linear(B,r)$
 $\langle proof \rangle$

lemma *tot-ord-subset*:

$\llbracket tot-ord(A,r); B \leq A \rrbracket ==> tot-ord(B,r)$
 $\langle proof \rangle$

lemma *well-ord-subset*:

$\llbracket well-ord(A,r); B \leq A \rrbracket ==> well-ord(B,r)$
 $\langle proof \rangle$

lemma *irrefl-Int-iff*: $irrefl(A,r \text{ Int } A*A) <-> irrefl(A,r)$
 $\langle proof \rangle$

lemma *trans-on-Int-iff*: $trans[A](r \text{ Int } A*A) <-> trans[A](r)$
 $\langle proof \rangle$

lemma *part-ord-Int-iff*: $part-ord(A,r \text{ Int } A*A) <-> part-ord(A,r)$
 $\langle proof \rangle$

lemma *linear-Int-iff*: $linear(A,r \text{ Int } A*A) <-> linear(A,r)$
 $\langle proof \rangle$

lemma *tot-ord-Int-iff*: $tot-ord(A,r \text{ Int } A*A) <-> tot-ord(A,r)$
 $\langle proof \rangle$

lemma *wf-on-Int-iff*: $wf[A](r \text{ Int } A*A) <-> wf[A](r)$
 $\langle proof \rangle$

lemma *well-ord-Int-iff*: $well-ord(A,r \text{ Int } A*A) <-> well-ord(A,r)$
 $\langle proof \rangle$

18.3 Empty and Unit Domains

lemma *wf-on-any-0*: $wf[A](0)$
 $\langle proof \rangle$

18.3.1 Relations over the Empty Set

lemma *irrefl-0*: $irrefl(0,r)$
 $\langle proof \rangle$

lemma *trans-on-0*: $trans[0](r)$
 $\langle proof \rangle$

lemma *part-ord-0*: *part-ord*(0,*r*)
 <proof>

lemma *linear-0*: *linear*(0,*r*)
 <proof>

lemma *tot-ord-0*: *tot-ord*(0,*r*)
 <proof>

lemma *wf-on-0*: *wf*[0](*r*)
 <proof>

lemma *well-ord-0*: *well-ord*(0,*r*)
 <proof>

18.3.2 The Empty Relation Well-Orders the Unit Set

by Grabczewski

lemma *tot-ord-unit*: *tot-ord*({*a*},0)
 <proof>

lemma *well-ord-unit*: *well-ord*({*a*},0)
 <proof>

18.4 Order-Isomorphisms

Suppes calls them "similarities"

lemma *mono-map-is-fun*: *f*: *mono-map*(*A*,*r*,*B*,*s*) ==> *f*: *A* → *B*
 <proof>

lemma *mono-map-is-inj*:
 [| *linear*(*A*,*r*); *wf*[*B*](*s*); *f*: *mono-map*(*A*,*r*,*B*,*s*) |] ==> *f*: *inj*(*A*,*B*)
 <proof>

lemma *ord-isoI*:
 [| *f*: *bij*(*A*, *B*);
 !!*x y*. [| *x*:*A*; *y*:*A* |] ==> <*x*, *y*> : *r* <-> <*f*'*x*, *f*'*y*> : *s* |]
 ==> *f*: *ord-iso*(*A*,*r*,*B*,*s*)
 <proof>

lemma *ord-iso-is-mono-map*:
f: *ord-iso*(*A*,*r*,*B*,*s*) ==> *f*: *mono-map*(*A*,*r*,*B*,*s*)
 <proof>

lemma *ord-iso-is-bij*:
f: *ord-iso*(*A*,*r*,*B*,*s*) ==> *f*: *bij*(*A*,*B*)
 <proof>

lemma *ord-iso-apply*:

$\llbracket f: \text{ord-iso}(A, r, B, s); \langle x, y \rangle: r; x:A; y:A \rrbracket \implies \langle f'x, f'y \rangle: s$
 $\langle \text{proof} \rangle$

lemma *ord-iso-converse*:

$\llbracket f: \text{ord-iso}(A, r, B, s); \langle x, y \rangle: s; x:B; y:B \rrbracket$
 $\implies \langle \text{converse}(f) 'x, \text{converse}(f) 'y \rangle: r$
 $\langle \text{proof} \rangle$

lemma *ord-iso-refl*: $\text{id}(A): \text{ord-iso}(A, r, A, r)$

$\langle \text{proof} \rangle$

lemma *ord-iso-sym*: $f: \text{ord-iso}(A, r, B, s) \implies \text{converse}(f): \text{ord-iso}(B, s, A, r)$

$\langle \text{proof} \rangle$

lemma *mono-map-trans*:

$\llbracket g: \text{mono-map}(A, r, B, s); f: \text{mono-map}(B, s, C, t) \rrbracket$
 $\implies (f \circ g): \text{mono-map}(A, r, C, t)$
 $\langle \text{proof} \rangle$

lemma *ord-iso-trans*:

$\llbracket g: \text{ord-iso}(A, r, B, s); f: \text{ord-iso}(B, s, C, t) \rrbracket$
 $\implies (f \circ g): \text{ord-iso}(A, r, C, t)$
 $\langle \text{proof} \rangle$

lemma *mono-ord-isoI*:

$\llbracket f: \text{mono-map}(A, r, B, s); g: \text{mono-map}(B, s, A, r);$
 $f \circ g = \text{id}(B); g \circ f = \text{id}(A) \rrbracket \implies f: \text{ord-iso}(A, r, B, s)$
 $\langle \text{proof} \rangle$

lemma *well-ord-mono-ord-isoI*:

$\llbracket \text{well-ord}(A, r); \text{well-ord}(B, s);$
 $f: \text{mono-map}(A, r, B, s); \text{converse}(f): \text{mono-map}(B, s, A, r) \rrbracket$
 $\implies f: \text{ord-iso}(A, r, B, s)$
 $\langle \text{proof} \rangle$

lemma *part-ord-ord-iso*:

$\llbracket \text{part-ord}(B,s); f: \text{ord-iso}(A,r,B,s) \rrbracket \implies \text{part-ord}(A,r)$
 $\langle \text{proof} \rangle$

lemma *linear-ord-iso*:

$\llbracket \text{linear}(B,s); f: \text{ord-iso}(A,r,B,s) \rrbracket \implies \text{linear}(A,r)$
 $\langle \text{proof} \rangle$

lemma *wf-on-ord-iso*:

$\llbracket \text{wf}[B](s); f: \text{ord-iso}(A,r,B,s) \rrbracket \implies \text{wf}[A](r)$
 $\langle \text{proof} \rangle$

lemma *well-ord-ord-iso*:

$\llbracket \text{well-ord}(B,s); f: \text{ord-iso}(A,r,B,s) \rrbracket \implies \text{well-ord}(A,r)$
 $\langle \text{proof} \rangle$

18.5 Main results of Kunen, Chapter 1 section 6

lemma *well-ord-iso-subset-lemma*:

$\llbracket \text{well-ord}(A,r); f: \text{ord-iso}(A,r, A',r); A' \leq A; y: A \rrbracket$
 $\implies \sim <f'y, y>: r$
 $\langle \text{proof} \rangle$

lemma *well-ord-iso-predE*:

$\llbracket \text{well-ord}(A,r); f: \text{ord-iso}(A, r, \text{pred}(A,x,r), r); x:A \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *well-ord-iso-pred-eq*:

$\llbracket \text{well-ord}(A,r); f: \text{ord-iso}(\text{pred}(A,a,r), r, \text{pred}(A,c,r), r);$
 $a:A; c:A \rrbracket \implies a=c$
 $\langle \text{proof} \rangle$

lemma *ord-iso-image-pred*:

$\llbracket f: \text{ord-iso}(A,r,B,s); a:A \rrbracket \implies f \text{ `` } \text{pred}(A,a,r) = \text{pred}(B, f'a, s)$
 $\langle \text{proof} \rangle$

lemma *ord-iso-restrict-image*:

$\llbracket f: \text{ord-iso}(A,r,B,s); C \leq A \rrbracket$
 $\implies \text{restrict}(f,C): \text{ord-iso}(C, r, f''C, s)$
 $\langle \text{proof} \rangle$

lemma *ord-iso-restrict-pred*:

$\llbracket f: \text{ord-iso}(A,r,B,s); a:A \rrbracket$
 $\implies \text{restrict}(f, \text{pred}(A,a,r)): \text{ord-iso}(\text{pred}(A,a,r), r, \text{pred}(B, f'a, s), s)$

$\langle \text{proof} \rangle$

lemma *well-ord-iso-preserving*:

$[[\text{well-ord}(A,r); \text{well-ord}(B,s); \langle a,c \rangle : r;$
 $f : \text{ord-iso}(\text{pred}(A,a,r), r, \text{pred}(B,b,s), s);$
 $g : \text{ord-iso}(\text{pred}(A,c,r), r, \text{pred}(B,d,s), s);$
 $a:A; c:A; b:B; d:B]] \implies \langle b,d \rangle : s$

$\langle \text{proof} \rangle$

lemma *well-ord-iso-unique-lemma*:

$[[\text{well-ord}(A,r);$
 $f : \text{ord-iso}(A,r, B,s); g : \text{ord-iso}(A,r, B,s); y : A]]$
 $\implies \sim \langle g'y, f'y \rangle : s$

$\langle \text{proof} \rangle$

lemma *well-ord-iso-unique*: $[[\text{well-ord}(A,r);$

$f : \text{ord-iso}(A,r, B,s); g : \text{ord-iso}(A,r, B,s)]]$ $\implies f = g$

$\langle \text{proof} \rangle$

18.6 Towards Kunen's Theorem 6.3: Linearity of the Similarity Relation

lemma *ord-iso-map-subset*: $\text{ord-iso-map}(A,r,B,s) \leq A * B$

$\langle \text{proof} \rangle$

lemma *domain-ord-iso-map*: $\text{domain}(\text{ord-iso-map}(A,r,B,s)) \leq A$

$\langle \text{proof} \rangle$

lemma *range-ord-iso-map*: $\text{range}(\text{ord-iso-map}(A,r,B,s)) \leq B$

$\langle \text{proof} \rangle$

lemma *converse-ord-iso-map*:

$\text{converse}(\text{ord-iso-map}(A,r,B,s)) = \text{ord-iso-map}(B,s,A,r)$

$\langle \text{proof} \rangle$

lemma *function-ord-iso-map*:

$\text{well-ord}(B,s) \implies \text{function}(\text{ord-iso-map}(A,r,B,s))$

$\langle \text{proof} \rangle$

lemma *ord-iso-map-fun*: $\text{well-ord}(B,s) \implies \text{ord-iso-map}(A,r,B,s)$

$: \text{domain}(\text{ord-iso-map}(A,r,B,s)) \rightarrow \text{range}(\text{ord-iso-map}(A,r,B,s))$

$\langle \text{proof} \rangle$

lemma *ord-iso-map-mono-map*:

$[[\text{well-ord}(A,r); \text{well-ord}(B,s)]]$

$$\begin{aligned} & \implies \text{ord-iso-map}(A, r, B, s) \\ & \quad : \text{mono-map}(\text{domain}(\text{ord-iso-map}(A, r, B, s)), r, \\ & \quad \quad \text{range}(\text{ord-iso-map}(A, r, B, s)), s) \end{aligned}$$
 $\langle \text{proof} \rangle$

lemma *ord-iso-map-ord-iso*:

$$\begin{aligned} & [\text{well-ord}(A, r); \text{well-ord}(B, s)] \implies \text{ord-iso-map}(A, r, B, s) \\ & \quad : \text{ord-iso}(\text{domain}(\text{ord-iso-map}(A, r, B, s)), r, \\ & \quad \quad \text{range}(\text{ord-iso-map}(A, r, B, s)), s) \end{aligned}$$
 $\langle \text{proof} \rangle$

lemma *domain-ord-iso-map-subset*:

$$\begin{aligned} & [\text{well-ord}(A, r); \text{well-ord}(B, s); \\ & \quad a: A; a \sim: \text{domain}(\text{ord-iso-map}(A, r, B, s))] \\ & \implies \text{domain}(\text{ord-iso-map}(A, r, B, s)) \leq \text{pred}(A, a, r) \end{aligned}$$
 $\langle \text{proof} \rangle$

lemma *domain-ord-iso-map-cases*:

$$\begin{aligned} & [\text{well-ord}(A, r); \text{well-ord}(B, s)] \\ & \implies \text{domain}(\text{ord-iso-map}(A, r, B, s)) = A \mid \\ & \quad (\exists x: A. \text{domain}(\text{ord-iso-map}(A, r, B, s)) = \text{pred}(A, x, r)) \end{aligned}$$
 $\langle \text{proof} \rangle$

lemma *range-ord-iso-map-cases*:

$$\begin{aligned} & [\text{well-ord}(A, r); \text{well-ord}(B, s)] \\ & \implies \text{range}(\text{ord-iso-map}(A, r, B, s)) = B \mid \\ & \quad (\exists y: B. \text{range}(\text{ord-iso-map}(A, r, B, s)) = \text{pred}(B, y, s)) \end{aligned}$$
 $\langle \text{proof} \rangle$

Kunen's Theorem 6.3: Fundamental Theorem for Well-Ordered Sets

theorem *well-ord-trichotomy*:

$$\begin{aligned} & [\text{well-ord}(A, r); \text{well-ord}(B, s)] \\ & \implies \text{ord-iso-map}(A, r, B, s) : \text{ord-iso}(A, r, B, s) \mid \\ & \quad (\exists x: A. \text{ord-iso-map}(A, r, B, s) : \text{ord-iso}(\text{pred}(A, x, r), r, B, s)) \mid \\ & \quad (\exists y: B. \text{ord-iso-map}(A, r, B, s) : \text{ord-iso}(A, r, \text{pred}(B, y, s), s)) \end{aligned}$$
 $\langle \text{proof} \rangle$

18.7 Miscellaneous Results by Krzysztof Grabczewski

lemma *irrefl-converse*: $\text{irrefl}(A, r) \implies \text{irrefl}(A, \text{converse}(r))$
 $\langle \text{proof} \rangle$

lemma *trans-on-converse*: $\text{trans}[A](r) \implies \text{trans}[A](\text{converse}(r))$
 $\langle \text{proof} \rangle$

lemma *part-ord-converse*: $\text{part-ord}(A, r) \implies \text{part-ord}(A, \text{converse}(r))$
 $\langle \text{proof} \rangle$

lemma *linear-converse*: $\text{linear}(A, r) \implies \text{linear}(A, \text{converse}(r))$
 $\langle \text{proof} \rangle$

lemma *tot-ord-converse*: $\text{tot-ord}(A, r) \implies \text{tot-ord}(A, \text{converse}(r))$
 $\langle \text{proof} \rangle$

lemma *first-is-elem*: $\text{first}(b, B, r) \implies b : B$
 $\langle \text{proof} \rangle$

lemma *well-ord-imp-ex1-first*:
 $\llbracket \text{well-ord}(A, r); B \leq A; B \sim 0 \rrbracket \implies (\text{EX! } b. \text{first}(b, B, r))$
 $\langle \text{proof} \rangle$

lemma *the-first-in*:
 $\llbracket \text{well-ord}(A, r); B \leq A; B \sim 0 \rrbracket \implies (\text{THE } b. \text{first}(b, B, r)) : B$
 $\langle \text{proof} \rangle$

18.8 Lemmas for the Reflexive Orders

lemma *subset-vimage-vimage-iff*:
 $\llbracket \text{Preorder}(r); A \subseteq \text{field}(r); B \subseteq \text{field}(r) \rrbracket \implies$
 $r - \text{“ } A \subseteq r - \text{“ } B < - > (\text{ALL } a : A. \text{EX } b : B. < a, b > : r)$
 $\langle \text{proof} \rangle$

lemma *subset-vimage1-vimage1-iff*:
 $\llbracket \text{Preorder}(r); a : \text{field}(r); b : \text{field}(r) \rrbracket \implies$
 $r - \text{“ } \{a\} \subseteq r - \text{“ } \{b\} < - > < a, b > : r$
 $\langle \text{proof} \rangle$

lemma *Refl-antisym-eq-Image1-Image1-iff*:
 $\llbracket \text{refl}(\text{field}(r), r); \text{antisym}(r); a : \text{field}(r); b : \text{field}(r) \rrbracket \implies$
 $r - \text{“ } \{a\} = r - \text{“ } \{b\} < - > a = b$
 $\langle \text{proof} \rangle$

lemma *Partial-order-eq-Image1-Image1-iff*:
 $\llbracket \text{Partial-order}(r); a : \text{field}(r); b : \text{field}(r) \rrbracket \implies$
 $r - \text{“ } \{a\} = r - \text{“ } \{b\} < - > a = b$
 $\langle \text{proof} \rangle$

lemma *Refl-antisym-eq-vimage1-vimage1-iff*:
 $\llbracket \text{refl}(\text{field}(r), r); \text{antisym}(r); a : \text{field}(r); b : \text{field}(r) \rrbracket \implies$
 $r - \text{“ } \{a\} = r - \text{“ } \{b\} < - > a = b$
 $\langle \text{proof} \rangle$

lemma *Partial-order-eq-vimage1-vimage1-iff*:

$$[\text{Partial-order}(r); a : \text{field}(r); b : \text{field}(r)] \implies$$

$$r -'' \{a\} = r -'' \{b\} \iff a = b$$

$$\langle \text{proof} \rangle$$

end

19 OrderArith: Combining Orderings: Foundations of Ordinal Arithmetic

theory *OrderArith* **imports** *Order Sum Ordinal* **begin**

definition

$radd :: [i, i, i, i] \Rightarrow i$ **where**
 $radd(A, r, B, s) ==$

$$\{z: (A+B) * (A+B).$$

$$(EX\ x\ y. z = \langle Inl(x), Inr(y) \rangle) \mid$$

$$(EX\ x'\ x. z = \langle Inl(x'), Inl(x) \rangle \ \& \ \langle x', x \rangle : r) \mid$$

$$(EX\ y'\ y. z = \langle Inr(y'), Inr(y) \rangle \ \& \ \langle y', y \rangle : s)\}$$

definition

$rmult :: [i, i, i, i] \Rightarrow i$ **where**
 $rmult(A, r, B, s) ==$

$$\{z: (A*B) * (A*B).$$

$$EX\ x'\ y'\ x\ y. z = \langle \langle x', y' \rangle, \langle x, y \rangle \rangle \ \&$$

$$(\langle x', x \rangle : r \mid (x' = x \ \& \ \langle y', y \rangle : s))\}$$

definition

$rvimage :: [i, i, i] \Rightarrow i$ **where**
 $rvimage(A, f, r) == \{z: A*A. EX\ x\ y. z = \langle x, y \rangle \ \& \ \langle f'x, f'y \rangle : r\}$

definition

$measure :: [i, i \Rightarrow i] \Rightarrow i$ **where**
 $measure(A, f) == \{\langle x, y \rangle : A*A. f(x) < f(y)\}$

19.1 Addition of Relations – Disjoint Sum

19.1.1 Rewrite rules. Can be used to obtain introduction rules

lemma *radd-Inl-Inr-iff* [*iff*]:

$$\langle Inl(a), Inr(b) \rangle : radd(A, r, B, s) \iff a:A \ \& \ b:B$$

$$\langle \text{proof} \rangle$$

lemma *radd-Inl-iff* [*iff*]:

$\langle \text{Inl}(a'), \text{Inl}(a) \rangle : \text{radd}(A, r, B, s) \leftrightarrow a':A \ \& \ a:A \ \& \ \langle a', a \rangle : r$
 $\langle \text{proof} \rangle$

lemma *radd-Inr-iff* [iff]:

$\langle \text{Inr}(b'), \text{Inr}(b) \rangle : \text{radd}(A, r, B, s) \leftrightarrow b':B \ \& \ b:B \ \& \ \langle b', b \rangle : s$
 $\langle \text{proof} \rangle$

lemma *radd-Inr-Inl-iff* [simp]:

$\langle \text{Inr}(b), \text{Inl}(a) \rangle : \text{radd}(A, r, B, s) \leftrightarrow \text{False}$
 $\langle \text{proof} \rangle$

declare *radd-Inr-Inl-iff* [THEN iffD1, dest!]

19.1.2 Elimination Rule

lemma *raddE*:

$\llbracket \langle p', p \rangle : \text{radd}(A, r, B, s);$
 $\quad !!x \ y. \llbracket p' = \text{Inl}(x); x:A; p = \text{Inr}(y); y:B \rrbracket \implies Q;$
 $\quad !!x' \ x. \llbracket p' = \text{Inl}(x'); p = \text{Inl}(x); \langle x', x \rangle : r; x':A; x:A \rrbracket \implies Q;$
 $\quad !!y' \ y. \llbracket p' = \text{Inr}(y'); p = \text{Inr}(y); \langle y', y \rangle : s; y':B; y:B \rrbracket \implies Q$
 $\rrbracket \implies Q$
 $\langle \text{proof} \rangle$

19.1.3 Type checking

lemma *radd-type*: $\text{radd}(A, r, B, s) \leq (A+B) * (A+B)$
 $\langle \text{proof} \rangle$

lemmas *field-radd* = *radd-type* [THEN field-rel-subset]

19.1.4 Linearity

lemma *linear-radd*:

$\llbracket \text{linear}(A, r); \text{linear}(B, s) \rrbracket \implies \text{linear}(A+B, \text{radd}(A, r, B, s))$
 $\langle \text{proof} \rangle$

19.1.5 Well-foundedness

lemma *wf-on-radd*: $\llbracket \text{wf}[A](r); \text{wf}[B](s) \rrbracket \implies \text{wf}[A+B](\text{radd}(A, r, B, s))$
 $\langle \text{proof} \rangle$

lemma *wf-radd*: $\llbracket \text{wf}(r); \text{wf}(s) \rrbracket \implies \text{wf}(\text{radd}(\text{field}(r), r, \text{field}(s), s))$
 $\langle \text{proof} \rangle$

lemma *well-ord-radd*:

$\llbracket \text{well-ord}(A, r); \text{well-ord}(B, s) \rrbracket \implies \text{well-ord}(A+B, \text{radd}(A, r, B, s))$
 $\langle \text{proof} \rangle$

19.1.6 An *ord-iso* congruence law

lemma *sum-bij*:

$$\begin{aligned} & \llbracket f: \text{bij}(A, C); \quad g: \text{bij}(B, D) \rrbracket \\ & \implies (\text{lam } z:A+B. \text{case}(\%x. \text{Inl}(f'x), \%y. \text{Inr}(g'y), z)) : \text{bij}(A+B, C+D) \end{aligned}$$

 $\langle \text{proof} \rangle$

lemma *sum-ord-iso-cong*:

$$\begin{aligned} & \llbracket f: \text{ord-iso}(A, r, A', r'); \quad g: \text{ord-iso}(B, s, B', s') \rrbracket \implies \\ & \quad (\text{lam } z:A+B. \text{case}(\%x. \text{Inl}(f'x), \%y. \text{Inr}(g'y), z)) \\ & \quad : \text{ord-iso}(A+B, \text{radd}(A, r, B, s), A'+B', \text{radd}(A', r', B', s')) \end{aligned}$$

 $\langle \text{proof} \rangle$

lemma *sum-disjoint-bij*: $A \text{ Int } B = 0 \implies$

$$(\text{lam } z:A+B. \text{case}(\%x. x, \%y. y, z)) : \text{bij}(A+B, A \text{ Un } B)$$

 $\langle \text{proof} \rangle$

19.1.7 Associativity

lemma *sum-assoc-bij*:

$$\begin{aligned} & (\text{lam } z:(A+B)+C. \text{case}(\text{case}(\text{Inl}, \%y. \text{Inr}(\text{Inl}(y))), \%y. \text{Inr}(\text{Inr}(y)), z)) \\ & : \text{bij}((A+B)+C, A+(B+C)) \end{aligned}$$

 $\langle \text{proof} \rangle$

lemma *sum-assoc-ord-iso*:

$$\begin{aligned} & (\text{lam } z:(A+B)+C. \text{case}(\text{case}(\text{Inl}, \%y. \text{Inr}(\text{Inl}(y))), \%y. \text{Inr}(\text{Inr}(y)), z)) \\ & : \text{ord-iso}((A+B)+C, \text{radd}(A+B, \text{radd}(A, r, B, s), C, t), \\ & \quad A+(B+C), \text{radd}(A, r, B+C, \text{radd}(B, s, C, t))) \end{aligned}$$

 $\langle \text{proof} \rangle$

19.2 Multiplication of Relations – Lexicographic Product

19.2.1 Rewrite rule. Can be used to obtain introduction rules

lemma *rmult-iff* [*iff*]:

$$\begin{aligned} & \langle \langle a', b' \rangle, \langle a, b \rangle \rangle : \text{rmult}(A, r, B, s) \iff \\ & \quad (\langle a', a \rangle : r \ \& \ a':A \ \& \ a:A \ \& \ b':B \ \& \ b:B) \mid \\ & \quad (\langle b', b \rangle : s \ \& \ a'=a \ \& \ a:A \ \& \ b':B \ \& \ b:B) \end{aligned}$$

$\langle \text{proof} \rangle$

lemma *rmultE*:

$$\begin{aligned} & \llbracket \langle \langle a', b' \rangle, \langle a, b \rangle \rangle : \text{rmult}(A, r, B, s); \\ & \quad \llbracket \langle a', a \rangle : r; \quad a':A; \quad a:A; \quad b':B; \quad b:B \rrbracket \implies Q; \\ & \quad \llbracket \langle b', b \rangle : s; \quad a:A; \quad a'=a; \quad b':B; \quad b:B \rrbracket \implies Q \\ & \rrbracket \implies Q \end{aligned}$$

 $\langle \text{proof} \rangle$

19.2.2 Type checking

lemma *rmult-type*: $rmult(A, r, B, s) \leq (A * B) * (A * B)$
 $\langle proof \rangle$

lemmas *field-rmult* = *rmult-type* [THEN *field-rel-subset*]

19.2.3 Linearity

lemma *linear-rmult*:
 $[[linear(A, r); linear(B, s)]] \implies linear(A * B, rmult(A, r, B, s))$
 $\langle proof \rangle$

19.2.4 Well-foundedness

lemma *wf-on-rmult*: $[[wf[A](r); wf[B](s)]] \implies wf[A * B](rmult(A, r, B, s))$
 $\langle proof \rangle$

lemma *wf-rmult*: $[[wf(r); wf(s)]] \implies wf(rmult(field(r), r, field(s), s))$
 $\langle proof \rangle$

lemma *well-ord-rmult*:
 $[[well-ord(A, r); well-ord(B, s)]] \implies well-ord(A * B, rmult(A, r, B, s))$
 $\langle proof \rangle$

19.2.5 An ord-iso congruence law

lemma *prod-bij*:
 $[[f: bij(A, C); g: bij(B, D)]] \implies (lam <x, y>: A * B. <f'x, g'y>) : bij(A * B, C * D)$
 $\langle proof \rangle$

lemma *prod-ord-iso-cong*:
 $[[f: ord-iso(A, r, A', r'); g: ord-iso(B, s, B', s')]] \implies (lam <x, y>: A * B. <f'x, g'y>) : ord-iso(A * B, rmult(A, r, B, s), A' * B', rmult(A', r', B', s'))$
 $\langle proof \rangle$

lemma *singleton-prod-bij*: $(lam z: A. <x, z>) : bij(A, \{x\} * A)$
 $\langle proof \rangle$

lemma *singleton-prod-ord-iso*:
 $well-ord(\{x\}, xr) \implies (lam z: A. <x, z>) : ord-iso(A, r, \{x\} * A, rmult(\{x\}, xr, A, r))$
 $\langle proof \rangle$

lemma *prod-sum-singleton-bij*:

$a \sim : C ==>$
 $(\text{lam } x : C * B + D. \text{ case } (\%x. x, \%y. <a, y>, x))$
 $: \text{bij}(C * B + D, C * B \text{ Un } \{a\} * D)$
 $\langle \text{proof} \rangle$

lemma *prod-sum-singleton-ord-iso*:

$[| a : A; \text{ well-ord}(A, r) |] ==>$
 $(\text{lam } x : \text{pred}(A, a, r) * B + \text{pred}(B, b, s). \text{ case } (\%x. x, \%y. <a, y>, x))$
 $: \text{ord-iso}(\text{pred}(A, a, r) * B + \text{pred}(B, b, s),$
 $\text{radd}(A * B, \text{rmult}(A, r, B, s), B, s),$
 $\text{pred}(A, a, r) * B \text{ Un } \{a\} * \text{pred}(B, b, s), \text{rmult}(A, r, B, s))$
 $\langle \text{proof} \rangle$

19.2.6 Distributive law

lemma *sum-prod-distrib-bij*:

$(\text{lam } <x, z> : (A + B) * C. \text{ case } (\%y. \text{Inl}(<y, z>), \%y. \text{Inr}(<y, z>), x))$
 $: \text{bij}((A + B) * C, (A * C) + (B * C))$
 $\langle \text{proof} \rangle$

lemma *sum-prod-distrib-ord-iso*:

$(\text{lam } <x, z> : (A + B) * C. \text{ case } (\%y. \text{Inl}(<y, z>), \%y. \text{Inr}(<y, z>), x))$
 $: \text{ord-iso}((A + B) * C, \text{rmult}(A + B, \text{radd}(A, r, B, s), C, t),$
 $(A * C) + (B * C), \text{radd}(A * C, \text{rmult}(A, r, C, t), B * C, \text{rmult}(B, s, C, t)))$
 $\langle \text{proof} \rangle$

19.2.7 Associativity

lemma *prod-assoc-bij*:

$(\text{lam } <<x, y>, z> : (A * B) * C. <x, <y, z>>) : \text{bij}((A * B) * C, A * (B * C))$
 $\langle \text{proof} \rangle$

lemma *prod-assoc-ord-iso*:

$(\text{lam } <<x, y>, z> : (A * B) * C. <x, <y, z>>)$
 $: \text{ord-iso}((A * B) * C, \text{rmult}(A * B, \text{rmult}(A, r, B, s), C, t),$
 $A * (B * C), \text{rmult}(A, r, B * C, \text{rmult}(B, s, C, t)))$
 $\langle \text{proof} \rangle$

19.3 Inverse Image of a Relation

19.3.1 Rewrite rule

lemma *rvimage-iff*: $<a, b> : \text{rvimage}(A, f, r) <-> <f'a, f'b> : r \ \& \ a : A \ \& \ b : A$
 $\langle \text{proof} \rangle$

19.3.2 Type checking

lemma *rvimage-type*: $\text{rvimage}(A, f, r) <= A * A$
 $\langle \text{proof} \rangle$

lemmas *field-rvimage* = *rvimage-type* [THEN *field-rel-subset*]

lemma *rvimage-converse*: $rvimage(A, f, converse(r)) = converse(rvimage(A, f, r))$
 $\langle proof \rangle$

19.3.3 Partial Ordering Properties

lemma *irrefl-rvimage*:
 $\llbracket f: inj(A, B); irrefl(B, r) \rrbracket ==> irrefl(A, rvimage(A, f, r))$
 $\langle proof \rangle$

lemma *trans-on-rvimage*:
 $\llbracket f: inj(A, B); trans[B](r) \rrbracket ==> trans[A](rvimage(A, f, r))$
 $\langle proof \rangle$

lemma *part-ord-rvimage*:
 $\llbracket f: inj(A, B); part-ord(B, r) \rrbracket ==> part-ord(A, rvimage(A, f, r))$
 $\langle proof \rangle$

19.3.4 Linearity

lemma *linear-rvimage*:
 $\llbracket f: inj(A, B); linear(B, r) \rrbracket ==> linear(A, rvimage(A, f, r))$
 $\langle proof \rangle$

lemma *tot-ord-rvimage*:
 $\llbracket f: inj(A, B); tot-ord(B, r) \rrbracket ==> tot-ord(A, rvimage(A, f, r))$
 $\langle proof \rangle$

19.3.5 Well-foundedness

lemma *wf-rvimage* [intro!]: $wf(r) ==> wf(rvimage(A, f, r))$
 $\langle proof \rangle$

But note that the combination of *wf-imp-wf-on* and *wf-rvimage* gives $wf(r) \implies wf[C](rvimage(A, f, r))$

lemma *wf-on-rvimage*: $\llbracket f: A \multimap B; wf[B](r) \rrbracket ==> wf[A](rvimage(A, f, r))$
 $\langle proof \rangle$

lemma *well-ord-rvimage*:
 $\llbracket f: inj(A, B); well-ord(B, r) \rrbracket ==> well-ord(A, rvimage(A, f, r))$
 $\langle proof \rangle$

lemma *ord-iso-rvimage*:
 $f: bij(A, B) ==> f: ord-iso(A, rvimage(A, f, s), B, s)$
 $\langle proof \rangle$

lemma *ord-iso-rvimage-eq*:
 $f: ord-iso(A, r, B, s) ==> rvimage(A, f, s) = r \text{ Int } A * A$

$\langle proof \rangle$

19.4 Every well-founded relation is a subset of some inverse image of an ordinal

lemma *wf-rvimage-Ord*: $Ord(i) \implies wf(rvimage(A, f, Memrel(i)))$
 $\langle proof \rangle$

definition

$wfrank :: [i, i] \implies i$ **where**
 $wfrank(r, a) == wfrec(r, a, \%x f. \bigcup y \in r - \{\{x\}. succ(f'y))$

definition

$wftype :: i \implies i$ **where**
 $wftype(r) == \bigcup y \in range(r). succ(wfrank(r, y))$

lemma *wfrank*: $wf(r) \implies wfrank(r, a) = (\bigcup y \in r - \{\{a\}. succ(wfrank(r, y))$
 $\langle proof \rangle$

lemma *Ord-wfrank*: $wf(r) \implies Ord(wfrank(r, a))$
 $\langle proof \rangle$

lemma *wfrank-lt*: $[|wf(r); \langle a, b \rangle \in r|] \implies wfrank(r, a) < wfrank(r, b)$
 $\langle proof \rangle$

lemma *Ord-wftype*: $wf(r) \implies Ord(wftype(r))$
 $\langle proof \rangle$

lemma *wftypeI*: $[|wf(r); x \in field(r)|] \implies wfrank(r, x) \in wftype(r)$
 $\langle proof \rangle$

lemma *wf-imp-subset-rvimage*:

$[|wf(r); r \subseteq A * A|] \implies \exists i f. Ord(i) \ \& \ r \leq rvimage(A, f, Memrel(i))$
 $\langle proof \rangle$

theorem *wf-iff-subset-rvimage*:

$relation(r) \implies wf(r) \iff (\exists i f A. Ord(i) \ \& \ r \leq rvimage(A, f, Memrel(i)))$
 $\langle proof \rangle$

19.5 Other Results

lemma *wf-times*: $A \ Int \ B = 0 \implies wf(A * B)$
 $\langle proof \rangle$

Could also be used to prove *wf-radd*

lemma *wf-Un*:

$[| range(r) \ Int \ domain(s) = 0; wf(r); wf(s) |] \implies wf(r \ Un \ s)$

<proof>

19.5.1 The Empty Relation

lemma *wf0*: *wf*(0)

<proof>

lemma *linear0*: *linear*(0,0)

<proof>

lemma *well-ord0*: *well-ord*(0,0)

<proof>

19.5.2 The "measure" relation is useful with wfrec

lemma *measure-eq-rvimage-Memrel*:

measure(A,f) = *rvimage*(A,Lambda(A,f),*Memrel*(*Collect*(*RepFun*(A,f),*Ord*)))

<proof>

lemma *wf-measure* [iff]: *wf*(*measure*(A,f))

<proof>

lemma *measure-iff* [iff]: $\langle x,y \rangle : \text{measure}(A,f) \prec \rightarrow x:A \ \& \ y:A \ \& \ f(x) < f(y)$

<proof>

lemma *linear-measure*:

assumes *Ord*f: $\forall x. x \in A \implies \text{Ord}(f(x))$

and *inj*: $\forall x y. [|x \in A; y \in A; f(x) = f(y)|] \implies x=y$

shows *linear*(A, *measure*(A,f))

<proof>

lemma *wf-on-measure*: *wf*[B](*measure*(A,f))

<proof>

lemma *well-ord-measure*:

assumes *Ord*f: $\forall x. x \in A \implies \text{Ord}(f(x))$

and *inj*: $\forall x y. [|x \in A; y \in A; f(x) = f(y)|] \implies x=y$

shows *well-ord*(A, *measure*(A,f))

<proof>

lemma *measure-type*: *measure*(A,f) $\leq A * A$

<proof>

19.5.3 Well-foundedness of Unions

lemma *wf-on-Union*:

assumes *wfA*: *wf*[A](*r*)

and *wfB*: $\forall a. a \in A \implies \text{wf}[B(a)](s)$

and *ok*: $\forall a u v. [| \langle u,v \rangle \in s; v \in B(a); a \in A |]$

$\implies (\exists a' \in A. \langle a',a \rangle \in r \ \& \ u \in B(a')) \mid u \in B(a)$

shows $wf[\bigcup a \in A. B(a)](s)$
 $\langle proof \rangle$

19.5.4 Bijections involving Powersets

lemma *Pow-sum-bij*:

$(\lambda Z \in Pow(A+B). <\{x \in A. Inl(x) \in Z\}, \{y \in B. Inr(y) \in Z\}>)$
 $\in bij(Pow(A+B), Pow(A)*Pow(B))$

$\langle proof \rangle$

As a special case, we have $bij(Pow(A \times B), A \rightarrow Pow(B))$

lemma *Pow-Sigma-bij*:

$(\lambda r \in Pow(Sigma(A,B)). \lambda x \in A. r''\{x\})$
 $\in bij(Pow(Sigma(A,B)), \Pi x \in A. Pow(B(x)))$

$\langle proof \rangle$

end

20 OrderType: Order Types and Ordinal Arithmetic

theory *OrderType* **imports** *OrderArith OrdQuant Nat-ZF* **begin**

The order type of a well-ordering is the least ordinal isomorphic to it. Ordinal arithmetic is traditionally defined in terms of order types, as it is here. But a definition by transfinite recursion would be much simpler!

definition

ordermap $:: [i,i] \Rightarrow i$ **where**
ordermap(A,r) $== lam\ x:A. wfrec[A](r, x, \%x\ f. f\ \text{“}\ pred(A,x,r))$

definition

ordertype $:: [i,i] \Rightarrow i$ **where**
ordertype(A,r) $== ordermap(A,r)\ \text{“}\ A$

definition

Ord-alt $:: i \Rightarrow o$ **where**
Ord-alt(X) $== well-ord(X, Memrel(X)) \ \& \ (ALL\ u:X. u=pred(X, u, Memrel(X)))$

definition

ordify $:: i \Rightarrow i$ **where**
ordify(x) $== if\ Ord(x)\ then\ x\ else\ 0$

definition

omult :: $[i,i] \Rightarrow i$ (**infixl** ** 70) **where**
*i ** j* == *ordertype(j*i, rmult(j,Memrel(j),i,Memrel(i)))*

definition

raw-oadd :: $[i,i] \Rightarrow i$ **where**
raw-oadd(i,j) == *ordertype(i+j, radd(i,Memrel(i),j,Memrel(j)))*

definition

oadd :: $[i,i] \Rightarrow i$ (**infixl** ++ 65) **where**
i ++ j == *raw-oadd(ordify(i),ordify(j))*

definition

odiff :: $[i,i] \Rightarrow i$ (**infixl** -- 65) **where**
i -- j == *ordertype(i-j, Memrel(i))*

notation (*xsymbols*)

omult (**infixl** $\times \times$ 70)

notation (*HTML output*)

omult (**infixl** $\times \times$ 70)

20.1 Proofs needing the combination of Ordinal.thy and Order.thy

lemma *le-well-ord-Memrel*: $j \text{ le } i \Rightarrow \text{well-ord}(j, \text{Memrel}(i))$
 $\langle \text{proof} \rangle$

lemmas *well-ord-Memrel* = *le-refl* [THEN *le-well-ord-Memrel*]

lemma *lt-pred-Memrel*:

$j < i \Rightarrow \text{pred}(i, j, \text{Memrel}(i)) = j$
 $\langle \text{proof} \rangle$

lemma *pred-Memrel*:

$x:A \Rightarrow \text{pred}(A, x, \text{Memrel}(A)) = A \text{ Int } x$
 $\langle \text{proof} \rangle$

lemma *Ord-iso-implies-eq-lemma*:

$[j < i; f: \text{ord-iso}(i, \text{Memrel}(i), j, \text{Memrel}(j))] \Rightarrow R$
 $\langle \text{proof} \rangle$

lemma *Ord-iso-implies-eq*:

$[\text{Ord}(i); \text{Ord}(j); f: \text{ord-iso}(i, \text{Memrel}(i), j, \text{Memrel}(j))] \Rightarrow$

$\implies i=j$
 $\langle \text{proof} \rangle$

20.2 Ordermap and ordertype

lemma *ordermap-type*:
 $\text{ordermap}(A,r) : A \rightarrow \text{ordertype}(A,r)$
 $\langle \text{proof} \rangle$

20.2.1 Unfolding of ordermap

lemma *ordermap-eq-image*:
 $[[\text{wf}[A](r); x:A]] \implies \text{ordermap}(A,r) \text{ ' } x = \text{ordermap}(A,r) \text{ ' ' } \text{pred}(A,x,r)$
 $\langle \text{proof} \rangle$

lemma *ordermap-pred-unfold*:
 $[[\text{wf}[A](r); x:A]] \implies \text{ordermap}(A,r) \text{ ' } x = \{ \text{ordermap}(A,r) \text{ ' } y \mid y : \text{pred}(A,x,r) \}$
 $\langle \text{proof} \rangle$

lemmas *ordermap-unfold* = *ordermap-pred-unfold* [*simplified pred-def*]

20.2.2 Showing that ordermap, ordertype yield ordinals

lemma *Ord-ordermap*:
 $[[\text{well-ord}(A,r); x:A]] \implies \text{Ord}(\text{ordermap}(A,r) \text{ ' } x)$
 $\langle \text{proof} \rangle$

lemma *Ord-ordertype*:
 $\text{well-ord}(A,r) \implies \text{Ord}(\text{ordertype}(A,r))$
 $\langle \text{proof} \rangle$

20.2.3 ordermap preserves the orderings in both directions

lemma *ordermap-mono*:
 $[[<w,x>: r; \text{wf}[A](r); w:A; x:A]] \implies \text{ordermap}(A,r) \text{ ' } w : \text{ordermap}(A,r) \text{ ' } x$
 $\langle \text{proof} \rangle$

lemma *converse-ordermap-mono*:
 $[[\text{ordermap}(A,r) \text{ ' } w : \text{ordermap}(A,r) \text{ ' } x; \text{well-ord}(A,r); w:A; x:A]] \implies <w,x>: r$
 $\langle \text{proof} \rangle$

lemmas *ordermap-surj* =
 ordermap-type [*THEN surj-image, unfolded ordertype-def* [*symmetric*]]

lemma *ordermap-bij*:

$well\text{-}ord(A,r) ==> ordermap(A,r) : bij(A, ordertype(A,r))$
 $\langle proof \rangle$

20.2.4 Isomorphisms involving ordertype

lemma *ordertype-ord-iso*:

$well\text{-}ord(A,r)$
 $==> ordermap(A,r) : ord\text{-}iso(A,r, ordertype(A,r), Memrel(ordertype(A,r)))$
 $\langle proof \rangle$

lemma *ordertype-eq*:

$[| f : ord\text{-}iso(A,r,B,s); well\text{-}ord(B,s) |]$
 $==> ordertype(A,r) = ordertype(B,s)$
 $\langle proof \rangle$

lemma *ordertype-eq-imp-ord-iso*:

$[| ordertype(A,r) = ordertype(B,s); well\text{-}ord(A,r); well\text{-}ord(B,s) |]$
 $==> EX f. f : ord\text{-}iso(A,r,B,s)$
 $\langle proof \rangle$

20.2.5 Basic equalities for ordertype

lemma *le-ordertype-Memrel*: $j \text{ le } i ==> ordertype(j, Memrel(i)) = j$
 $\langle proof \rangle$

lemmas *ordertype-Memrel* = *le-reft* [THEN *le-ordertype-Memrel*]

lemma *ordertype-0* [*simp*]: $ordertype(0,r) = 0$
 $\langle proof \rangle$

lemmas *bij-ordertype-vimage* = *ord-iso-rvimage* [THEN *ordertype-eq*]

20.2.6 A fundamental unfolding law for ordertype.

lemma *ordermap-pred-eq-ordermap*:

$[| well\text{-}ord(A,r); y:A; z: pred(A,y,r) |]$
 $==> ordermap(pred(A,y,r), r) ' z = ordermap(A, r) ' z$
 $\langle proof \rangle$

lemma *ordertype-unfold*:

$ordertype(A,r) = \{ ordermap(A,r) ' y . y : A \}$
 $\langle proof \rangle$

Theorems by Krzysztof Grabczewski; proofs simplified by lcp

lemma *ordertype-pred-subset*: $[| well\text{-}ord(A,r); x:A |] ==>$
 $ordertype(pred(A,x,r),r) <= ordertype(A,r)$

$\langle \text{proof} \rangle$

lemma *ordertype-pred-lt*:

$[[\text{well-ord}(A, r); x:A]] \implies \text{ordertype}(\text{pred}(A, x, r), r) < \text{ordertype}(A, r)$

$\langle \text{proof} \rangle$

lemma *ordertype-pred-unfold*:

$\text{well-ord}(A, r) \implies \text{ordertype}(A, r) = \{ \text{ordertype}(\text{pred}(A, x, r), r). x:A \}$

$\langle \text{proof} \rangle$

20.3 Alternative definition of ordinal

lemma *Ord-is-Ord-alt*: $\text{Ord}(i) \implies \text{Ord-alt}(i)$

$\langle \text{proof} \rangle$

lemma *Ord-alt-is-Ord*:

$\text{Ord-alt}(i) \implies \text{Ord}(i)$

$\langle \text{proof} \rangle$

20.4 Ordinal Addition

20.4.1 Order Type calculations for radd

Addition with 0

lemma *bij-sum-0*: $(\text{lam } z:A+0. \text{case}(\%x. x, \%y. y, z)) : \text{bij}(A+0, A)$

$\langle \text{proof} \rangle$

lemma *ordertype-sum-0-eq*:

$\text{well-ord}(A, r) \implies \text{ordertype}(A+0, \text{radd}(A, r, 0, s)) = \text{ordertype}(A, r)$

$\langle \text{proof} \rangle$

lemma *bij-0-sum*: $(\text{lam } z:0+A. \text{case}(\%x. x, \%y. y, z)) : \text{bij}(0+A, A)$

$\langle \text{proof} \rangle$

lemma *ordertype-0-sum-eq*:

$\text{well-ord}(A, r) \implies \text{ordertype}(0+A, \text{radd}(0, s, A, r)) = \text{ordertype}(A, r)$

$\langle \text{proof} \rangle$

Initial segments of radd. Statements by Grabczewski

lemma *pred-Inl-bij*:

$a:A \implies (\text{lam } x:\text{pred}(A, a, r). \text{Inl}(x)) : \text{bij}(\text{pred}(A, a, r), \text{pred}(A+B, \text{Inl}(a), \text{radd}(A, r, B, s)))$

$\langle \text{proof} \rangle$

lemma *ordertype-pred-Inl-eq*:

$$\begin{aligned}
& [| a:A; \text{well-ord}(A,r) |] \\
& \implies \text{ordertype}(\text{pred}(A+B, \text{Inl}(a), \text{radd}(A,r,B,s)), \text{radd}(A,r,B,s)) = \\
& \quad \text{ordertype}(\text{pred}(A,a,r), r)
\end{aligned}$$
 $\langle \text{proof} \rangle$

lemma *pred-Inr-bij*:

$$\begin{aligned}
& b:B \implies \\
& \quad \text{id}(A+\text{pred}(B,b,s)) \\
& \quad : \text{bij}(A+\text{pred}(B,b,s), \text{pred}(A+B, \text{Inr}(b), \text{radd}(A,r,B,s)))
\end{aligned}$$
 $\langle \text{proof} \rangle$

lemma *ordertype-pred-Inr-eq*:

$$\begin{aligned}
& [| b:B; \text{well-ord}(A,r); \text{well-ord}(B,s) |] \\
& \implies \text{ordertype}(\text{pred}(A+B, \text{Inr}(b), \text{radd}(A,r,B,s)), \text{radd}(A,r,B,s)) = \\
& \quad \text{ordertype}(A+\text{pred}(B,b,s), \text{radd}(A,r,\text{pred}(B,b,s),s))
\end{aligned}$$
 $\langle \text{proof} \rangle$

20.4.2 ordify: trivial coercion to an ordinal

lemma *Ord-ordify* [*iff*, *TC*]: $\text{Ord}(\text{ordify}(x))$

$\langle \text{proof} \rangle$

lemma *ordify-idem* [*simp*]: $\text{ordify}(\text{ordify}(x)) = \text{ordify}(x)$

$\langle \text{proof} \rangle$

20.4.3 Basic laws for ordinal addition

lemma *Ord-raw-oadd*: $[|\text{Ord}(i); \text{Ord}(j)|] \implies \text{Ord}(\text{raw-oadd}(i,j))$

$\langle \text{proof} \rangle$

lemma *Ord-oadd* [*iff*, *TC*]: $\text{Ord}(i++j)$

$\langle \text{proof} \rangle$

Ordinal addition with zero

lemma *raw-oadd-0*: $\text{Ord}(i) \implies \text{raw-oadd}(i,0) = i$

$\langle \text{proof} \rangle$

lemma *oadd-0* [*simp*]: $\text{Ord}(i) \implies i++0 = i$

$\langle \text{proof} \rangle$

lemma *raw-oadd-0-left*: $\text{Ord}(i) \implies \text{raw-oadd}(0,i) = i$

$\langle \text{proof} \rangle$

lemma *oadd-0-left* [*simp*]: $\text{Ord}(i) \implies 0++i = i$

$\langle \text{proof} \rangle$

lemma *oadd-eq-if-raw-oadd*:

$$i++j = (\text{if } \text{Ord}(i) \text{ then } (\text{if } \text{Ord}(j) \text{ then } \text{raw-oadd}(i,j) \text{ else } i))$$

else (if Ord(j) then j else 0))

<proof>

lemma *raw-oadd-eq-oadd*: $[[Ord(i); Ord(j)]] ==> raw-oadd(i,j) = i++j$

<proof>

lemma *lt-oadd1*: $k < i ==> k < i++j$

<proof>

lemma *oadd-le-self*: $Ord(i) ==> i \leq i++j$

<proof>

Various other results

lemma *id-ord-iso-Memrel*: $A \leq B ==> id(A) : ord-iso(A, Memrel(A), A, Memrel(B))$

<proof>

lemma *subset-ord-iso-Memrel*:

$[[f : ord-iso(A, Memrel(B), C, r); A \leq B]] ==> f : ord-iso(A, Memrel(A), C, r)$

<proof>

lemma *restrict-ord-iso*:

$[[f \in ord-iso(i, Memrel(i), Order.pred(A, a, r), r); a \in A; j < i;$
 $trans[A](r)]]$
 $==> restrict(f, j) \in ord-iso(j, Memrel(j), Order.pred(A, f'j, r), r)$

<proof>

lemma *restrict-ord-iso2*:

$[[f \in ord-iso(Order.pred(A, a, r), r, i, Memrel(i)); a \in A;$
 $j < i; trans[A](r)]]$
 $==> converse(restrict(converse(f), j))$
 $\in ord-iso(Order.pred(A, converse(f)'j, r), r, j, Memrel(j))$

<proof>

lemma *ordertype-sum-Memrel*:

$[[well-ord(A, r); k < j]]$
 $==> ordertype(A+k, radd(A, r, k, Memrel(j))) =$
 $ordertype(A+k, radd(A, r, k, Memrel(k)))$

<proof>

lemma *oadd-lt-mono2*: $k < j ==> i++k < i++j$

<proof>

lemma *oadd-lt-cancel2*: $[[i++j < i++k; Ord(j)]] ==> j < k$

<proof>

lemma *oadd-lt-iff2*: $Ord(j) \implies i++j < i++k \iff j < k$
 $\langle proof \rangle$

lemma *oadd-inject*: $[| i++j = i++k; Ord(j); Ord(k) |] \implies j=k$
 $\langle proof \rangle$

lemma *lt-oadd-disj*: $k < i++j \implies k < i \mid (EX\ l:j. k = i++l)$
 $\langle proof \rangle$

20.4.4 Ordinal addition with successor – via associativity!

lemma *oadd-assoc*: $(i++j)++k = i++(j++k)$
 $\langle proof \rangle$

lemma *oadd-unfold*: $[| Ord(i); Ord(j) |] \implies i++j = i\ Un\ (\bigcup_{k \in j}. \{i++k\})$
 $\langle proof \rangle$

lemma *oadd-1*: $Ord(i) \implies i++1 = succ(i)$
 $\langle proof \rangle$

lemma *oadd-succ* [*simp*]: $Ord(j) \implies i++succ(j) = succ(i++j)$
 $\langle proof \rangle$

Ordinal addition with limit ordinals

lemma *oadd-UN*:
 $[| !!x. x:A \implies Ord(j(x)); a:A |]$
 $\implies i++(\bigcup_{x \in A}. j(x)) = (\bigcup_{x \in A}. i++j(x))$
 $\langle proof \rangle$

lemma *oadd-Limit*: $Limit(j) \implies i++j = (\bigcup_{k \in j}. i++k)$
 $\langle proof \rangle$

lemma *oadd-eq-0-iff*: $[| Ord(i); Ord(j) |] \implies (i++j) = 0 \iff i=0 \ \& \ j=0$
 $\langle proof \rangle$

lemma *oadd-eq-lt-iff*: $[| Ord(i); Ord(j) |] \implies 0 < (i++j) \iff 0 < i \mid 0 < j$
 $\langle proof \rangle$

lemma *oadd-LimitI*: $[| Ord(i); Limit(j) |] \implies Limit(i++j)$
 $\langle proof \rangle$

Order/monotonicity properties of ordinal addition

lemma *oadd-le-self2*: $Ord(i) \implies i\ le\ j++i$
 $\langle proof \rangle$

lemma *oadd-le-mono1*: $k\ le\ j \implies k++i\ le\ j++i$
 $\langle proof \rangle$

lemma *oadd-lt-mono*: $[[i' \text{ le } i; j' < j]] ==> i'++j' < i++j$
 $\langle \text{proof} \rangle$

lemma *oadd-le-mono*: $[[i' \text{ le } i; j' \text{ le } j]] ==> i'++j' \text{ le } i++j$
 $\langle \text{proof} \rangle$

lemma *oadd-le-iff2*: $[[\text{Ord}(j); \text{Ord}(k)]] ==> i++j \text{ le } i++k <-> j \text{ le } k$
 $\langle \text{proof} \rangle$

lemma *oadd-lt-self*: $[[\text{Ord}(i); 0 < j]] ==> i < i++j$
 $\langle \text{proof} \rangle$

Every ordinal is exceeded by some limit ordinal.

lemma *Ord-imp-greater-Limit*: $\text{Ord}(i) ==> \exists k. i < k \ \& \ \text{Limit}(k)$
 $\langle \text{proof} \rangle$

lemma *Ord2-imp-greater-Limit*: $[[\text{Ord}(i); \text{Ord}(j)]] ==> \exists k. i < k \ \& \ j < k \ \& \ \text{Limit}(k)$
 $\langle \text{proof} \rangle$

20.5 Ordinal Subtraction

The difference is $\text{ordertype}(j - i, \text{Memrel}(j))$. It's probably simpler to define the difference recursively!

lemma *bij-sum-Diff*:
 $A <= B ==> (\text{lam } y:B. \text{ if } (y:A, \text{Inl}(y), \text{Inr}(y))) : \text{bij}(B, A+(B-A))$
 $\langle \text{proof} \rangle$

lemma *ordertype-sum-Diff*:
 $i \text{ le } j ==>$
 $\text{ordertype}(i+(j-i), \text{radd}(i, \text{Memrel}(j), j-i, \text{Memrel}(j))) =$
 $\text{ordertype}(j, \text{Memrel}(j))$
 $\langle \text{proof} \rangle$

lemma *Ord-odiff* $[\text{simp}, \text{TC}]$:
 $[[\text{Ord}(i); \text{Ord}(j)]] ==> \text{Ord}(i--j)$
 $\langle \text{proof} \rangle$

lemma *raw-oadd-ordertype-Diff*:
 $i \text{ le } j$
 $==> \text{raw-oadd}(i, j--i) = \text{ordertype}(i+(j-i), \text{radd}(i, \text{Memrel}(j), j-i, \text{Memrel}(j)))$
 $\langle \text{proof} \rangle$

lemma *oadd-odiff-inverse*: $i \text{ le } j ==> i ++ (j--i) = j$
 $\langle \text{proof} \rangle$

lemma *odiff-oadd-inverse*: $[[\text{Ord}(i); \text{Ord}(j)]] ==> (i++j) -- i = j$

$\langle proof \rangle$

lemma *odiff-lt-mono2*: $[i < j; k \leq i] \implies i - k < j - k$
 $\langle proof \rangle$

20.6 Ordinal Multiplication

lemma *Ord-omult* [*simp*, *TC*]:
 $[Ord(i); Ord(j)] \implies Ord(i ** j)$
 $\langle proof \rangle$

20.6.1 A useful unfolding law

lemma *pred-Pair-eq*:
 $[a:A; b:B] \implies pred(A*B, \langle a, b \rangle, rmult(A, r, B, s)) =$
 $pred(A, a, r) * B \text{ Un } (\{a\} * pred(B, b, s))$
 $\langle proof \rangle$

lemma *ordertype-pred-Pair-eq*:
 $[a:A; b:B; well-ord(A, r); well-ord(B, s)] \implies$
 $ordertype(pred(A*B, \langle a, b \rangle, rmult(A, r, B, s)), rmult(A, r, B, s)) =$
 $ordertype(pred(A, a, r) * B + pred(B, b, s),$
 $radd(A*B, rmult(A, r, B, s), B, s))$
 $\langle proof \rangle$

lemma *ordertype-pred-Pair-lemma*:
 $[i' < i; j' < j] \implies ordertype(pred(i*j, \langle i', j' \rangle, rmult(i, Memrel(i), j, Memrel(j))),$
 $rmult(i, Memrel(i), j, Memrel(j))) =$
 $raw-oadd(j ** i', j')$
 $\langle proof \rangle$

lemma *lt-omult*:
 $[Ord(i); Ord(j); k < j ** i] \implies EX j' i'. k = j ** i' ++ j' \ \& \ j' < j \ \& \ i' < i$
 $\langle proof \rangle$

lemma *omult-oadd-lt*:
 $[j' < j; i' < i] \implies j ** i' ++ j' < j ** i$
 $\langle proof \rangle$

lemma *omult-unfold*:
 $[Ord(i); Ord(j)] \implies j ** i = (\bigcup j' \in j. \bigcup i' \in i. \{j ** i' ++ j'\})$
 $\langle proof \rangle$

20.6.2 Basic laws for ordinal multiplication

Ordinal multiplication by zero

lemma *omult-0* [*simp*]: $i ** 0 = 0$

$\langle proof \rangle$

lemma *omult-0-left* [simp]: $0 ** i = 0$

$\langle proof \rangle$

Ordinal multiplication by 1

lemma *omult-1* [simp]: $Ord(i) ==> i ** 1 = i$

$\langle proof \rangle$

lemma *omult-1-left* [simp]: $Ord(i) ==> 1 ** i = i$

$\langle proof \rangle$

Distributive law for ordinal multiplication and addition

lemma *oadd-omult-distrib*:

$[| Ord(i); Ord(j); Ord(k) |] ==> i ** (j ++ k) = (i ** j) ++ (i ** k)$

$\langle proof \rangle$

lemma *omult-succ*: $[| Ord(i); Ord(j) |] ==> i ** succ(j) = (i ** j) ++ i$

$\langle proof \rangle$

Associative law

lemma *omult-assoc*:

$[| Ord(i); Ord(j); Ord(k) |] ==> (i ** j) ** k = i ** (j ** k)$

$\langle proof \rangle$

Ordinal multiplication with limit ordinals

lemma *omult-UN*:

$[| Ord(i); !!x. x:A ==> Ord(j(x)) |]$
 $==> i ** (\bigcup x \in A. j(x)) = (\bigcup x \in A. i ** j(x))$

$\langle proof \rangle$

lemma *omult-Limit*: $[| Ord(i); Limit(j) |] ==> i ** j = (\bigcup k \in j. i ** k)$

$\langle proof \rangle$

20.6.3 Ordering/monotonicity properties of ordinal multiplication

lemma *lt-omult1*: $[| k < i; 0 < j |] ==> k < i ** j$

$\langle proof \rangle$

lemma *omult-le-self*: $[| Ord(i); 0 < j |] ==> i \leq i ** j$

$\langle proof \rangle$

lemma *omult-le-mono1*: $[| k \leq j; Ord(i) |] ==> k ** i \leq j ** i$

$\langle proof \rangle$

lemma *omult-lt-mono2*: $[| k < j; 0 < i |] ==> i ** k < i ** j$

$\langle proof \rangle$

lemma *omult-le-mono2*: $[[k \leq j; \text{Ord}(i)]] \implies i ** k \leq i ** j$
 $\langle \text{proof} \rangle$

lemma *omult-le-mono*: $[[i' \leq i; j' \leq j]] \implies i' ** j' \leq i ** j$
 $\langle \text{proof} \rangle$

lemma *omult-lt-mono*: $[[i' \leq i; j' < j; 0 < i]] \implies i' ** j' < i ** j$
 $\langle \text{proof} \rangle$

lemma *omult-le-self2*: $[[\text{Ord}(i); 0 < j]] \implies i \leq j ** i$
 $\langle \text{proof} \rangle$

Further properties of ordinal multiplication

lemma *omult-inject*: $[[i ** j = i ** k; 0 < i; \text{Ord}(j); \text{Ord}(k)]] \implies j = k$
 $\langle \text{proof} \rangle$

20.7 The Relation Lt

lemma *wf-Lt*: $\text{wf}(Lt)$
 $\langle \text{proof} \rangle$

lemma *irrefl-Lt*: $\text{irrefl}(A, Lt)$
 $\langle \text{proof} \rangle$

lemma *trans-Lt*: $\text{trans}[A](Lt)$
 $\langle \text{proof} \rangle$

lemma *part-ord-Lt*: $\text{part-ord}(A, Lt)$
 $\langle \text{proof} \rangle$

lemma *linear-Lt*: $\text{linear}(\text{nat}, Lt)$
 $\langle \text{proof} \rangle$

lemma *tot-ord-Lt*: $\text{tot-ord}(\text{nat}, Lt)$
 $\langle \text{proof} \rangle$

lemma *well-ord-Lt*: $\text{well-ord}(\text{nat}, Lt)$
 $\langle \text{proof} \rangle$

end

21 Finite: Finite Powerset Operator and Finite Function Space

theory *Finite* **imports** *Inductive-ZF Epsilon Nat-ZF* **begin**

```

rep-datatype
  elimination    natE
  induction      nat-induct
  case-eqns      nat-case-0 nat-case-succ
  recursor-eqns  recursor-0 recursor-succ

```

```

consts
  Fin      ::  $i \Rightarrow i$ 
  FiniteFun ::  $[i, i] \Rightarrow i$        $((- \text{ -- } ||> / -) [61, 60] 60)$ 

```

```

inductive
  domains    Fin(A) <= Pow(A)
  intros
    emptyI: 0 : Fin(A)
    consI:   $[[ a : A; b : Fin(A) ]] \Rightarrow cons(a, b) : Fin(A)$ 
  type-intros empty-subsetI cons-subsetI PowI
  type-elim   PowD [THEN revcut-rl]

```

```

inductive
  domains    FiniteFun(A, B) <= Fin(A*B)
  intros
    emptyI: 0 :  $A \text{ -- } ||> B$ 
    consI:   $[[ a : A; b : B; h : A \text{ -- } ||> B; a \sim : domain(h) ]] \Rightarrow cons(<a, b>, h) : A \text{ -- } ||> B$ 
  type-intros Fin.intros

```

21.1 Finite Powerset Operator

```

lemma Fin-mono:  $A \leq B \Rightarrow Fin(A) \leq Fin(B)$ 
<proof>

```

```

lemmas FinD = Fin.dom-subset [THEN subsetD, THEN PowD, standard]

```

```

lemma Fin-induct [case-names 0 cons, induct set: Fin]:
   $[[ b : Fin(A);$ 
     $P(0);$ 
     $!!x y. [[ x : A; y : Fin(A); x \sim y; P(y) ]] \Rightarrow P(cons(x, y))$ 
   $]] \Rightarrow P(b)$ 
<proof>

```

```

declare Fin.intros [simp]

```

lemma *Fin-0*: $Fin(0) = \{0\}$

$\langle proof \rangle$

lemma *Fin-UnI* [*simp*]: $[| b: Fin(A); c: Fin(A) |] ==> b \text{ Un } c : Fin(A)$

$\langle proof \rangle$

lemma *Fin-UnionI*: $C : Fin(Fin(A)) ==> Union(C) : Fin(A)$

$\langle proof \rangle$

lemma *Fin-subset-lemma* [*rule-format*]: $b: Fin(A) ==> \forall z. z \leq b \text{ --> } z: Fin(A)$

$\langle proof \rangle$

lemma *Fin-subset*: $[| c \leq b; b: Fin(A) |] ==> c: Fin(A)$

$\langle proof \rangle$

lemma *Fin-IntI1* [*intro,simp*]: $b: Fin(A) ==> b \text{ Int } c : Fin(A)$

$\langle proof \rangle$

lemma *Fin-IntI2* [*intro,simp*]: $c: Fin(A) ==> b \text{ Int } c : Fin(A)$

$\langle proof \rangle$

lemma *Fin-0-induct-lemma* [*rule-format*]:

$[| c: Fin(A); b: Fin(A); P(b);$
 $!!x y. [| x: A; y: Fin(A); x:y; P(y) |] ==> P(y-\{x\})$
 $|] ==> c \leq b \text{ --> } P(b-c)$

$\langle proof \rangle$

lemma *Fin-0-induct*:

$[| b: Fin(A);$
 $P(b);$
 $!!x y. [| x: A; y: Fin(A); x:y; P(y) |] ==> P(y-\{x\})$
 $|] ==> P(0)$

$\langle proof \rangle$

lemma *nat-fun-subset-Fin*: $n: nat ==> n \text{ --> } A \leq Fin(nat * A)$

$\langle proof \rangle$

21.2 Finite Function Space

lemma *FiniteFun-mono*:

$[| A \leq C; B \leq D |] ==> A \text{ --||> } B \leq C \text{ --||> } D$

$\langle proof \rangle$

lemma *FiniteFun-mono1*: $A \leq B \implies A -||> A \leq B -||> B$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-is-fun*: $h: A -||> B \implies h: \text{domain}(h) \rightarrow B$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-domain-Fin*: $h: A -||> B \implies \text{domain}(h) : \text{Fin}(A)$
 $\langle \text{proof} \rangle$

lemmas *FiniteFun-apply-type* = *FiniteFun-is-fun* [THEN *apply-type*, *standard*]

lemma *FiniteFun-subset-lemma* [rule-format]:
 $b: A -||> B \implies \text{ALL } z. z \leq b \implies z: A -||> B$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-subset*: $[c \leq b; b: A -||> B] \implies c: A -||> B$
 $\langle \text{proof} \rangle$

lemma *fun-FiniteFunI* [rule-format]: $A: \text{Fin}(X) \implies \text{ALL } f. f: A \rightarrow B \implies f: A -||> B$
 $\langle \text{proof} \rangle$

lemma *lam-FiniteFun*: $A: \text{Fin}(X) \implies (\text{lam } x:A. b(x)) : A -||> \{b(x). x:A\}$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-Collect-iff*:
 $f : \text{FiniteFun}(A, \{y:B. P(y)\})$
 $\iff f : \text{FiniteFun}(A,B) \ \& \ (\text{ALL } x:\text{domain}(f). P(f'x))$
 $\langle \text{proof} \rangle$

21.3 The Contents of a Singleton Set

definition

$\text{contents} :: i \Rightarrow i$ **where**
 $\text{contents}(X) == \text{THE } x. X = \{x\}$

lemma *contents-eq* [simp]: $\text{contents } (\{x\}) = x$
 $\langle \text{proof} \rangle$

end

22 Cardinal: Cardinal Numbers Without the Axiom of Choice

theory *Cardinal* **imports** *OrderType Finite Nat-ZF Sum* **begin**

definition

Least :: $(i \Rightarrow o) \Rightarrow i$ (**binder** *LEAST* 10) **where**
Least(*P*) == *THE* *i*. *Ord*(*i*) & *P*(*i*) & (*ALL* *j*. $j < i \Rightarrow \sim P(j)$)

definition

eqpoll :: $[i, i] \Rightarrow o$ (**infixl** *eqpoll* 50) **where**
A eqpoll B == *EX* *f*. *f*: *bij*(*A*, *B*)

definition

lepoll :: $[i, i] \Rightarrow o$ (**infixl** *lepoll* 50) **where**
A lepoll B == *EX* *f*. *f*: *inj*(*A*, *B*)

definition

lesspoll :: $[i, i] \Rightarrow o$ (**infixl** *lesspoll* 50) **where**
A lesspoll B == *A lepoll B* & $\sim(A \text{ eqpoll } B)$

definition

cardinal :: $i \Rightarrow i$ (**|**-) **where**
 $|A|$ == *LEAST* *i*. *i eqpoll A*

definition

Finite :: $i \Rightarrow o$ **where**
Finite(*A*) == *EX* *n*:*nat*. *A eqpoll n*

definition

Card :: $i \Rightarrow o$ **where**
Card(*i*) == (*i* = $|i|$)

notation (*xsymbols*)

eqpoll (**infixl** \approx 50) **and**
lepoll (**infixl** \lesssim 50) **and**
lesspoll (**infixl** \prec 50) **and**
Least (**binder** μ 10)

notation (*HTML output*)

eqpoll (**infixl** \approx 50) **and**
Least (**binder** μ 10)

22.1 The Schroeder-Bernstein Theorem

See Davey and Priestly, page 106

lemma *decomp-bnd-mono*: *bnd-mono*(*X*, $\%W$. $X - g^{''}(Y - f^{''}W)$)
 $\langle proof \rangle$

lemma *Banach-last-equation*:

$g: Y \rightarrow X$
 $\Rightarrow g^{''}(Y - f^{''} \text{ lfp}(X, \%W. X - g^{''}(Y - f^{''}W))) =$

$X = \text{lfp}(X, \% W. X = g^{''}(Y - f^{''}W))$
 $\langle \text{proof} \rangle$

lemma *decomposition*:

$[| f: X \multimap Y; g: Y \multimap X |] ==>$
 $EX\ XA\ XB\ YA\ YB. (XA\ Int\ XB = 0) \ \& \ (XA\ Un\ XB = X) \ \&$
 $(YA\ Int\ YB = 0) \ \& \ (YA\ Un\ YB = Y) \ \&$
 $f^{''}XA=YA \ \& \ g^{''}YB=XB$

$\langle \text{proof} \rangle$

lemma *schroeder-bernstein*:

$[| f: inj(X,Y); g: inj(Y,X) |] ==> EX\ h. h: bij(X,Y)$
 $\langle \text{proof} \rangle$

lemma *bij-imp-epoll*: $f: bij(A,B) ==> A \approx B$
 $\langle \text{proof} \rangle$

lemmas *epoll-refl* = *id-bij* [*THEN* *bij-imp-epoll*, *standard*, *simp*]

lemma *epoll-sym*: $X \approx Y ==> Y \approx X$
 $\langle \text{proof} \rangle$

lemma *epoll-trans*:

$[| X \approx Y; Y \approx Z |] ==> X \approx Z$
 $\langle \text{proof} \rangle$

lemma *subset-imp-lepoll*: $X \leq Y ==> X \lesssim Y$
 $\langle \text{proof} \rangle$

lemmas *lepoll-refl* = *subset-refl* [*THEN* *subset-imp-lepoll*, *standard*, *simp*]

lemmas *le-imp-lepoll* = *le-imp-subset* [*THEN* *subset-imp-lepoll*, *standard*]

lemma *epoll-imp-lepoll*: $X \approx Y ==> X \lesssim Y$
 $\langle \text{proof} \rangle$

lemma *lepoll-trans*: $[| X \lesssim Y; Y \lesssim Z |] ==> X \lesssim Z$
 $\langle \text{proof} \rangle$

lemma *epollIII*: $[| X \lesssim Y; Y \lesssim X |] ==> X \approx Y$
 $\langle \text{proof} \rangle$

lemma *eqpollE*:

$\llbracket X \approx Y; \llbracket X \lesssim Y; Y \lesssim X \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *eqpoll-iff*: $X \approx Y \iff X \lesssim Y \ \& \ Y \lesssim X$

$\langle proof \rangle$

lemma *lepoll-0-is-0*: $A \lesssim 0 \implies A = 0$

$\langle proof \rangle$

lemmas *empty-lepollI* = *empty-subsetI* [*THEN subset-imp-lepoll, standard*]

lemma *lepoll-0-iff*: $A \lesssim 0 \iff A = 0$

$\langle proof \rangle$

lemma *Un-lepoll-Un*:

$\llbracket A \lesssim B; C \lesssim D; B \text{ Int } D = 0 \rrbracket \implies A \text{ Un } C \lesssim B \text{ Un } D$
 $\langle proof \rangle$

lemmas *eqpoll-0-is-0* = *eqpoll-imp-lepoll* [*THEN lepoll-0-is-0, standard*]

lemma *eqpoll-0-iff*: $A \approx 0 \iff A = 0$

$\langle proof \rangle$

lemma *eqpoll-disjoint-Un*:

$\llbracket A \approx B; C \approx D; A \text{ Int } C = 0; B \text{ Int } D = 0 \rrbracket \implies A \text{ Un } C \approx B \text{ Un } D$
 $\langle proof \rangle$

22.2 lesspoll: contributions by Krzysztof Grabczewski

lemma *lesspoll-not-refl*: $\sim (i \prec i)$

$\langle proof \rangle$

lemma *lesspoll-irrefl* [*elim!*]: $i \prec i \implies P$

$\langle proof \rangle$

lemma *lesspoll-imp-lepoll*: $A \prec B \implies A \lesssim B$

$\langle proof \rangle$

lemma *lepoll-well-ord*: $\llbracket A \lesssim B; \text{well-ord}(B, r) \rrbracket \implies \exists X \text{ s. } \text{well-ord}(A, s)$

$\langle proof \rangle$

lemma *lepoll-iff-leqpoll*: $A \lesssim B \iff A \prec B \mid A \approx B$

$\langle proof \rangle$

lemma *inj-not-surj-succ*:

$\llbracket f : inj(A, succ(m)); f \sim: surj(A, succ(m)) \rrbracket ==> EX f. f:inj(A,m)$
 $\langle proof \rangle$

lemma *lesspoll-trans*:
 $\llbracket X \prec Y; Y \prec Z \rrbracket ==> X \prec Z$
 $\langle proof \rangle$

lemma *lesspoll-trans1*:
 $\llbracket X \lesssim Y; Y \prec Z \rrbracket ==> X \prec Z$
 $\langle proof \rangle$

lemma *lesspoll-trans2*:
 $\llbracket X \prec Y; Y \lesssim Z \rrbracket ==> X \prec Z$
 $\langle proof \rangle$

lemma *Least-equality*:
 $\llbracket P(i); Ord(i); !!x. x < i ==> \sim P(x) \rrbracket ==> (LEAST x. P(x)) = i$
 $\langle proof \rangle$

lemma *LeastI*: $\llbracket P(i); Ord(i) \rrbracket ==> P(LEAST x. P(x))$
 $\langle proof \rangle$

lemma *Least-le*: $\llbracket P(i); Ord(i) \rrbracket ==> (LEAST x. P(x)) le i$
 $\langle proof \rangle$

lemma *less-LeastE*: $\llbracket P(i); i < (LEAST x. P(x)) \rrbracket ==> Q$
 $\langle proof \rangle$

lemma *LeastI2*:
 $\llbracket P(i); Ord(i); !!j. P(j) ==> Q(j) \rrbracket ==> Q(LEAST j. P(j))$
 $\langle proof \rangle$

lemma *Least-0*:
 $\llbracket \sim (EX i. Ord(i) \ \& \ P(i)) \rrbracket ==> (LEAST x. P(x)) = 0$
 $\langle proof \rangle$

lemma *Ord-Least* *[intro,simp,TC]*: $Ord(LEAST x. P(x))$
 $\langle proof \rangle$

lemma *Least-cong*:

$(\lambda y. P(y) \leftrightarrow Q(y)) \implies (\text{LEAST } x. P(x)) = (\text{LEAST } x. Q(x))$
 $\langle \text{proof} \rangle$

lemma *cardinal-cong*: $X \approx Y \implies |X| = |Y|$

$\langle \text{proof} \rangle$

lemma *well-ord-cardinal-epoll*:

$\text{well-ord}(A, r) \implies |A| \approx A$
 $\langle \text{proof} \rangle$

lemmas *Ord-cardinal-epoll* = *well-ord-Memrel* [*THEN well-ord-cardinal-epoll*]

lemma *well-ord-cardinal-eqE*:

$[\text{well-ord}(X, r); \text{well-ord}(Y, s); |X| = |Y|] \implies X \approx Y$
 $\langle \text{proof} \rangle$

lemma *well-ord-cardinal-epoll-iff*:

$[\text{well-ord}(X, r); \text{well-ord}(Y, s)] \implies |X| = |Y| \leftrightarrow X \approx Y$
 $\langle \text{proof} \rangle$

lemma *Ord-cardinal-le*: $\text{Ord}(i) \implies |i| \text{ le } i$

$\langle \text{proof} \rangle$

lemma *Card-cardinal-eq*: $\text{Card}(K) \implies |K| = K$

$\langle \text{proof} \rangle$

lemma *CardI*: $[\text{Ord}(i); \lambda j. j < i \implies \sim(j \approx i)] \implies \text{Card}(i)$

$\langle \text{proof} \rangle$

lemma *Card-is-Ord*: $\text{Card}(i) \implies \text{Ord}(i)$

$\langle \text{proof} \rangle$

lemma *Card-cardinal-le*: $\text{Card}(K) \implies K \text{ le } |K|$

$\langle \text{proof} \rangle$

lemma *Ord-cardinal* [*simp,intro!*]: $\text{Ord}(|A|)$

$\langle \text{proof} \rangle$

lemma *Card-iff-initial*: $\text{Card}(K) <-> \text{Ord}(K) \ \& \ (\text{ALL } j. j < K \longrightarrow \sim j \approx K)$
 $\langle \text{proof} \rangle$

lemma *lt-Card-imp-lesspoll*: $[\mid \text{Card}(a); i < a \mid] \implies i \prec a$
 $\langle \text{proof} \rangle$

lemma *Card-0*: $\text{Card}(0)$
 $\langle \text{proof} \rangle$

lemma *Card-Un*: $[\mid \text{Card}(K); \text{Card}(L) \mid] \implies \text{Card}(K \text{ Un } L)$
 $\langle \text{proof} \rangle$

lemma *Card-cardinal*: $\text{Card}(|A|)$
 $\langle \text{proof} \rangle$

lemma *cardinal-eq-lemma*: $[\mid |i| \text{ le } j; j \text{ le } i \mid] \implies |j| = |i|$
 $\langle \text{proof} \rangle$

lemma *cardinal-mono*: $i \text{ le } j \implies |i| \text{ le } |j|$
 $\langle \text{proof} \rangle$

lemma *cardinal-lt-imp-lt*: $[\mid |i| < |j|; \text{Ord}(i); \text{Ord}(j) \mid] \implies i < j$
 $\langle \text{proof} \rangle$

lemma *Card-lt-imp-lt*: $[\mid |i| < K; \text{Ord}(i); \text{Card}(K) \mid] \implies i < K$
 $\langle \text{proof} \rangle$

lemma *Card-lt-iff*: $[\mid \text{Ord}(i); \text{Card}(K) \mid] \implies (|i| < K) <-> (i < K)$
 $\langle \text{proof} \rangle$

lemma *Card-le-iff*: $[\mid \text{Ord}(i); \text{Card}(K) \mid] \implies (K \text{ le } |i|) <-> (K \text{ le } i)$
 $\langle \text{proof} \rangle$

lemma *well-ord-lepoll-imp-Card-le*:
 $[\mid \text{well-ord}(B, r); A \lesssim B \mid] \implies |A| \text{ le } |B|$
 $\langle \text{proof} \rangle$

lemma *lepoll-cardinal-le*: $[\mid A \lesssim i; \text{Ord}(i) \mid] \implies |A| \text{ le } i$
 $\langle \text{proof} \rangle$

lemma *lepoll-Ord-imp-eqpoll*: $[\mid A \lesssim i; \text{Ord}(i) \mid] \implies |A| \approx A$
 $\langle \text{proof} \rangle$

lemma *lesspoll-imp-epoll*: $[| A \prec i; \text{Ord}(i) |] \implies |A| \approx A$
 $\langle \text{proof} \rangle$

lemma *cardinal-subset-Ord*: $[| A \leq i; \text{Ord}(i) |] \implies |A| \leq i$
 $\langle \text{proof} \rangle$

22.3 The finite cardinals

lemma *cons-lepoll-consD*:
 $[| \text{cons}(u,A) \lesssim \text{cons}(v,B); u \sim A; v \sim B |] \implies A \lesssim B$
 $\langle \text{proof} \rangle$

lemma *cons-epoll-consD*: $[| \text{cons}(u,A) \approx \text{cons}(v,B); u \sim A; v \sim B |] \implies A \approx B$
 $\langle \text{proof} \rangle$

lemma *succ-lepoll-succD*: $\text{succ}(m) \lesssim \text{succ}(n) \implies m \lesssim n$
 $\langle \text{proof} \rangle$

lemma *nat-lepoll-imp-le* *[rule-format]*:
 $m:\text{nat} \implies \text{ALL } n:\text{nat}. m \lesssim n \dashv\vdash m \text{ le } n$
 $\langle \text{proof} \rangle$

lemma *nat-epoll-iff*: $[| m:\text{nat}; n:\text{nat} |] \implies m \approx n \iff m = n$
 $\langle \text{proof} \rangle$

lemma *nat-into-Card*:
 $n:\text{nat} \implies \text{Card}(n)$
 $\langle \text{proof} \rangle$

lemmas *cardinal-0* = *nat-0I* *[THEN nat-into-Card, THEN Card-cardinal-eq, iff]*
lemmas *cardinal-1* = *nat-1I* *[THEN nat-into-Card, THEN Card-cardinal-eq, iff]*

lemma *succ-lepoll-natE*: $[| \text{succ}(n) \lesssim n; n:\text{nat} |] \implies P$
 $\langle \text{proof} \rangle$

lemma *n-lesspoll-nat*: $n \in \text{nat} \implies n \prec \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-lepoll-imp-ex-epoll-n*:
 $[| n \in \text{nat}; \text{nat} \lesssim X |] \implies \exists Y. Y \subseteq X \ \& \ n \approx Y$
 $\langle \text{proof} \rangle$

lemma *lepoll-imp-lesspoll-succ*:

$\llbracket A \lesssim m; m:\text{nat} \rrbracket \implies A \prec \text{succ}(m)$
 $\langle \text{proof} \rangle$

lemma *lesspoll-succ-imp-lepoll*:

$\llbracket A \prec \text{succ}(m); m:\text{nat} \rrbracket \implies A \lesssim m$
 $\langle \text{proof} \rangle$

lemma *lesspoll-succ-iff*: $m:\text{nat} \implies A \prec \text{succ}(m) \iff A \lesssim m$

$\langle \text{proof} \rangle$

lemma *lepoll-succ-disj*: $\llbracket A \lesssim \text{succ}(m); m:\text{nat} \rrbracket \implies A \lesssim m \mid A \approx \text{succ}(m)$

$\langle \text{proof} \rangle$

lemma *lesspoll-cardinal-lt*: $\llbracket A \prec i; \text{Ord}(i) \rrbracket \implies |A| < i$

$\langle \text{proof} \rangle$

22.4 The first infinite cardinal: Omega, or nat

lemma *lt-not-lepoll*: $\llbracket n < i; n:\text{nat} \rrbracket \implies \sim i \lesssim n$

$\langle \text{proof} \rangle$

lemma *Ord-nat-eqpoll-iff*: $\llbracket \text{Ord}(i); n:\text{nat} \rrbracket \implies i \approx n \iff i = n$

$\langle \text{proof} \rangle$

lemma *Card-nat*: $\text{Card}(\text{nat})$

$\langle \text{proof} \rangle$

lemma *nat-le-cardinal*: $\text{nat} \text{ le } i \implies \text{nat} \text{ le } |i|$

$\langle \text{proof} \rangle$

22.5 Towards Cardinal Arithmetic

lemma *cons-lepoll-cong*:

$\llbracket A \lesssim B; b \sim: B \rrbracket \implies \text{cons}(a, A) \lesssim \text{cons}(b, B)$
 $\langle \text{proof} \rangle$

lemma *cons-eqpoll-cong*:

$\llbracket A \approx B; a \sim: A; b \sim: B \rrbracket \implies \text{cons}(a, A) \approx \text{cons}(b, B)$
 $\langle \text{proof} \rangle$

lemma *cons-lepoll-cons-iff*:

$\llbracket a \sim: A; b \sim: B \rrbracket \implies \text{cons}(a, A) \lesssim \text{cons}(b, B) \iff A \lesssim B$
 $\langle \text{proof} \rangle$

lemma *cons-eqpoll-cons-iff*:

$\llbracket a \sim: A; b \sim: B \rrbracket \implies \text{cons}(a, A) \approx \text{cons}(b, B) \iff A \approx B$
 $\langle \text{proof} \rangle$

lemma *singleton-epoll-1*: $\{a\} \approx 1$

$\langle proof \rangle$

lemma *cardinal-singleton*: $|\{a\}| = 1$

$\langle proof \rangle$

lemma *not-0-is-lepoll-1*: $A \sim 0 \implies 1 \lesssim A$

$\langle proof \rangle$

lemma *succ-epoll-cong*: $A \approx B \implies succ(A) \approx succ(B)$

$\langle proof \rangle$

lemma *sum-epoll-cong*: $[| A \approx C; B \approx D |] \implies A+B \approx C+D$

$\langle proof \rangle$

lemma *prod-epoll-cong*:

$[| A \approx C; B \approx D |] \implies A*B \approx C*D$

$\langle proof \rangle$

lemma *inj-disjoint-epoll*:

$[| f: inj(A,B); A \text{ Int } B = 0 |] \implies A \text{ Un } (B - range(f)) \approx B$

$\langle proof \rangle$

22.6 Lemmas by Krzysztof Grabczewski

lemma *Diff-sing-lepoll*:

$[| a:A; A \lesssim succ(n) |] \implies A - \{a\} \lesssim n$

$\langle proof \rangle$

lemma *lepoll-Diff-sing*:

$[| succ(n) \lesssim A |] \implies n \lesssim A - \{a\}$

$\langle proof \rangle$

lemma *Diff-sing-epoll*: $[| a:A; A \approx succ(n) |] \implies A - \{a\} \approx n$

$\langle proof \rangle$

lemma *lepoll-1-is-sing*: $[| A \lesssim 1; a:A |] \implies A = \{a\}$

$\langle proof \rangle$

lemma *Un-lepoll-sum*: $A \text{ Un } B \lesssim A+B$

$\langle proof \rangle$

lemma *well-ord-Un*:

$[| well-ord(X,R); well-ord(Y,S) |] \implies \exists X \ T. well-ord(X \text{ Un } Y, T)$

$\langle proof \rangle$

lemma *disj-Un-eqpoll-sum*: $A \text{ Int } B = 0 \implies A \text{ Un } B \approx A + B$
 $\langle proof \rangle$

22.7 Finite and infinite sets

lemma *Finite-0* [*simp*]: $Finite(0)$
 $\langle proof \rangle$

lemma *lepoll-nat-imp-Finite*: $[| A \lesssim n; n:nat |] \implies Finite(A)$
 $\langle proof \rangle$

lemma *lesspoll-nat-is-Finite*:
 $A \prec nat \implies Finite(A)$
 $\langle proof \rangle$

lemma *lepoll-Finite*:
 $[| Y \lesssim X; Finite(X) |] \implies Finite(Y)$
 $\langle proof \rangle$

lemmas *subset-Finite* = *subset-imp-lepoll* [*THEN lepoll-Finite, standard*]

lemma *Finite-Int*: $Finite(A) \mid Finite(B) \implies Finite(A \text{ Int } B)$
 $\langle proof \rangle$

lemmas *Finite-Diff* = *Diff-subset* [*THEN subset-Finite, standard*]

lemma *Finite-cons*: $Finite(x) \implies Finite(cons(y,x))$
 $\langle proof \rangle$

lemma *Finite-succ*: $Finite(x) \implies Finite(succ(x))$
 $\langle proof \rangle$

lemma *Finite-cons-iff* [*iff*]: $Finite(cons(y,x)) \iff Finite(x)$
 $\langle proof \rangle$

lemma *Finite-succ-iff* [*iff*]: $Finite(succ(x)) \iff Finite(x)$
 $\langle proof \rangle$

lemma *nat-le-infinite-Ord*:
 $[| Ord(i); \sim Finite(i) |] \implies nat \text{ le } i$
 $\langle proof \rangle$

lemma *Finite-imp-well-ord*:
 $Finite(A) \implies \exists r. \text{ well-ord}(A,r)$
 $\langle proof \rangle$

lemma *succ-lepoll-imp-not-empty*: $\text{succ}(x) \lesssim y \implies y \neq 0$
 $\langle \text{proof} \rangle$

lemma *eqpoll-succ-imp-not-empty*: $x \approx \text{succ}(n) \implies x \neq 0$
 $\langle \text{proof} \rangle$

lemma *Finite-Fin-lemma* [rule-format]:
 $n \in \text{nat} \implies \forall A. (A \approx n \ \& \ A \subseteq X) \dashrightarrow A \in \text{Fin}(X)$
 $\langle \text{proof} \rangle$

lemma *Finite-Fin*: $[| \text{Finite}(A); A \subseteq X |] \implies A \in \text{Fin}(X)$
 $\langle \text{proof} \rangle$

lemma *eqpoll-imp-Finite-iff*: $A \approx B \implies \text{Finite}(A) <-> \text{Finite}(B)$
 $\langle \text{proof} \rangle$

lemma *Fin-lemma* [rule-format]: $n: \text{nat} \implies \text{ALL } A. A \approx n \dashrightarrow A : \text{Fin}(A)$
 $\langle \text{proof} \rangle$

lemma *Finite-into-Fin*: $\text{Finite}(A) \implies A : \text{Fin}(A)$
 $\langle \text{proof} \rangle$

lemma *Fin-into-Finite*: $A : \text{Fin}(U) \implies \text{Finite}(A)$
 $\langle \text{proof} \rangle$

lemma *Finite-Fin-iff*: $\text{Finite}(A) <-> A : \text{Fin}(A)$
 $\langle \text{proof} \rangle$

lemma *Finite-Un*: $[| \text{Finite}(A); \text{Finite}(B) |] \implies \text{Finite}(A \text{ Un } B)$
 $\langle \text{proof} \rangle$

lemma *Finite-Un-iff* [simp]: $\text{Finite}(A \text{ Un } B) <-> (\text{Finite}(A) \ \& \ \text{Finite}(B))$
 $\langle \text{proof} \rangle$

The converse must hold too.

lemma *Finite-Union*: $[| \text{ALL } y:X. \text{Finite}(y); \text{Finite}(X) |] \implies \text{Finite}(\text{Union}(X))$
 $\langle \text{proof} \rangle$

lemma *Finite-induct* [case-names 0 cons, induct set: Finite]:
 $[| \text{Finite}(A); P(0);$
 $\quad !! x B. \quad [| \text{Finite}(B); x \sim: B; P(B) |] \implies P(\text{cons}(x, B)) |]$
 $\implies P(A)$
 $\langle \text{proof} \rangle$

lemma *Diff-sing-Finite*: $\text{Finite}(A - \{a\}) \implies \text{Finite}(A)$
 $\langle \text{proof} \rangle$

lemma *Diff-Finite* [rule-format]: $Finite(B) ==> Finite(A-B) --> Finite(A)$
 <proof>

lemma *Finite-RepFun*: $Finite(A) ==> Finite(RepFun(A,f))$
 <proof>

lemma *Finite-RepFun-iff-lemma* [rule-format]:
 $[| Finite(x); !!x y. f(x)=f(y) ==> x=y |]$
 $==> \forall A. x = RepFun(A,f) --> Finite(A)$
 <proof>

I don't know why, but if the premise is expressed using meta-connectives then the simplifier cannot prove it automatically in conditional rewriting.

lemma *Finite-RepFun-iff*:
 $(\forall x y. f(x)=f(y) --> x=y) ==> Finite(RepFun(A,f)) <-> Finite(A)$
 <proof>

lemma *Finite-Pow*: $Finite(A) ==> Finite(Pow(A))$
 <proof>

lemma *Finite-Pow-imp-Finite*: $Finite(Pow(A)) ==> Finite(A)$
 <proof>

lemma *Finite-Pow-iff* [iff]: $Finite(Pow(A)) <-> Finite(A)$
 <proof>

lemma *nat-wf-on-converse-Memrel*: $n:nat ==> wf[n](converse(Memrel(n)))$
 <proof>

lemma *nat-well-ord-converse-Memrel*: $n:nat ==> well-ord(n, converse(Memrel(n)))$
 <proof>

lemma *well-ord-converse*:
 $[| well-ord(A,r);$
 $well-ord(ordertype(A,r), converse(Memrel(ordertype(A, r)))) |]$
 $==> well-ord(A, converse(r))$
 <proof>

lemma *ordertype-eq-n*:
 $[| well-ord(A,r); A \approx n; n:nat |] ==> ordertype(A,r)=n$
 <proof>

lemma *Finite-well-ord-converse*:
 $[| Finite(A); well-ord(A,r) |] ==> well-ord(A, converse(r))$

$\langle proof \rangle$

lemma *nat-into-Finite*: $n:nat ==> Finite(n)$
 $\langle proof \rangle$

lemma *nat-not-Finite*: $\sim Finite(nat)$
 $\langle proof \rangle$

$\langle ML \rangle$

end

23 Univ: The Cumulative Hierarchy and a Small Universe for Recursive Types

theory *Univ* **imports** *Epsilon Cardinal* **begin**

definition

$Vfrom :: [i,i] => i$ **where**
 $Vfrom(A,i) == transrec(i, \%x f. A \ Un \ (\bigcup y \in x. Pow(f'y)))$

abbreviation

$Vset :: i => i$ **where**
 $Vset(x) == Vfrom(0,x)$

definition

$Vrec :: [i, [i,i] => i] => i$ **where**
 $Vrec(a,H) == transrec(rank(a), \%x g. lam z: Vset(succ(x)).$
 $H(z, lam w: Vset(x). g'rank(w)'w)) ' a$

definition

$Vrecursor :: [[i,i] => i, i] => i$ **where**
 $Vrecursor(H,a) == transrec(rank(a), \%x g. lam z: Vset(succ(x)).$
 $H(lam w: Vset(x). g'rank(w)'w, z)) ' a$

definition

$univ :: i => i$ **where**
 $univ(A) == Vfrom(A,nat)$

23.1 Immediate Consequences of the Definition of $Vfrom(A, i)$

NOT SUITABLE FOR REWRITING – RECURSIVE!

lemma *Vfrom*: $Vfrom(A,i) = A \ Un \ (\bigcup j \in i. Pow(Vfrom(A,j)))$
 $\langle proof \rangle$

23.1.1 Monotonicity

lemma *Vfrom-mono* [rule-format]:

$$A \leq B \implies \forall j. i \leq j \longrightarrow Vfrom(A, i) \leq Vfrom(B, j)$$

<proof>

lemma *VfromI*: $[[a \in Vfrom(A, j); j < i]] \implies a \in Vfrom(A, i)$

<proof>

23.1.2 A fundamental equality: Vfrom does not require ordinals!

lemma *Vfrom-rank-subset1*: $Vfrom(A, x) \leq Vfrom(A, rank(x))$

<proof>

lemma *Vfrom-rank-subset2*: $Vfrom(A, rank(x)) \leq Vfrom(A, x)$

<proof>

lemma *Vfrom-rank-eq*: $Vfrom(A, rank(x)) = Vfrom(A, x)$

<proof>

23.2 Basic Closure Properties

lemma *zero-in-Vfrom*: $y : x \implies 0 \in Vfrom(A, x)$

<proof>

lemma *i-subset-Vfrom*: $i \leq Vfrom(A, i)$

<proof>

lemma *A-subset-Vfrom*: $A \leq Vfrom(A, i)$

<proof>

lemmas *A-into-Vfrom* = *A-subset-Vfrom* [THEN subsetD]

lemma *subset-mem-Vfrom*: $a \leq Vfrom(A, i) \implies a \in Vfrom(A, succ(i))$

<proof>

23.2.1 Finite sets and ordered pairs

lemma *singleton-in-Vfrom*: $a \in Vfrom(A, i) \implies \{a\} \in Vfrom(A, succ(i))$

<proof>

lemma *doubleton-in-Vfrom*:

$$[[a \in Vfrom(A, i); b \in Vfrom(A, i)]] \implies \{a, b\} \in Vfrom(A, succ(i))$$

<proof>

lemma *Pair-in-Vfrom*:

$$[[a \in Vfrom(A, i); b \in Vfrom(A, i)]] \implies \langle a, b \rangle \in Vfrom(A, succ(succ(i)))$$

<proof>

lemma *succ-in-Vfrom*: $a \leq Vfrom(A, i) \implies succ(a) \in Vfrom(A, succ(succ(i)))$

$\langle proof \rangle$

23.3 0, Successor and Limit Equations for $Vfrom$

lemma $Vfrom-0$: $Vfrom(A, 0) = A$

$\langle proof \rangle$

lemma $Vfrom-succ-lemma$: $Ord(i) ==> Vfrom(A, succ(i)) = A \text{ Un } Pow(Vfrom(A, i))$

$\langle proof \rangle$

lemma $Vfrom-succ$: $Vfrom(A, succ(i)) = A \text{ Un } Pow(Vfrom(A, i))$

$\langle proof \rangle$

lemma $Vfrom-Union$: $y:X ==> Vfrom(A, Union(X)) = (\bigcup y \in X. Vfrom(A, y))$

$\langle proof \rangle$

23.4 $Vfrom$ applied to Limit Ordinals

lemma $Limit-Vfrom-eq$:

$Limit(i) ==> Vfrom(A, i) = (\bigcup y \in i. Vfrom(A, y))$

$\langle proof \rangle$

lemma $Limit-VfromE$:

$[[a \in Vfrom(A, i); \sim R ==> Limit(i);$
 $!!x. [[x < i; a \in Vfrom(A, x)]] ==> R$
 $]] ==> R$

$\langle proof \rangle$

lemma $singleton-in-VLimit$:

$[[a \in Vfrom(A, i); Limit(i)]] ==> \{a\} \in Vfrom(A, i)$

$\langle proof \rangle$

lemmas $Vfrom-UnI1 =$

$Un-upper1 [THEN subset-refl [THEN Vfrom-mono, THEN subsetD], standard]$

lemmas $Vfrom-UnI2 =$

$Un-upper2 [THEN subset-refl [THEN Vfrom-mono, THEN subsetD], standard]$

Hard work is finding a single $j:i$ such that $a, b_i = Vfrom(A, j)$

lemma $doubleton-in-VLimit$:

$[[a \in Vfrom(A, i); b \in Vfrom(A, i); Limit(i)]] ==> \{a, b\} \in Vfrom(A, i)$

$\langle proof \rangle$

lemma $Pair-in-VLimit$:

$[[a \in Vfrom(A, i); b \in Vfrom(A, i); Limit(i)]] ==> \langle a, b \rangle \in Vfrom(A, i) \langle proof \rangle$

lemma $product-VLimit$: $Limit(i) ==> Vfrom(A, i) * Vfrom(A, i) \leq Vfrom(A, i)$

$\langle proof \rangle$

lemmas *Sigma-subset-VLimit* =
subset-trans [*OF Sigma-mono product-VLimit*]

lemmas *nat-subset-VLimit* =
subset-trans [*OF nat-le-Limit* [*THEN le-imp-subset*] *i-subset-Vfrom*]

lemma *nat-into-VLimit*: [$n: \text{nat}; \text{Limit}(i)$] $\implies n \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

23.4.1 Closure under Disjoint Union

lemmas *zero-in-VLimit* = *Limit-has-0* [*THEN ltD*, *THEN zero-in-Vfrom*, *standard*]

lemma *one-in-VLimit*: $\text{Limit}(i) \implies 1 \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemma *Inl-in-VLimit*:
 $[a \in \text{Vfrom}(A, i); \text{Limit}(i)] \implies \text{Inl}(a) \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemma *Inr-in-VLimit*:
 $[b \in \text{Vfrom}(A, i); \text{Limit}(i)] \implies \text{Inr}(b) \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemma *sum-VLimit*: $\text{Limit}(i) \implies \text{Vfrom}(C, i) + \text{Vfrom}(C, i) \leq \text{Vfrom}(C, i)$
 $\langle \text{proof} \rangle$

lemmas *sum-subset-VLimit* = *subset-trans* [*OF sum-mono sum-VLimit*]

23.5 Properties assuming *Transset*(*A*)

lemma *Transset-Vfrom*: $\text{Transset}(A) \implies \text{Transset}(\text{Vfrom}(A, i))$
 $\langle \text{proof} \rangle$

lemma *Transset-Vfrom-succ*:
 $\text{Transset}(A) \implies \text{Vfrom}(A, \text{succ}(i)) = \text{Pow}(\text{Vfrom}(A, i))$
 $\langle \text{proof} \rangle$

lemma *Transset-Pair-subset*: [$\langle a, b \rangle \leq C; \text{Transset}(C)$] $\implies a: C \ \& \ b: C$
 $\langle \text{proof} \rangle$

lemma *Transset-Pair-subset-VLimit*:
 $[\langle a, b \rangle \leq \text{Vfrom}(A, i); \text{Transset}(A); \text{Limit}(i)]$
 $\implies \langle a, b \rangle \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemma *Union-in-Vfrom*:
 $[X \in \text{Vfrom}(A, j); \text{Transset}(A)] \implies \text{Union}(X) \in \text{Vfrom}(A, \text{succ}(j))$
 $\langle \text{proof} \rangle$

lemma *Union-in-VLimit:*

$[[X \in Vfrom(A,i); \text{Limit}(i); \text{Transset}(A)]] \implies Union(X) \in Vfrom(A,i)$
 $\langle proof \rangle$

General theorem for membership in $Vfrom(A,i)$ when i is a limit ordinal

lemma *in-VLimit:*

$[[a \in Vfrom(A,i); b \in Vfrom(A,i); \text{Limit}(i);$
 $!!x y j. [[j < i; 1:j; x \in Vfrom(A,j); y \in Vfrom(A,j)]]$
 $\implies EX k. h(x,y) \in Vfrom(A,k) \ \& \ k < i]]$
 $\implies h(a,b) \in Vfrom(A,i) \langle proof \rangle$

23.5.1 Products

lemma *prod-in-Vfrom:*

$[[a \in Vfrom(A,j); b \in Vfrom(A,j); \text{Transset}(A)]]$
 $\implies a*b \in Vfrom(A, succ(succ(succ(j))))$
 $\langle proof \rangle$

lemma *prod-in-VLimit:*

$[[a \in Vfrom(A,i); b \in Vfrom(A,i); \text{Limit}(i); \text{Transset}(A)]]$
 $\implies a*b \in Vfrom(A,i)$
 $\langle proof \rangle$

23.5.2 Disjoint Sums, or Quine Ordered Pairs

lemma *sum-in-Vfrom:*

$[[a \in Vfrom(A,j); b \in Vfrom(A,j); \text{Transset}(A); 1:j]]$
 $\implies a+b \in Vfrom(A, succ(succ(succ(j))))$
 $\langle proof \rangle$

lemma *sum-in-VLimit:*

$[[a \in Vfrom(A,i); b \in Vfrom(A,i); \text{Limit}(i); \text{Transset}(A)]]$
 $\implies a+b \in Vfrom(A,i)$
 $\langle proof \rangle$

23.5.3 Function Space!

lemma *fun-in-Vfrom:*

$[[a \in Vfrom(A,j); b \in Vfrom(A,j); \text{Transset}(A)]] \implies$
 $a \multimap b \in Vfrom(A, succ(succ(succ(succ(j)))))$
 $\langle proof \rangle$

lemma *fun-in-VLimit:*

$[[a \in Vfrom(A,i); b \in Vfrom(A,i); \text{Limit}(i); \text{Transset}(A)]]$
 $\implies a \multimap b \in Vfrom(A,i)$
 $\langle proof \rangle$

lemma *Pow-in-Vfrom:*

$[[a \in Vfrom(A,j); \text{Transset}(A)]] \implies Pow(a) \in Vfrom(A, succ(succ(j)))$

$\langle proof \rangle$

lemma *Pow-in-VLimit*:

$\llbracket a \in Vfrom(A,i); Limit(i); Transset(A) \rrbracket ==> Pow(a) \in Vfrom(A,i)$
 $\langle proof \rangle$

23.6 The Set $Vset(i)$

lemma *Vset*: $Vset(i) = (\bigcup_{j \in i}. Pow(Vset(j)))$
 $\langle proof \rangle$

lemmas *Vset-succ* = *Transset-0* [*THEN Transset-Vfrom-succ, standard*]

lemmas *Transset-Vset* = *Transset-0* [*THEN Transset-Vfrom, standard*]

23.6.1 Characterisation of the elements of $Vset(i)$

lemma *VsetD* [*rule-format*]: $Ord(i) ==> \forall b. b \in Vset(i) \rightarrow rank(b) < i$
 $\langle proof \rangle$

lemma *VsetI-lemma* [*rule-format*]:
 $Ord(i) ==> \forall b. rank(b) \in i \rightarrow b \in Vset(i)$
 $\langle proof \rangle$

lemma *VsetI*: $rank(x) < i ==> x \in Vset(i)$
 $\langle proof \rangle$

Merely a lemma for the next result

lemma *Vset-Ord-rank-iff*: $Ord(i) ==> b \in Vset(i) \leftrightarrow rank(b) < i$
 $\langle proof \rangle$

lemma *Vset-rank-iff* [*simp*]: $b \in Vset(a) \leftrightarrow rank(b) < rank(a)$
 $\langle proof \rangle$

This is $rank(rank(a)) = rank(a)$

declare *Ord-rank* [*THEN rank-of-Ord, simp*]

lemma *rank-Vset*: $Ord(i) ==> rank(Vset(i)) = i$
 $\langle proof \rangle$

lemma *Finite-Vset*: $i \in nat ==> Finite(Vset(i))$
 $\langle proof \rangle$

23.6.2 Reasoning about Sets in Terms of Their Elements' Ranks

lemma *arg-subset-Vset-rank*: $a \leq Vset(rank(a))$
 $\langle proof \rangle$

lemma *Int-Vset-subset*:
 $\llbracket !!i. Ord(i) ==> a \in Vset(i) \leq b \rrbracket ==> a \leq b$
 $\langle proof \rangle$

23.6.3 Set Up an Environment for Simplification

lemma *rank-Inl*: $\text{rank}(a) < \text{rank}(\text{Inl}(a))$
<proof>

lemma *rank-Inr*: $\text{rank}(a) < \text{rank}(\text{Inr}(a))$
<proof>

lemmas *rank-rls* = *rank-Inl rank-Inr rank-pair1 rank-pair2*

23.6.4 Recursion over Vset Levels!

NOT SUITABLE FOR REWRITING: recursive!

lemma *Vrec*: $\text{Vrec}(a, H) = H(a, \text{lam } x: \text{Vset}(\text{rank}(a)). \text{Vrec}(x, H))$
<proof>

This form avoids giant explosions in proofs. NOTE USE OF ==

lemma *def-Vrec*:
 $[[\text{!!}x. h(x) == \text{Vrec}(x, H)]] ==>$
 $h(a) = H(a, \text{lam } x: \text{Vset}(\text{rank}(a)). h(x))$
<proof>

NOT SUITABLE FOR REWRITING: recursive!

lemma *Vrecursor*:
 $\text{Vrecursor}(H, a) = H(\text{lam } x: \text{Vset}(\text{rank}(a)). \text{Vrecursor}(H, x), a)$
<proof>

This form avoids giant explosions in proofs. NOTE USE OF ==

lemma *def-Vrecursor*:
 $h == \text{Vrecursor}(H) ==> h(a) = H(\text{lam } x: \text{Vset}(\text{rank}(a)). h(x), a)$
<proof>

23.7 The Datatype Universe: *univ*(A)

lemma *univ-mono*: $A \leq B ==> \text{univ}(A) \leq \text{univ}(B)$
<proof>

lemma *Transset-univ*: $\text{Transset}(A) ==> \text{Transset}(\text{univ}(A))$
<proof>

23.7.1 The Set *univ*(A) as a Limit

lemma *univ-eq-UN*: $\text{univ}(A) = (\bigcup i \in \text{nat}. \text{Vfrom}(A, i))$
<proof>

lemma *subset-univ-eq-Int*: $c \leq \text{univ}(A) ==> c = (\bigcup i \in \text{nat}. c \text{ Int } \text{Vfrom}(A, i))$
<proof>

lemma *univ-Int-Vfrom-subset*:

$$\begin{aligned} & [| a \leq \text{univ}(X); \\ & \quad !!i. i:\text{nat} ==> a \text{ Int } \text{Vfrom}(X,i) \leq b |] \\ & ==> a \leq b \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *univ-Int-Vfrom-eq*:

$$\begin{aligned} & [| a \leq \text{univ}(X); \quad b \leq \text{univ}(X); \\ & \quad !!i. i:\text{nat} ==> a \text{ Int } \text{Vfrom}(X,i) = b \text{ Int } \text{Vfrom}(X,i) \\ & |] ==> a = b \\ & \langle \text{proof} \rangle \end{aligned}$$

23.8 Closure Properties for $\text{univ}(A)$

lemma *zero-in-univ*: $0 \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *zero-subset-univ*: $\{0\} \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *A-subset-univ*: $A \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemmas *A-into-univ* = *A-subset-univ* [*THEN subsetD, standard*]

23.8.1 Closure under Unordered and Ordered Pairs

lemma *singleton-in-univ*: $a: \text{univ}(A) ==> \{a\} \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *doubleton-in-univ*:

$$[| a: \text{univ}(A); \quad b: \text{univ}(A) |] ==> \{a,b\} \in \text{univ}(A)$$
 $\langle \text{proof} \rangle$

lemma *Pair-in-univ*:

$$[| a: \text{univ}(A); \quad b: \text{univ}(A) |] ==> \langle a,b \rangle \in \text{univ}(A)$$
 $\langle \text{proof} \rangle$

lemma *Union-in-univ*:

$$[| X: \text{univ}(A); \quad \text{Transset}(A) |] ==> \text{Union}(X) \in \text{univ}(A)$$
 $\langle \text{proof} \rangle$

lemma *product-univ*: $\text{univ}(A) * \text{univ}(A) \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

23.8.2 The Natural Numbers

lemma *nat-subset-univ*: $\text{nat} \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

$\text{n:nat} ==_i \text{n:univ}(A)$

lemmas *nat-into-univ* = *nat-subset-univ* [*THEN subsetD, standard*]

23.8.3 Instances for 1 and 2

lemma *one-in-univ*: $1 \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

unused!

lemma *two-in-univ*: $2 \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *bool-subset-univ*: $\text{bool} \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemmas *bool-into-univ* = *bool-subset-univ* [*THEN subsetD, standard*]

23.8.4 Closure under Disjoint Union

lemma *Inl-in-univ*: $a: \text{univ}(A) \implies \text{Inl}(a) \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *Inr-in-univ*: $b: \text{univ}(A) \implies \text{Inr}(b) \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *sum-univ*: $\text{univ}(C) + \text{univ}(C) \leq \text{univ}(C)$
 $\langle \text{proof} \rangle$

lemmas *sum-subset-univ* = *subset-trans* [*OF sum-mono sum-univ*]

lemma *Sigma-subset-univ*:
 $[[A \subseteq \text{univ}(D); \bigwedge x. x \in A \implies B(x) \subseteq \text{univ}(D)]] \implies \text{Sigma}(A, B) \subseteq \text{univ}(D)$
 $\langle \text{proof} \rangle$

23.9 Finite Branching Closure Properties

23.9.1 Closure under Finite Powerset

lemma *Fin-Vfrom-lemma*:
 $[[b: \text{Fin}(\text{Vfrom}(A, i)); \text{Limit}(i)]] \implies \exists j. b \leq \text{Vfrom}(A, j) \ \& \ j < i$
 $\langle \text{proof} \rangle$

lemma *Fin-VLimit*: $\text{Limit}(i) \implies \text{Fin}(\text{Vfrom}(A, i)) \leq \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemmas *Fin-subset-VLimit* = *subset-trans* [*OF Fin-mono Fin-VLimit*]

lemma *Fin-univ*: $\text{Fin}(\text{univ}(A)) \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

23.9.2 Closure under Finite Powers: Functions from a Natural Number

lemma *nat-fun-VLimit*:

$[[n: \text{nat}; \text{Limit}(i)]] \implies n \rightarrow V\text{from}(A, i) \leq V\text{from}(A, i)$
 $\langle \text{proof} \rangle$

lemmas *nat-fun-subset-VLimit* = *subset-trans* [OF *Pi-mono nat-fun-VLimit*]

lemma *nat-fun-univ*: $n: \text{nat} \implies n \rightarrow \text{univ}(A) \leq \text{univ}(A)$

$\langle \text{proof} \rangle$

23.9.3 Closure under Finite Function Space

General but seldom-used version; normally the domain is fixed

lemma *FiniteFun-VLimit1*:

$\text{Limit}(i) \implies V\text{from}(A, i) -||> V\text{from}(A, i) \leq V\text{from}(A, i)$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-univ1*: $\text{univ}(A) -||> \text{univ}(A) \leq \text{univ}(A)$

$\langle \text{proof} \rangle$

Version for a fixed domain

lemma *FiniteFun-VLimit*:

$[[W \leq V\text{from}(A, i); \text{Limit}(i)]] \implies W -||> V\text{from}(A, i) \leq V\text{from}(A, i)$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-univ*:

$W \leq \text{univ}(A) \implies W -||> \text{univ}(A) \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-in-univ*:

$[[f: W -||> \text{univ}(A); W \leq \text{univ}(A)]] \implies f \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

Remove $\text{!} =$ from the rule above

lemmas *FiniteFun-in-univ'* = *FiniteFun-in-univ* [OF - *subsetI*]

23.10 * For QUniv. Properties of Vfrom analogous to the "take-lemma" *

Intersecting $a*b$ with $V\text{from}...$

This version says a, b exist one level down, in the smaller set $V\text{from}(X, i)$

lemma *doubleton-in-Vfrom-D*:

$[[\{a, b\} \in V\text{from}(X, \text{succ}(i)); \text{Transset}(X)]] \implies a \in V\text{from}(X, i) \ \& \ b \in V\text{from}(X, i)$
 $\langle \text{proof} \rangle$

This weaker version says a, b exist at the same level

lemmas *Vfrom-doubleton-D = Transset-Vfrom [THEN Transset-doubleton-D, standard]*

lemma *Pair-in-Vfrom-D:*

$$[| <a,b> \in Vfrom(X, succ(i)); Transset(X) |]$$

$$\implies a \in Vfrom(X, i) \ \& \ b \in Vfrom(X, i)$$

$$\langle proof \rangle$$

lemma *product-Int-Vfrom-subset:*

$$Transset(X) \implies$$

$$(a*b) \text{ Int } Vfrom(X, succ(i)) \leq (a \text{ Int } Vfrom(X, i)) * (b \text{ Int } Vfrom(X, i))$$

$$\langle proof \rangle$$

$\langle ML \rangle$

end

24 QUniv: A Small Universe for Lazy Recursive Types

theory *QUniv imports Univ QPair begin*

rep-datatype

elimination *sumE*
induction *TrueI*
case-eqns *case-Inl case-Inr*

rep-datatype

elimination *qsumE*
induction *TrueI*
case-eqns *qcase-QInl qcase-QInr*

definition

$$quniv :: i \implies i \text{ where}$$

$$quniv(A) == Pow(univ(eclose(A)))$$

24.1 Properties involving Transset and Sum

lemma *Transset-includes-summands:*

$$[| Transset(C); A+B \leq C |] \implies A \leq C \ \& \ B \leq C$$

$$\langle proof \rangle$$

lemma *Transset-sum-Int-subset*:

$\text{Transset}(C) \implies (A+B) \text{ Int } C \leq (A \text{ Int } C) + (B \text{ Int } C)$
 $\langle \text{proof} \rangle$

24.2 Introduction and Elimination Rules

lemma *qunivI*: $X \leq \text{univ}(\text{eclose}(A)) \implies X : \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemma *qunivD*: $X : \text{quniv}(A) \implies X \leq \text{univ}(\text{eclose}(A))$
 $\langle \text{proof} \rangle$

lemma *quniv-mono*: $A \leq B \implies \text{quniv}(A) \leq \text{quniv}(B)$
 $\langle \text{proof} \rangle$

24.3 Closure Properties

lemma *univ-eclose-subset-quniv*: $\text{univ}(\text{eclose}(A)) \leq \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemma *univ-subset-quniv*: $\text{univ}(A) \leq \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemmas *univ-into-quniv* = *univ-subset-quniv* [THEN subsetD, standard]

lemma *Pow-univ-subset-quniv*: $\text{Pow}(\text{univ}(A)) \leq \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemmas *univ-subset-into-quniv* =
PowI [THEN *Pow-univ-subset-quniv* [THEN subsetD], standard]

lemmas *zero-in-quniv* = *zero-in-univ* [THEN *univ-into-quniv*, standard]

lemmas *one-in-quniv* = *one-in-univ* [THEN *univ-into-quniv*, standard]

lemmas *two-in-quniv* = *two-in-univ* [THEN *univ-into-quniv*, standard]

lemmas *A-subset-quniv* = *subset-trans* [OF *A-subset-univ* *univ-subset-quniv*]

lemmas *A-into-quniv* = *A-subset-quniv* [THEN subsetD, standard]

lemma *QPair-subset-univ*:

$[| a \leq \text{univ}(A); b \leq \text{univ}(A) |] \implies \langle a; b \rangle \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

24.4 Quine Disjoint Sum

lemma *QInl-subset-univ*: $a \leq \text{univ}(A) \implies \text{QInl}(a) \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemmas *naturals-subset-nat* =
Ord-nat [*THEN Ord-is-Transset, unfolded Transset-def, THEN bspec, standard*]

lemmas *naturals-subset-univ* =
subset-trans [*OF naturals-subset-nat nat-subset-univ*]

lemma *QInr-subset-univ*: $a \leq \text{univ}(A) \implies \text{QInr}(a) \leq \text{univ}(A)$
 $\langle \text{proof} \rangle$

24.5 Closure for Quine-Inspired Products and Sums

lemma *QPair-in-quniv*:
 $[a : \text{quniv}(A); b : \text{quniv}(A)] \implies \langle a; b \rangle : \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemma *QSigma-quniv*: $\text{quniv}(A) \lt * \text{quniv}(A) \leq \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemmas *QSigma-subset-quniv* = *subset-trans* [*OF QSigma-mono QSigma-quniv*]

lemma *quniv-QPair-D*:
 $\langle a; b \rangle : \text{quniv}(A) \implies a : \text{quniv}(A) \ \& \ b : \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemmas *quniv-QPair-E* = *quniv-QPair-D* [*THEN conjE, standard*]

lemma *quniv-QPair-iff*: $\langle a; b \rangle : \text{quniv}(A) \iff a : \text{quniv}(A) \ \& \ b : \text{quniv}(A)$
 $\langle \text{proof} \rangle$

24.6 Quine Disjoint Sum

lemma *QInl-in-quniv*: $a : \text{quniv}(A) \implies \text{QInl}(a) : \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemma *QInr-in-quniv*: $b : \text{quniv}(A) \implies \text{QInr}(b) : \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemma *qsum-quniv*: $\text{quniv}(C) \lt + \text{quniv}(C) \leq \text{quniv}(C)$
 $\langle \text{proof} \rangle$

lemmas *qsum-subset-quniv* = *subset-trans* [*OF qsum-mono qsum-quniv*]

24.7 The Natural Numbers

lemmas *nat-subset-quniv* = *subset-trans* [*OF nat-subset-univ univ-subset-quniv*]

lemmas *nat-into-quniv* = *nat-subset-quniv* [*THEN subsetD, standard*]

lemmas *bool-subset-quniv* = *subset-trans* [*OF bool-subset-univ univ-subset-quniv*]

lemmas *bool-into-quniv* = *bool-subset-quniv* [*THEN subsetD, standard*]

lemma *QPair-Int-Vfrom-succ-subset*:

Transset(*X*) ==>
 $\langle a; b \rangle \text{ Int Vfrom}(X, \text{succ}(i)) \leq \langle a \text{ Int Vfrom}(X, i); b \text{ Int Vfrom}(X, i) \rangle$
 <proof>

24.8 "Take-Lemma" Rules

lemma *QPair-Int-Vfrom-subset*:

Transset(*X*) ==>
 $\langle a; b \rangle \text{ Int Vfrom}(X, i) \leq \langle a \text{ Int Vfrom}(X, i); b \text{ Int Vfrom}(X, i) \rangle$
 <proof>

lemmas *QPair-Int-Vset-subset-trans* =

subset-trans [*OF Transset-0* [*THEN QPair-Int-Vfrom-subset*] *QPair-mono*]

lemma *QPair-Int-Vset-subset-UN*:

Ord(*i*) ==> $\langle a; b \rangle \text{ Int Vset}(i) \leq (\bigcup_{j \in i}. \langle a \text{ Int Vset}(j); b \text{ Int Vset}(j) \rangle)$
 <proof>

end

25 Datatype-ZF: Datatype and CoDatatype Definitions

theory *Datatype-ZF*

imports *Inductive-ZF Univ QUniv*

uses *Tools/datatype-package.ML*

begin

<ML>

end

definition

$raw-mod :: [i,i] => i$ **where**
 $raw-mod (m, n) ==$
 $transrec(m, \%j f. \text{if } j < n \mid n=0 \text{ then } j \text{ else } f(j\#-n))$

definition

$div :: [i,i] => i$ (**infixl** div 70) **where**
 $m \text{ div } n == raw-div (natify(m), natify(n))$

definition

$mod :: [i,i] => i$ (**infixl** mod 70) **where**
 $m \text{ mod } n == raw-mod (natify(m), natify(n))$

notation (*xsymbols*)

$mult$ (**infixr** $\# \times$ 70)

notation (*HTML output*)

$mult$ (**infixr** $\# \times$ 70)

declare *rec-type* [*simp*]

$nat-0-le$ [*simp*]

lemma *zero-lt-lemma*: $[\mid 0 < k; k \in nat \mid] ==> \exists j \in nat. k = succ(j)$
 $\langle proof \rangle$

lemmas *zero-lt-natE* = *zero-lt-lemma* [*THEN bexE, standard*]

26.1 natify, the Coercion to nat

lemma *pred-succ-eq* [*simp*]: $pred(succ(y)) = y$
 $\langle proof \rangle$

lemma *natify-succ*: $natify(succ(x)) = succ(natify(x))$
 $\langle proof \rangle$

lemma *natify-0* [*simp*]: $natify(0) = 0$
 $\langle proof \rangle$

lemma *natify-non-succ*: $\forall z. x \sim = succ(z) ==> natify(x) = 0$
 $\langle proof \rangle$

lemma *natify-in-nat* [*iff, TC*]: $natify(x) \in nat$
 $\langle proof \rangle$

lemma *natify-ident* [*simp*]: $n \in nat ==> natify(n) = n$
 $\langle proof \rangle$

lemma *natify-eqE*: $[|natify(x) = y; x \in nat|] ==> x=y$
 $\langle proof \rangle$

lemma *natify-idem* [simp]: $natify(natify(x)) = natify(x)$
 $\langle proof \rangle$

lemma *add-natify1* [simp]: $natify(m) \# + n = m \# + n$
 $\langle proof \rangle$

lemma *add-natify2* [simp]: $m \# + natify(n) = m \# + n$
 $\langle proof \rangle$

lemma *mult-natify1* [simp]: $natify(m) \# * n = m \# * n$
 $\langle proof \rangle$

lemma *mult-natify2* [simp]: $m \# * natify(n) = m \# * n$
 $\langle proof \rangle$

lemma *diff-natify1* [simp]: $natify(m) \# - n = m \# - n$
 $\langle proof \rangle$

lemma *diff-natify2* [simp]: $m \# - natify(n) = m \# - n$
 $\langle proof \rangle$

lemma *mod-natify1* [simp]: $natify(m) \bmod n = m \bmod n$
 $\langle proof \rangle$

lemma *mod-natify2* [simp]: $m \bmod natify(n) = m \bmod n$
 $\langle proof \rangle$

lemma *div-natify1* [simp]: $natify(m) \bmod n = m \bmod n$
 $\langle proof \rangle$

lemma *div-natify2* [simp]: $m \bmod natify(n) = m \bmod n$

$\langle proof \rangle$

26.2 Typing rules

lemma *raw-add-type*: $[| m \in nat; n \in nat |] ==> raw-add (m, n) \in nat$
 $\langle proof \rangle$

lemma *add-type* [*iff*, *TC*]: $m \# + n \in nat$
 $\langle proof \rangle$

lemma *raw-mult-type*: $[| m \in nat; n \in nat |] ==> raw-mult (m, n) \in nat$
 $\langle proof \rangle$

lemma *mult-type* [*iff*, *TC*]: $m \# * n \in nat$
 $\langle proof \rangle$

lemma *raw-diff-type*: $[| m \in nat; n \in nat |] ==> raw-diff (m, n) \in nat$
 $\langle proof \rangle$

lemma *diff-type* [*iff*, *TC*]: $m \# - n \in nat$
 $\langle proof \rangle$

lemma *diff-0-eq-0* [*simp*]: $0 \# - n = 0$
 $\langle proof \rangle$

lemma *diff-succ-succ* [*simp*]: $succ(m) \# - succ(n) = m \# - n$
 $\langle proof \rangle$

declare *raw-diff-succ* [*simp del*]

lemma *diff-0* [*simp*]: $m \# - 0 = natify(m)$
 $\langle proof \rangle$

lemma *diff-le-self*: $m \in nat ==> (m \# - n) \leq m$
 $\langle proof \rangle$

26.3 Addition

lemma *add-0-natify* [*simp*]: $0 \# + m = natify(m)$
 $\langle proof \rangle$

lemma *add-succ* [*simp*]: $succ(m) \# + n = succ(m \# + n)$

$\langle proof \rangle$

lemma *add-0*: $m \in nat \implies 0 \# + m = m$
 $\langle proof \rangle$

lemma *add-assoc*: $(m \# + n) \# + k = m \# + (n \# + k)$
 $\langle proof \rangle$

lemma *add-0-right-natify* [*simp*]: $m \# + 0 = natify(m)$
 $\langle proof \rangle$

lemma *add-succ-right* [*simp*]: $m \# + succ(n) = succ(m \# + n)$
 $\langle proof \rangle$

lemma *add-0-right*: $m \in nat \implies m \# + 0 = m$
 $\langle proof \rangle$

lemma *add-commute*: $m \# + n = n \# + m$
 $\langle proof \rangle$

lemma *add-left-commute*: $m \# + (n \# + k) = n \# + (m \# + k)$
 $\langle proof \rangle$

lemmas *add-ac* = *add-assoc add-commute add-left-commute*

lemma *raw-add-left-cancel*:
[$raw-add(k, m) = raw-add(k, n); k \in nat$] $\implies m = n$
 $\langle proof \rangle$

lemma *add-left-cancel-natify*: $k \# + m = k \# + n \implies natify(m) = natify(n)$
 $\langle proof \rangle$

lemma *add-left-cancel*:
[$i = j; i \# + m = j \# + n; m \in nat; n \in nat$] $\implies m = n$
 $\langle proof \rangle$

lemma *add-le-elim1-natify*: $k \# + m \leq k \# + n \implies natify(m) \leq natify(n)$
 $\langle proof \rangle$

lemma *add-le-elim1*: [$k \# + m \leq k \# + n; m \in nat; n \in nat$] $\implies m \leq n$
 $\langle proof \rangle$

lemma *add-lt-elim1-natify*: $k\# + m < k\# + n \implies \text{natify}(m) < \text{natify}(n)$
 $\langle \text{proof} \rangle$

lemma *add-lt-elim1*: $[k\# + m < k\# + n; m \in \text{nat}; n \in \text{nat}] \implies m < n$
 $\langle \text{proof} \rangle$

lemma *zero-less-add*: $[n \in \text{nat}; m \in \text{nat}] \implies 0 < m \# + n \iff (0 < m \mid 0 < n)$
 $\langle \text{proof} \rangle$

26.4 Monotonicity of Addition

lemma *add-lt-mono1*: $[i < j; j \in \text{nat}] \implies i\# + k < j\# + k$
 $\langle \text{proof} \rangle$

strict, in second argument

lemma *add-lt-mono2*: $[i < j; j \in \text{nat}] \implies k\# + i < k\# + j$
 $\langle \text{proof} \rangle$

A [clumsy] way of lifting $\#$ monotonicity to \leq monotonicity

lemma *Ord-lt-mono-imp-le-mono*:
assumes *lt-mono*: $\forall i j. [i < j; j \in \text{nat}] \implies f(i) < f(j)$
and ford: $\forall i. i \in \text{nat} \implies \text{Ord}(f(i))$
and leij: $i \leq j$
and jink: $j \in \text{nat}$
shows $f(i) \leq f(j)$
 $\langle \text{proof} \rangle$

\leq monotonicity, 1st argument

lemma *add-le-mono1*: $[i \leq j; j \in \text{nat}] \implies i\# + k \leq j\# + k$
 $\langle \text{proof} \rangle$

\leq monotonicity, both arguments

lemma *add-le-mono*: $[i \leq j; k \leq l; j \in \text{nat}; l \in \text{nat}] \implies i\# + k \leq j\# + l$
 $\langle \text{proof} \rangle$

Combinations of less-than and less-than-or-equals

lemma *add-lt-le-mono*: $[i < j; k \leq l; j \in \text{nat}; l \in \text{nat}] \implies i\# + k < j\# + l$
 $\langle \text{proof} \rangle$

lemma *add-le-lt-mono*: $[i \leq j; k < l; j \in \text{nat}; l \in \text{nat}] \implies i\# + k < j\# + l$
 $\langle \text{proof} \rangle$

Less-than: in other words, strict in both arguments

lemma *add-lt-mono*: $[i < j; k < l; j \in \text{nat}; l \in \text{nat}] \implies i\# + k < j\# + l$
 $\langle \text{proof} \rangle$

lemma *diff-add-inverse*: $(n \# + m) \# - n = \text{natify}(m)$
 $\langle \text{proof} \rangle$

lemma *diff-add-inverse2*: $(m \# + n) \# - n = \text{natify}(m)$
 $\langle \text{proof} \rangle$

lemma *diff-cancel*: $(k \# + m) \# - (k \# + n) = m \# - n$
 $\langle \text{proof} \rangle$

lemma *diff-cancel2*: $(m \# + k) \# - (n \# + k) = m \# - n$
 $\langle \text{proof} \rangle$

lemma *diff-add-0*: $n \# - (n \# + m) = 0$
 $\langle \text{proof} \rangle$

lemma *pred-0* [*simp*]: $\text{pred}(0) = 0$
 $\langle \text{proof} \rangle$

lemma *eq-succ-imp-eq-m1*: $[[i = \text{succ}(j); i \in \text{nat}]] \implies j = i \# - 1 \ \& \ j \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *pred-Un-distrib*:
 $[[i \in \text{nat}; j \in \text{nat}]] \implies \text{pred}(i \text{ Un } j) = \text{pred}(i) \text{ Un } \text{pred}(j)$
 $\langle \text{proof} \rangle$

lemma *pred-type* [*TC, simp*]:
 $i \in \text{nat} \implies \text{pred}(i) \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-diff-pred*: $[[i \in \text{nat}; j \in \text{nat}]] \implies i \# - \text{succ}(j) = \text{pred}(i \# - j)$
 $\langle \text{proof} \rangle$

lemma *diff-succ-eq-pred*: $i \# - \text{succ}(j) = \text{pred}(i \# - j)$
 $\langle \text{proof} \rangle$

lemma *nat-diff-Un-distrib*:
 $[[i \in \text{nat}; j \in \text{nat}; k \in \text{nat}]] \implies (i \text{ Un } j) \# - k = (i \# - k) \text{ Un } (j \# - k)$
 $\langle \text{proof} \rangle$

lemma *diff-Un-distrib*:
 $[[i \in \text{nat}; j \in \text{nat}]] \implies (i \text{ Un } j) \# - k = (i \# - k) \text{ Un } (j \# - k)$
 $\langle \text{proof} \rangle$

We actually prove $i \# - j \# - k = i \# - (j \# + k)$

lemma *diff-diff-left* [*simplified*]:
 $\text{natify}(i) \# - \text{natify}(j) \# - k = \text{natify}(i) \# - (\text{natify}(j) \# + k)$
 $\langle \text{proof} \rangle$

lemma *eq-add-iff*: $(u \# + m = u \# + n) <-> (0 \# + m = \text{natty}(n))$
 $\langle \text{proof} \rangle$

lemma *less-add-iff*: $(u \# + m < u \# + n) <-> (0 \# + m < \text{natty}(n))$
 $\langle \text{proof} \rangle$

lemma *diff-add-eq*: $((u \# + m) \# - (u \# + n)) = ((0 \# + m) \# - n)$
 $\langle \text{proof} \rangle$

lemma *eq-cong2*: $u = u' ==> (t == u) == (t == u')$
 $\langle \text{proof} \rangle$

lemma *iff-cong2*: $u <-> u' ==> (t == u) == (t == u')$
 $\langle \text{proof} \rangle$

26.5 Multiplication

lemma *mult-0* [*simp*]: $0 \# * m = 0$
 $\langle \text{proof} \rangle$

lemma *mult-succ* [*simp*]: $\text{succ}(m) \# * n = n \# + (m \# * n)$
 $\langle \text{proof} \rangle$

lemma *mult-0-right* [*simp*]: $m \# * 0 = 0$
 $\langle \text{proof} \rangle$

lemma *mult-succ-right* [*simp*]: $m \# * \text{succ}(n) = m \# + (m \# * n)$
 $\langle \text{proof} \rangle$

lemma *mult-1-natty* [*simp*]: $1 \# * n = \text{natty}(n)$
 $\langle \text{proof} \rangle$

lemma *mult-1-right-natty* [*simp*]: $n \# * 1 = \text{natty}(n)$
 $\langle \text{proof} \rangle$

lemma *mult-1*: $n \in \text{nat} ==> 1 \# * n = n$
 $\langle \text{proof} \rangle$

lemma *mult-1-right*: $n \in \text{nat} ==> n \# * 1 = n$
 $\langle \text{proof} \rangle$

lemma *mult-commute*: $m \# * n = n \# * m$

$\langle proof \rangle$

lemma *add-mult-distrib*: $(m \# + n) \# * k = (m \# * k) \# + (n \# * k)$
 $\langle proof \rangle$

lemma *add-mult-distrib-left*: $k \# * (m \# + n) = (k \# * m) \# + (k \# * n)$
 $\langle proof \rangle$

lemma *mult-assoc*: $(m \# * n) \# * k = m \# * (n \# * k)$
 $\langle proof \rangle$

lemma *mult-left-commute*: $m \# * (n \# * k) = n \# * (m \# * k)$
 $\langle proof \rangle$

lemmas *mult-ac = mult-assoc mult-commute mult-left-commute*

lemma *lt-succ-eq-0-disj*:
[[$m \in nat$; $n \in nat$]]
==> $(m < succ(n)) <-> (m = 0 \mid (\exists j \in nat. m = succ(j) \ \& \ j < n))$
 $\langle proof \rangle$

lemma *less-diff-conv* [rule-format]:
[[$j \in nat$; $k \in nat$]] ==> $\forall i \in nat. (i < j \# - k) <-> (i \# + k < j)$
 $\langle proof \rangle$

lemmas *nat-typechecks = rec-type nat-0I nat-1I nat-succI Ord-nat*

end

27 ArithSimp: Arithmetic with simplification

theory *ArithSimp*
imports *Arith*
uses $\sim \sim /src/Provers/Arith/cancel-numerals.ML$
 $\sim \sim /src/Provers/Arith/combine-numerals.ML$
arith-data.ML
begin

27.1 Difference

lemma *diff-self-eq-0* [simp]: $m \# - m = 0$
 $\langle proof \rangle$

lemma *add-diff-inverse*: $[| n \text{ le } m; m:\text{nat} |] ==> n \# + (m \# - n) = m$
 $\langle \text{proof} \rangle$

lemma *add-diff-inverse2*: $[| n \text{ le } m; m:\text{nat} |] ==> (m \# - n) \# + n = m$
 $\langle \text{proof} \rangle$

lemma *diff-succ*: $[| n \text{ le } m; m:\text{nat} |] ==> \text{succ}(m) \# - n = \text{succ}(m \# - n)$
 $\langle \text{proof} \rangle$

lemma *zero-less-diff* [*simp*]:
 $[| m:\text{nat}; n:\text{nat} |] ==> 0 < (n \# - m) \quad <-> \quad m < n$
 $\langle \text{proof} \rangle$

lemma *diff-mult-distrib*: $(m \# - n) \# * k = (m \# * k) \# - (n \# * k)$
 $\langle \text{proof} \rangle$

lemma *diff-mult-distrib2*: $k \# * (m \# - n) = (k \# * m) \# - (k \# * n)$
 $\langle \text{proof} \rangle$

27.2 Remainder

lemma *div-termination*: $[| 0 < n; n \text{ le } m; m:\text{nat} |] ==> m \# - n < m$
 $\langle \text{proof} \rangle$

lemmas *div-rls* =
nat-typechecks *Ord-transrec-type* *apply-funtype*
div-termination [*THEN ltD*]
nat-into-Ord *not-lt-iff-le* [*THEN iffD1*]

lemma *raw-mod-type*: $[| m:\text{nat}; n:\text{nat} |] ==> \text{raw-mod } (m, n) : \text{nat}$
 $\langle \text{proof} \rangle$

lemma *mod-type* [*TC,iff*]: $m \text{ mod } n : \text{nat}$
 $\langle \text{proof} \rangle$

lemma *DIVISION-BY-ZERO-DIV*: $a \text{ div } 0 = 0$
 $\langle \text{proof} \rangle$

lemma *DIVISION-BY-ZERO-MOD*: $a \text{ mod } 0 = \text{nativify}(a)$
 $\langle \text{proof} \rangle$

lemma *raw-mod-less*: $m < n \implies \text{raw-mod } (m, n) = m$
 $\langle \text{proof} \rangle$

lemma *mod-less* [simp]: $[| m < n; n : \text{nat} |] \implies m \text{ mod } n = m$
 $\langle \text{proof} \rangle$

lemma *raw-mod-geq*:
 $[| 0 < n; n \leq m; m : \text{nat} |] \implies \text{raw-mod } (m, n) = \text{raw-mod } (m \# -n, n)$
 $\langle \text{proof} \rangle$

lemma *mod-geq*: $[| n \leq m; m : \text{nat} |] \implies m \text{ mod } n = (m \# -n) \text{ mod } n$
 $\langle \text{proof} \rangle$

27.3 Division

lemma *raw-div-type*: $[| m : \text{nat}; n : \text{nat} |] \implies \text{raw-div } (m, n) : \text{nat}$
 $\langle \text{proof} \rangle$

lemma *div-type* [TC, iff]: $m \text{ div } n : \text{nat}$
 $\langle \text{proof} \rangle$

lemma *raw-div-less*: $m < n \implies \text{raw-div } (m, n) = 0$
 $\langle \text{proof} \rangle$

lemma *div-less* [simp]: $[| m < n; n : \text{nat} |] \implies m \text{ div } n = 0$
 $\langle \text{proof} \rangle$

lemma *raw-div-geq*: $[| 0 < n; n \leq m; m : \text{nat} |] \implies \text{raw-div}(m, n) = \text{succ}(\text{raw-div}(m \# -n, n))$
 $\langle \text{proof} \rangle$

lemma *div-geq* [simp]:
 $[| 0 < n; n \leq m; m : \text{nat} |] \implies m \text{ div } n = \text{succ } ((m \# -n) \text{ div } n)$
 $\langle \text{proof} \rangle$

declare *div-less* [simp] *div-geq* [simp]

lemma *mod-div-lemma*: $[| m : \text{nat}; n : \text{nat} |] \implies (m \text{ div } n) \# * n \# + m \text{ mod } n = m$
 $\langle \text{proof} \rangle$

lemma *mod-div-equality-nativify*: $(m \text{ div } n) \# * n \# + m \text{ mod } n = \text{nativify}(m)$
 $\langle \text{proof} \rangle$

lemma *mod-div-equality*: $m: \text{nat} \implies (m \text{ div } n) \# * n \# + m \text{ mod } n = m$
 $\langle \text{proof} \rangle$

27.4 Further Facts about Remainder

(mainly for mutilated chess board)

lemma *mod-succ-lemma*:
 $[[\ 0 < n; \ m: \text{nat}; \ n: \text{nat} \]]$
 $\implies \text{succ}(m) \text{ mod } n = (\text{if } \text{succ}(m \text{ mod } n) = n \text{ then } 0 \text{ else } \text{succ}(m \text{ mod } n))$
 $\langle \text{proof} \rangle$

lemma *mod-succ*:
 $n: \text{nat} \implies \text{succ}(m) \text{ mod } n = (\text{if } \text{succ}(m \text{ mod } n) = n \text{ then } 0 \text{ else } \text{succ}(m \text{ mod } n))$
 $\langle \text{proof} \rangle$

lemma *mod-less-divisor*: $[[\ 0 < n; \ n: \text{nat} \]]$ $\implies m \text{ mod } n < n$
 $\langle \text{proof} \rangle$

lemma *mod-1-eq* [simp]: $m \text{ mod } 1 = 0$
 $\langle \text{proof} \rangle$

lemma *mod2-cases*: $b < 2 \implies k \text{ mod } 2 = b \mid k \text{ mod } 2 = (\text{if } b=1 \text{ then } 0 \text{ else } 1)$
 $\langle \text{proof} \rangle$

lemma *mod2-succ-succ* [simp]: $\text{succ}(\text{succ}(m)) \text{ mod } 2 = m \text{ mod } 2$
 $\langle \text{proof} \rangle$

lemma *mod2-add-more* [simp]: $(m \# + m \# + n) \text{ mod } 2 = n \text{ mod } 2$
 $\langle \text{proof} \rangle$

lemma *mod2-add-self* [simp]: $(m \# + m) \text{ mod } 2 = 0$
 $\langle \text{proof} \rangle$

27.5 Additional theorems about \leq

lemma *add-le-self*: $m: \text{nat} \implies m \text{ le } (m \# + n)$
 $\langle \text{proof} \rangle$

lemma *add-le-self2*: $m: \text{nat} \implies m \text{ le } (n \# + m)$
 $\langle \text{proof} \rangle$

lemma *mult-le-mono1*: $[[\ i \text{ le } j; \ j: \text{nat} \]]$ $\implies (i \# * k) \text{ le } (j \# * k)$
 $\langle \text{proof} \rangle$

lemma *mult-le-mono*: $[[\ i \text{ le } j; \ k \text{ le } l; \ j: \text{nat}; \ l: \text{nat} \]]$ $\implies i \# * k \text{ le } j \# * l$

$\langle proof \rangle$

lemma *mult-lt-mono2*: $[| i < j; 0 < k; j : nat; k : nat |] ==> k \#* i < k \#* j$
 $\langle proof \rangle$

lemma *mult-lt-mono1*: $[| i < j; 0 < k; j : nat; k : nat |] ==> i \#* k < j \#* k$
 $\langle proof \rangle$

lemma *add-eq-0-iff* [iff]: $m \# + n = 0 <-> natify(m)=0 \ \& \ natify(n)=0$
 $\langle proof \rangle$

lemma *zero-lt-mult-iff* [iff]: $0 < m \#* n <-> 0 < natify(m) \ \& \ 0 < natify(n)$
 $\langle proof \rangle$

lemma *mult-eq-1-iff* [iff]: $m \#* n = 1 <-> natify(m)=1 \ \& \ natify(n)=1$
 $\langle proof \rangle$

lemma *mult-is-zero*: $[| m : nat; n : nat |] ==> (m \#* n = 0) <-> (m = 0 \mid n = 0)$
 $\langle proof \rangle$

lemma *mult-is-zero-natify* [iff]:
 $(m \#* n = 0) <-> (natify(m) = 0 \mid natify(n) = 0)$
 $\langle proof \rangle$

27.6 Cancellation Laws for Common Factors in Comparisons

lemma *mult-less-cancel-lemma*:
 $[| k : nat; m : nat; n : nat |] ==> (m \#* k < n \#* k) <-> (0 < k \ \& \ m < n)$
 $\langle proof \rangle$

lemma *mult-less-cancel2* [simp]:
 $(m \#* k < n \#* k) <-> (0 < natify(k) \ \& \ natify(m) < natify(n))$
 $\langle proof \rangle$

lemma *mult-less-cancel1* [simp]:
 $(k \#* m < k \#* n) <-> (0 < natify(k) \ \& \ natify(m) < natify(n))$
 $\langle proof \rangle$

lemma *mult-le-cancel2* [simp]: $(m \#* k \leq n \#* k) <-> (0 < natify(k) \ \longrightarrow \ natify(m) \leq natify(n))$
 $\langle proof \rangle$

lemma *mult-le-cancel1* [simp]: $(k \#* m \leq k \#* n) <-> (0 < natify(k) \ \longrightarrow \ natify(m) \leq natify(n))$
 $\langle proof \rangle$

lemma *mult-le-cancel-le1*: $k : \text{nat} \implies k \#* m \text{ le } k \longleftrightarrow (0 < k \longrightarrow \text{nativify}(m) \text{ le } 1)$
 $\langle \text{proof} \rangle$

lemma *Ord-eq-iff-le*: $[| \text{Ord}(m); \text{Ord}(n) |] \implies m=n \longleftrightarrow (m \text{ le } n \ \& \ n \text{ le } m)$
 $\langle \text{proof} \rangle$

lemma *mult-cancel2-lemma*:
 $[| k : \text{nat}; m : \text{nat}; n : \text{nat} |] \implies (m \#* k = n \#* k) \longleftrightarrow (m=n \mid k=0)$
 $\langle \text{proof} \rangle$

lemma *mult-cancel2* [simp]:
 $(m \#* k = n \#* k) \longleftrightarrow (\text{nativify}(m) = \text{nativify}(n) \mid \text{nativify}(k) = 0)$
 $\langle \text{proof} \rangle$

lemma *mult-cancel1* [simp]:
 $(k \#* m = k \#* n) \longleftrightarrow (\text{nativify}(m) = \text{nativify}(n) \mid \text{nativify}(k) = 0)$
 $\langle \text{proof} \rangle$

lemma *div-cancel-raw*:
 $[| 0 < n; 0 < k; k : \text{nat}; m : \text{nat}; n : \text{nat} |] \implies (k \#* m) \text{ div } (k \#* n) = m \text{ div } n$
 $\langle \text{proof} \rangle$

lemma *div-cancel*:
 $[| 0 < \text{nativify}(n); 0 < \text{nativify}(k) |] \implies (k \#* m) \text{ div } (k \#* n) = m \text{ div } n$
 $\langle \text{proof} \rangle$

27.7 More Lemmas about Remainder

lemma *mult-mod-distrib-raw*:
 $[| k : \text{nat}; m : \text{nat}; n : \text{nat} |] \implies (k \#* m) \text{ mod } (k \#* n) = k \#* (m \text{ mod } n)$
 $\langle \text{proof} \rangle$

lemma *mod-mult-distrib2*: $k \#* (m \text{ mod } n) = (k \#* m) \text{ mod } (k \#* n)$
 $\langle \text{proof} \rangle$

lemma *mult-mod-distrib*: $(m \text{ mod } n) \#* k = (m \#* k) \text{ mod } (n \#* k)$
 $\langle \text{proof} \rangle$

lemma *mod-add-self2-raw*: $n \in \text{nat} \implies (m \#+ n) \text{ mod } n = m \text{ mod } n$
 $\langle \text{proof} \rangle$

lemma *mod-add-self2* [simp]: $(m \#+ n) \text{ mod } n = m \text{ mod } n$
 $\langle \text{proof} \rangle$

lemma *mod-add-self1* [simp]: $(n \#+ m) \text{ mod } n = m \text{ mod } n$

$\langle proof \rangle$

lemma *mod-mult-self1-raw*: $k \in nat \implies (m \# + k \# * n) \bmod n = m \bmod n$
 $\langle proof \rangle$

lemma *mod-mult-self1* [simp]: $(m \# + k \# * n) \bmod n = m \bmod n$
 $\langle proof \rangle$

lemma *mod-mult-self2* [simp]: $(m \# + n \# * k) \bmod n = m \bmod n$
 $\langle proof \rangle$

lemma *mult-eq-self-implies-10*: $m = m \# * n \implies \text{natify}(n) = 1 \mid m = 0$
 $\langle proof \rangle$

lemma *less-imp-succ-add* [rule-format]:
 $\llbracket m < n; n: nat \rrbracket \implies \exists k: nat. n = \text{succ}(m \# + k)$
 $\langle proof \rangle$

lemma *less-iff-succ-add*:
 $\llbracket m: nat; n: nat \rrbracket \implies (m < n) \iff (\exists k: nat. n = \text{succ}(m \# + k))$
 $\langle proof \rangle$

lemma *add-lt-elim2*:
 $\llbracket a \# + d = b \# + c; a < b; b \in nat; c \in nat; d \in nat \rrbracket \implies c < d$
 $\langle proof \rangle$

lemma *add-le-elim2*:
 $\llbracket a \# + d = b \# + c; a \leq b; b \in nat; c \in nat; d \in nat \rrbracket \implies c \leq d$
 $\langle proof \rangle$

27.7.1 More Lemmas About Difference

lemma *diff-is-0-lemma*:
 $\llbracket m: nat; n: nat \rrbracket \implies m \# - n = 0 \iff m \leq n$
 $\langle proof \rangle$

lemma *diff-is-0-iff*: $m \# - n = 0 \iff \text{natify}(m) \leq \text{natify}(n)$
 $\langle proof \rangle$

lemma *nat-lt-imp-diff-eq-0*:
 $\llbracket a: nat; b: nat; a < b \rrbracket \implies a \# - b = 0$
 $\langle proof \rangle$

lemma *raw-nat-diff-split*:
 $\llbracket a: nat; b: nat \rrbracket \implies$
 $(P(a \# - b)) \iff ((a < b \implies P(0)) \ \& \ (\forall d: nat. a = b \# + d \implies P(d)))$
 $\langle proof \rangle$

lemma *nat-diff-split*:
 $(P(a \#- b)) <->$
 $(\text{natify}(a) < \text{natify}(b) \dashrightarrow P(0)) \ \& \ (ALL \ d:\text{nat}. \text{natify}(a) = b \#+ d \dashrightarrow P(d))$
 $\langle \text{proof} \rangle$

Difference and less-than

lemma *diff-lt-imp-lt*: $[(k \#- i) < (k \#- j); i \in \text{nat}; j \in \text{nat}; k \in \text{nat}] ==> j < i$
 $\langle \text{proof} \rangle$

lemma *lt-imp-diff-lt*: $[j < i; i \leq k; k \in \text{nat}] ==> (k \#- i) < (k \#- j)$
 $\langle \text{proof} \rangle$

lemma *diff-lt-iff-lt*: $[i \leq k; j \in \text{nat}; k \in \text{nat}] ==> (k \#- i) < (k \#- j) <-> j < i$
 $\langle \text{proof} \rangle$

end

28 List-ZF: Lists in Zermelo-Fraenkel Set Theory

theory *List-ZF* **imports** *Datatype-ZF ArithSimp* **begin**

consts

list :: $i ==> i$

datatype

list(A) = *Nil* | *Cons* ($a:A, l: \text{list}(A)$)

syntax

-*Nil* :: $i \ (\ [])$

-*List* :: $is ==> i \ ([(-)])$

translations

$[x, xs]$ == *CONST Cons*($x, [xs]$)

$[x]$ == *CONST Cons*($x, []$)

$[]$ == *CONST Nil*

consts

length :: $i ==> i$

hd :: $i ==> i$

tl :: $i ==> i$

primrec

length($[]$) = 0

$length(Cons(a,l)) = succ(length(l))$

primrec

$hd([]) = 0$
 $hd(Cons(a,l)) = a$

primrec

$tl([]) = []$
 $tl(Cons(a,l)) = l$

consts

$map \quad \quad \quad :: [i=>i, i] => i$
 $set-of-list \quad :: i=>i$
 $app \quad \quad \quad :: [i,i] => i \quad \quad \quad (\text{infixr } @ \ 60)$

primrec

$map(f,[]) = []$
 $map(f,Cons(a,l)) = Cons(f(a), map(f,l))$

primrec

$set-of-list([]) = 0$
 $set-of-list(Cons(a,l)) = cons(a, set-of-list(l))$

primrec

$app-Nil: [] @ ys = ys$
 $app-Cons: (Cons(a,l)) @ ys = Cons(a, l @ ys)$

consts

$rev \quad :: i=>i$
 $flat \quad \quad :: i=>i$
 $list-add \quad :: i=>i$

primrec

$rev([]) = []$
 $rev(Cons(a,l)) = rev(l) @ [a]$

primrec

$flat([]) = []$
 $flat(Cons(l,ls)) = l @ flat(ls)$

primrec

$list-add([]) = 0$
 $list-add(Cons(a,l)) = a \#+ list-add(l)$

consts

$drop \quad \quad :: [i,i] => i$

primrec

drop-0: $\text{drop}(0, l) = l$
drop-succ: $\text{drop}(\text{succ}(i), l) = \text{tl } (\text{drop}(i, l))$

definition

take :: $[i, i] \Rightarrow i$ **where**
take(*n*, *as*) == *list-rec*(*lam n:nat. []*,
 $\%a \ l \ r. \text{lam } n:\text{nat}. \text{nat-case}([], \%m. \text{Cons}(a, r' m), n), \text{as}) 'n$

definition

nth :: $[i, i] \Rightarrow i$ **where**
— returns the (n+1)th element of a list, or 0 if the list is too short.
nth(*n*, *as*) == *list-rec*(*lam n:nat. 0*,
 $\%a \ l \ r. \text{lam } n:\text{nat}. \text{nat-case}(a, \%m. r' m, n), \text{as}) 'n$

definition

list-update :: $[i, i, i] \Rightarrow i$ **where**
list-update(*xs*, *i*, *v*) == *list-rec*(*lam n:nat. Nil*,
 $\%u \ us \ vs. \text{lam } n:\text{nat}. \text{nat-case}(\text{Cons}(v, us), \%m. \text{Cons}(u, vs' m), n), xs) 'i$

consts

filter :: $[i \Rightarrow o, i] \Rightarrow i$
upt :: $[i, i] \Rightarrow i$

primrec

filter(*P*, *Nil*) = *Nil*
filter(*P*, *Cons*(*x*, *xs*)) =
 $(\text{if } P(x) \text{ then } \text{Cons}(x, \text{filter}(P, xs)) \text{ else } \text{filter}(P, xs))$

primrec

upt(*i*, 0) = *Nil*
upt(*i*, *succ*(*j*)) = $(\text{if } i \text{ le } j \text{ then } \text{upt}(i, j)@[j] \text{ else } \text{Nil})$

definition

min :: $[i, i] \Rightarrow i$ **where**
min(*x*, *y*) == $(\text{if } x \text{ le } y \text{ then } x \text{ else } y)$

definition

max :: $[i, i] \Rightarrow i$ **where**
max(*x*, *y*) == $(\text{if } x \text{ le } y \text{ then } y \text{ else } x)$

declare *list.intros* [*simp*, *TC*]

inductive-cases *ConsE*: $\text{Cons}(a,l) : \text{list}(A)$

lemma *Cons-type-iff* [*simp*]: $\text{Cons}(a,l) \in \text{list}(A) <-> a \in A \ \& \ l \in \text{list}(A)$
<proof>

lemma *Cons-iff*: $\text{Cons}(a,l) = \text{Cons}(a',l') <-> a = a' \ \& \ l = l'$
<proof>

lemma *Nil-Cons-iff*: $\sim \text{Nil} = \text{Cons}(a,l)$
<proof>

lemma *list-unfold*: $\text{list}(A) = \{0\} + (A * \text{list}(A))$
<proof>

lemma *list-mono*: $A \leq B ==> \text{list}(A) \leq \text{list}(B)$
<proof>

lemma *list-univ*: $\text{list}(\text{univ}(A)) \leq \text{univ}(A)$
<proof>

lemmas *list-subset-univ* = *subset-trans* [*OF list-mono list-univ*]

lemma *list-into-univ*: $[\mid l : \text{list}(A); \ A \leq \text{univ}(B) \mid] ==> l : \text{univ}(B)$
<proof>

lemma *list-case-type*:
 $[\mid l : \text{list}(A);$
 $c : C(\text{Nil});$
 $!!x \ y. [\mid x : A; \ y : \text{list}(A) \mid] ==> h(x,y) : C(\text{Cons}(x,y))$
 $\mid] ==> \text{list-case}(c,h,l) : C(l)$
<proof>

lemma *list-0-triv*: $\text{list}(0) = \{\text{Nil}\}$
<proof>

lemma *tl-type*: $l : \text{list}(A) ==> \text{tl}(l) : \text{list}(A)$
<proof>

lemma *drop-Nil* [*simp*]: $i:\text{nat} \implies \text{drop}(i, \text{Nil}) = \text{Nil}$
 $\langle \text{proof} \rangle$

lemma *drop-succ-Cons* [*simp*]: $i:\text{nat} \implies \text{drop}(\text{succ}(i), \text{Cons}(a,l)) = \text{drop}(i,l)$
 $\langle \text{proof} \rangle$

lemma *drop-type* [*simp*, *TC*]: $[i:\text{nat}; l:\text{list}(A)] \implies \text{drop}(i,l) : \text{list}(A)$
 $\langle \text{proof} \rangle$

declare *drop-succ* [*simp del*]

lemma *list-rec-type* [*TC*]:
 $[l:\text{list}(A);$
 $c:C(\text{Nil});$
 $!!x\ y\ r. [x:A; y:\text{list}(A); r:C(y)] \implies h(x,y,r):C(\text{Cons}(x,y))$
 $] \implies \text{list-rec}(c,h,l) : C(l)$
 $\langle \text{proof} \rangle$

lemma *map-type* [*TC*]:
 $[l:\text{list}(A); !!x. x:A \implies h(x):B] \implies \text{map}(h,l) : \text{list}(B)$
 $\langle \text{proof} \rangle$

lemma *map-type2* [*TC*]: $l:\text{list}(A) \implies \text{map}(h,l) : \text{list}(\{h(u). u:A\})$
 $\langle \text{proof} \rangle$

lemma *length-type* [*TC*]: $l:\text{list}(A) \implies \text{length}(l) : \text{nat}$
 $\langle \text{proof} \rangle$

lemma *lt-length-in-nat*:
 $[x < \text{length}(xs); xs \in \text{list}(A)] \implies x \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *app-type* [*TC*]: $[xs:\text{list}(A); ys:\text{list}(A)] \implies xs@ys : \text{list}(A)$
 $\langle \text{proof} \rangle$

lemma *rev-type* [*TC*]: $xs:\text{list}(A) \implies \text{rev}(xs) : \text{list}(A)$

$\langle proof \rangle$

lemma *flat-type* [TC]: $ls: list(list(A)) \implies flat(ls) : list(A)$
 $\langle proof \rangle$

lemma *set-of-list-type* [TC]: $l: list(A) \implies set-of-list(l) : Pow(A)$
 $\langle proof \rangle$

lemma *set-of-list-append*:
 $xs: list(A) \implies set-of-list (xs@ys) = set-of-list(xs) \cup set-of-list(ys)$
 $\langle proof \rangle$

lemma *list-add-type* [TC]: $xs: list(nat) \implies list-add(xs) : nat$
 $\langle proof \rangle$

lemma *map-ident* [simp]: $l: list(A) \implies map(\%u. u, l) = l$
 $\langle proof \rangle$

lemma *map-compose*: $l: list(A) \implies map(h, map(j,l)) = map(\%u. h(j(u)), l)$
 $\langle proof \rangle$

lemma *map-app-distrib*: $xs: list(A) \implies map(h, xs@ys) = map(h,xs) @ map(h,ys)$
 $\langle proof \rangle$

lemma *map-flat*: $ls: list(list(A)) \implies map(h, flat(ls)) = flat(map(map(h),ls))$
 $\langle proof \rangle$

lemma *list-rec-map*:
 $l: list(A) \implies$
 $list-rec(c, d, map(h,l)) =$
 $list-rec(c, \%x xs r. d(h(x), map(h,xs), r), l)$
 $\langle proof \rangle$

lemmas *list-CollectD* = *Collect-subset* [THEN *list-mono*, THEN *subsetD*, stan-

dard]

lemma *map-list-Collect*: $l: \text{list}(\{x:A. h(x)=j(x)\}) \implies \text{map}(h,l) = \text{map}(j,l)$
 $\langle \text{proof} \rangle$

lemma *length-map* [*simp*]: $xs: \text{list}(A) \implies \text{length}(\text{map}(h,xs)) = \text{length}(xs)$
 $\langle \text{proof} \rangle$

lemma *length-app* [*simp*]:
 $[[\, xs: \text{list}(A); ys: \text{list}(A) \,]]$
 $\implies \text{length}(xs@ys) = \text{length}(xs) \# + \text{length}(ys)$
 $\langle \text{proof} \rangle$

lemma *length-rev* [*simp*]: $xs: \text{list}(A) \implies \text{length}(\text{rev}(xs)) = \text{length}(xs)$
 $\langle \text{proof} \rangle$

lemma *length-flat*:
 $ls: \text{list}(\text{list}(A)) \implies \text{length}(\text{flat}(ls)) = \text{list-add}(\text{map}(\text{length},ls))$
 $\langle \text{proof} \rangle$

lemma *drop-length-Cons* [*rule-format*]:
 $xs: \text{list}(A) \implies$
 $\forall x. \text{EX } z \text{ } zs. \text{drop}(\text{length}(xs), \text{Cons}(x,xs)) = \text{Cons}(z,zs)$
 $\langle \text{proof} \rangle$

lemma *drop-length* [*rule-format*]:
 $l: \text{list}(A) \implies \forall i \in \text{length}(l). (\text{EX } z \text{ } zs. \text{drop}(i,l) = \text{Cons}(z,zs))$
 $\langle \text{proof} \rangle$

lemma *app-right-Nil* [*simp*]: $xs: \text{list}(A) \implies xs@Nil=xs$
 $\langle \text{proof} \rangle$

lemma *app-assoc*: $xs: \text{list}(A) \implies (xs@ys)@zs = xs@(ys@zs)$
 $\langle \text{proof} \rangle$

lemma *flat-app-distrib*: $ls: \text{list}(\text{list}(A)) \implies \text{flat}(ls@ms) = \text{flat}(ls)@flat(ms)$
 $\langle \text{proof} \rangle$

lemma *rev-map-distrib*: $l: \text{list}(A) \implies \text{rev}(\text{map}(h,l)) = \text{map}(h,\text{rev}(l))$

$\langle proof \rangle$

lemma *rev-app-distrib*:

$\llbracket xs: list(A); ys: list(A) \rrbracket ==> rev(xs@ys) = rev(ys)@rev(xs)$
 $\langle proof \rangle$

lemma *rev-rev-ident* [simp]: $l: list(A) ==> rev(rev(l))=l$

$\langle proof \rangle$

lemma *rev-flat*: $ls: list(list(A)) ==> rev(flat(ls)) = flat(map(rev,rev(ls)))$

$\langle proof \rangle$

lemma *list-add-app*:

$\llbracket xs: list(nat); ys: list(nat) \rrbracket$
 $==> list-add(xs@ys) = list-add(ys) \# + list-add(xs)$
 $\langle proof \rangle$

lemma *list-add-rev*: $l: list(nat) ==> list-add(rev(l)) = list-add(l)$

$\langle proof \rangle$

lemma *list-add-flat*:

$ls: list(list(nat)) ==> list-add(flat(ls)) = list-add(map(list-add,ls))$
 $\langle proof \rangle$

lemma *list-append-induct* [case-names Nil snoc, consumes 1]:

$\llbracket l: list(A);$
 $P(Nil);$
 $!!x y. \llbracket x: A; y: list(A); P(y) \rrbracket ==> P(y @ [x])$
 $\rrbracket ==> P(l)$
 $\langle proof \rangle$

lemma *list-complete-induct-lemma* [rule-format]:

assumes *ih*:
 $\bigwedge l. \llbracket l \in list(A);$
 $\forall l' \in list(A). length(l') < length(l) --> P(l') \rrbracket$
 $==> P(l)$
shows $n \in nat ==> \forall l \in list(A). length(l) < n --> P(l)$
 $\langle proof \rangle$

theorem *list-complete-induct*:

$\llbracket l \in list(A);$
 $\bigwedge l. \llbracket l \in list(A);$
 $\forall l' \in list(A). length(l') < length(l) --> P(l') \rrbracket$

$$\begin{aligned} & \implies P(l) \\ \text{[]} & \implies P(l) \\ \langle \text{proof} \rangle & \end{aligned}$$

lemma *min-sym*: [$i:\text{nat}; j:\text{nat}$] $\implies \text{min}(i,j)=\text{min}(j,i)$
 $\langle \text{proof} \rangle$

lemma *min-type* [simp,TC]: [$i:\text{nat}; j:\text{nat}$] $\implies \text{min}(i,j):\text{nat}$
 $\langle \text{proof} \rangle$

lemma *min-0* [simp]: $i:\text{nat} \implies \text{min}(0,i) = 0$
 $\langle \text{proof} \rangle$

lemma *min-02* [simp]: $i:\text{nat} \implies \text{min}(i, 0) = 0$
 $\langle \text{proof} \rangle$

lemma *lt-min-iff*: [$i:\text{nat}; j:\text{nat}; k:\text{nat}$] $\implies i < \text{min}(j,k) \iff i < j \ \& \ i < k$
 $\langle \text{proof} \rangle$

lemma *min-succ-succ* [simp]:
[$i:\text{nat}; j:\text{nat}$] $\implies \text{min}(\text{succ}(i), \text{succ}(j)) = \text{succ}(\text{min}(i, j))$
 $\langle \text{proof} \rangle$

lemma *filter-append* [simp]:
 $xs:\text{list}(A) \implies \text{filter}(P, xs @ ys) = \text{filter}(P, xs) @ \text{filter}(P, ys)$
 $\langle \text{proof} \rangle$

lemma *filter-type* [simp,TC]: $xs:\text{list}(A) \implies \text{filter}(P, xs):\text{list}(A)$
 $\langle \text{proof} \rangle$

lemma *length-filter*: $xs:\text{list}(A) \implies \text{length}(\text{filter}(P, xs)) \leq \text{length}(xs)$
 $\langle \text{proof} \rangle$

lemma *filter-is-subset*: $xs:\text{list}(A) \implies \text{set-of-list}(\text{filter}(P, xs)) \leq \text{set-of-list}(xs)$
 $\langle \text{proof} \rangle$

lemma *filter-False* [simp]: $xs:\text{list}(A) \implies \text{filter}(\%p. \text{False}, xs) = \text{Nil}$
 $\langle \text{proof} \rangle$

lemma *filter-True* [simp]: $xs:\text{list}(A) \implies \text{filter}(\%p. \text{True}, xs) = xs$

$\langle proof \rangle$

lemma *length-is-0-iff* [simp]: $xs:list(A) ==> length(xs)=0 <-> xs=Nil$
 $\langle proof \rangle$

lemma *length-is-0-iff2* [simp]: $xs:list(A) ==> 0 = length(xs) <-> xs=Nil$
 $\langle proof \rangle$

lemma *length-tl* [simp]: $xs:list(A) ==> length(tl(xs)) = length(xs) \#- 1$
 $\langle proof \rangle$

lemma *length-greater-0-iff*: $xs:list(A) ==> 0 < length(xs) <-> xs \sim Nil$
 $\langle proof \rangle$

lemma *length-succ-iff*: $xs:list(A) ==> length(xs)=succ(n) <-> (EX y ys. xs=Cons(y, ys) \& length(ys)=n)$
 $\langle proof \rangle$

lemma *append-is-Nil-iff* [simp]:
 $xs:list(A) ==> (xs@ys = Nil) <-> (xs=Nil \& ys = Nil)$
 $\langle proof \rangle$

lemma *append-is-Nil-iff2* [simp]:
 $xs:list(A) ==> (Nil = xs@ys) <-> (xs=Nil \& ys = Nil)$
 $\langle proof \rangle$

lemma *append-left-is-self-iff* [simp]:
 $xs:list(A) ==> (xs@ys = xs) <-> (ys = Nil)$
 $\langle proof \rangle$

lemma *append-left-is-self-iff2* [simp]:
 $xs:list(A) ==> (xs = xs@ys) <-> (ys = Nil)$
 $\langle proof \rangle$

lemma *append-left-is-Nil-iff* [rule-format]:
 $[| xs:list(A); ys:list(A); zs:list(A) |] ==>$
 $length(ys)=length(zs) \dashrightarrow (xs@ys=zs <-> (xs=Nil \& ys=zs))$
 $\langle proof \rangle$

lemma *append-left-is-Nil-iff2* [rule-format]:
 $[| xs:list(A); ys:list(A); zs:list(A) |] ==>$
 $length(ys)=length(zs) \dashrightarrow (zs=ys@xs <-> (xs=Nil \& ys=zs))$
 $\langle proof \rangle$

lemma *append-eq-append-iff* [rule-format,simp]:
 $xs: \text{list}(A) \implies \forall ys \in \text{list}(A).$
 $\text{length}(xs) = \text{length}(ys) \dashv\vdash (xs @ us = ys @ us) \iff (xs = ys \ \& \ us = vs)$
 <proof>

lemma *append-eq-append* [rule-format]:
 $xs: \text{list}(A) \implies$
 $\forall ys \in \text{list}(A). \forall us \in \text{list}(A). \forall vs \in \text{list}(A).$
 $\text{length}(us) = \text{length}(vs) \dashv\vdash (xs @ us = ys @ vs) \dashv\vdash (xs = ys \ \& \ us = vs)$
 <proof>

lemma *append-eq-append-iff2* [simp]:
 $[\mid xs: \text{list}(A); ys: \text{list}(A); us: \text{list}(A); vs: \text{list}(A); \text{length}(us) = \text{length}(vs) \mid]$
 $\implies xs @ us = ys @ vs \iff (xs = ys \ \& \ us = vs)$
 <proof>

lemma *append-self-iff* [simp]:
 $[\mid xs: \text{list}(A); ys: \text{list}(A); zs: \text{list}(A) \mid] \implies xs @ ys = xs @ zs \iff ys = zs$
 <proof>

lemma *append-self-iff2* [simp]:
 $[\mid xs: \text{list}(A); ys: \text{list}(A); zs: \text{list}(A) \mid] \implies ys @ xs = zs @ xs \iff ys = zs$
 <proof>

lemma *append1-eq-iff* [rule-format,simp]:
 $xs: \text{list}(A) \implies \forall ys \in \text{list}(A). xs @ [x] = ys @ [y] \iff (xs = ys \ \& \ x = y)$
 <proof>

lemma *append-right-is-self-iff* [simp]:
 $[\mid xs: \text{list}(A); ys: \text{list}(A) \mid] \implies (xs @ ys = ys) \iff (xs = \text{Nil})$
 <proof>

lemma *append-right-is-self-iff2* [simp]:
 $[\mid xs: \text{list}(A); ys: \text{list}(A) \mid] \implies (ys = xs @ ys) \iff (xs = \text{Nil})$
 <proof>

lemma *hd-append* [rule-format,simp]:
 $xs: \text{list}(A) \implies xs \sim \text{Nil} \dashv\vdash \text{hd}(xs @ ys) = \text{hd}(xs)$
 <proof>

lemma *tl-append* [rule-format,simp]:
 $xs: \text{list}(A) \implies xs \sim \text{Nil} \dashv\vdash \text{tl}(xs @ ys) = \text{tl}(xs) @ ys$
 <proof>

lemma *rev-is-Nil-iff* [simp]: $xs: \text{list}(A) \implies (\text{rev}(xs) = \text{Nil} \iff xs = \text{Nil})$

$\langle proof \rangle$

lemma *Nil-is-rev-iff* [simp]: $xs: list(A) \implies (Nil = rev(xs) \iff xs = Nil)$
 $\langle proof \rangle$

lemma *rev-is-rev-iff* [rule-format,simp]:
 $xs: list(A) \implies \forall ys \in list(A). rev(xs) = rev(ys) \iff xs = ys$
 $\langle proof \rangle$

lemma *rev-list-elim* [rule-format]:
 $xs: list(A) \implies$
 $(xs = Nil \implies P) \implies (\forall ys \in list(A). \forall y \in A. xs = ys @ [y] \implies P) \implies P$
 $\langle proof \rangle$

lemma *length-drop* [rule-format,simp]:
 $n: nat \implies \forall xs \in list(A). length(drop(n, xs)) = length(xs) \# - n$
 $\langle proof \rangle$

lemma *drop-all* [rule-format,simp]:
 $n: nat \implies \forall xs \in list(A). length(xs) \leq n \implies drop(n, xs) = Nil$
 $\langle proof \rangle$

lemma *drop-append* [rule-format]:
 $n: nat \implies$
 $\forall xs \in list(A). drop(n, xs @ ys) = drop(n, xs) @ drop(n \# - length(xs), ys)$
 $\langle proof \rangle$

lemma *drop-drop*:
 $m: nat \implies \forall xs \in list(A). \forall n \in nat. drop(n, drop(m, xs)) = drop(n \# + m, xs)$
 $\langle proof \rangle$

lemma *take-0* [simp]: $xs: list(A) \implies take(0, xs) = Nil$
 $\langle proof \rangle$

lemma *take-succ-Cons* [simp]:
 $n: nat \implies take(succ(n), Cons(a, xs)) = Cons(a, take(n, xs))$
 $\langle proof \rangle$

lemma *take-Nil* [simp]: $n: nat \implies take(n, Nil) = Nil$
 $\langle proof \rangle$

lemma *take-all* [rule-format,simp]:

$n: \text{nat} \implies \forall xs \in \text{list}(A). \text{length}(xs) \leq n \implies \text{take}(n, xs) = xs$
 $\langle \text{proof} \rangle$

lemma *take-type* [rule-format,simp,TC]:
 $xs: \text{list}(A) \implies \forall n \in \text{nat}. \text{take}(n, xs): \text{list}(A)$
 $\langle \text{proof} \rangle$

lemma *take-append* [rule-format,simp]:
 $xs: \text{list}(A) \implies$
 $\forall ys \in \text{list}(A). \forall n \in \text{nat}. \text{take}(n, xs @ ys) =$
 $\text{take}(n, xs) @ \text{take}(n \# - \text{length}(xs), ys)$
 $\langle \text{proof} \rangle$

lemma *take-take* [rule-format]:
 $m : \text{nat} \implies$
 $\forall xs \in \text{list}(A). \forall n \in \text{nat}. \text{take}(n, \text{take}(m, xs)) = \text{take}(\min(n, m), xs)$
 $\langle \text{proof} \rangle$

lemma *nth-0* [simp]: $\text{nth}(0, \text{Cons}(a, l)) = a$
 $\langle \text{proof} \rangle$

lemma *nth-Cons* [simp]: $n: \text{nat} \implies \text{nth}(\text{succ}(n), \text{Cons}(a, l)) = \text{nth}(n, l)$
 $\langle \text{proof} \rangle$

lemma *nth-empty* [simp]: $\text{nth}(n, \text{Nil}) = 0$
 $\langle \text{proof} \rangle$

lemma *nth-type* [rule-format,simp,TC]:
 $xs: \text{list}(A) \implies \forall n. n < \text{length}(xs) \implies \text{nth}(n, xs) : A$
 $\langle \text{proof} \rangle$

lemma *nth-eq-0* [rule-format]:
 $xs: \text{list}(A) \implies \forall n \in \text{nat}. \text{length}(xs) \leq n \implies \text{nth}(n, xs) = 0$
 $\langle \text{proof} \rangle$

lemma *nth-append* [rule-format]:
 $xs: \text{list}(A) \implies$
 $\forall n \in \text{nat}. \text{nth}(n, xs @ ys) = (\text{if } n < \text{length}(xs) \text{ then } \text{nth}(n, xs)$
 $\text{else } \text{nth}(n \# - \text{length}(xs), ys))$
 $\langle \text{proof} \rangle$

lemma *set-of-list-conv-nth*:
 $xs: \text{list}(A)$
 $\implies \text{set-of-list}(xs) = \{x: A. \exists i: \text{nat}. i < \text{length}(xs) \ \& \ x = \text{nth}(i, xs)\}$
 $\langle \text{proof} \rangle$

lemma *nth-take-lemma* [rule-format]:

$k: \text{nat} ==>$
 $\forall xs \in \text{list}(A). (\forall ys \in \text{list}(A). k \text{ le } \text{length}(xs) \text{ --> } k \text{ le } \text{length}(ys) \text{ -->}$
 $(\forall i \in \text{nat}. i < k \text{ --> } \text{nth}(i, xs) = \text{nth}(i, ys)) \text{ --> } \text{take}(k, xs) = \text{take}(k, ys))$
 $\langle \text{proof} \rangle$

lemma *nth-equalityI* [rule-format]:

$[[xs: \text{list}(A); ys: \text{list}(A); \text{length}(xs) = \text{length}(ys);$
 $\forall i \in \text{nat}. i < \text{length}(xs) \text{ --> } \text{nth}(i, xs) = \text{nth}(i, ys)]]$
 $==> xs = ys$
 $\langle \text{proof} \rangle$

lemma *take-equalityI* [rule-format]:

$[[xs: \text{list}(A); ys: \text{list}(A); (\forall i \in \text{nat}. \text{take}(i, xs) = \text{take}(i, ys))]]$
 $==> xs = ys$
 $\langle \text{proof} \rangle$

lemma *nth-drop* [rule-format]:

$n: \text{nat} ==> \forall i \in \text{nat}. \forall xs \in \text{list}(A). \text{nth}(i, \text{drop}(n, xs)) = \text{nth}(n \# + i, xs)$
 $\langle \text{proof} \rangle$

lemma *take-succ* [rule-format]:

$xs \in \text{list}(A)$
 $==> \forall i. i < \text{length}(xs) \text{ --> } \text{take}(\text{succ}(i), xs) = \text{take}(i, xs) @ [\text{nth}(i, xs)]$
 $\langle \text{proof} \rangle$

lemma *take-add* [rule-format]:

$[[xs \in \text{list}(A); j \in \text{nat}]]$
 $==> \forall i \in \text{nat}. \text{take}(i \# + j, xs) = \text{take}(i, xs) @ \text{take}(j, \text{drop}(i, xs))$
 $\langle \text{proof} \rangle$

lemma *length-take*:

$l \in \text{list}(A) ==> \forall n \in \text{nat}. \text{length}(\text{take}(n, l)) = \min(n, \text{length}(l))$
 $\langle \text{proof} \rangle$

28.1 The function zip

Crafty definition to eliminate a type argument

consts

zip-aux :: $[i, i] ==> i$

primrec

$\text{zip-aux}(B, []) =$
 $(\lambda ys \in \text{list}(B). \text{list-case}([], \%y l. [], ys))$

$\text{zip-aux}(B, \text{Cons}(x, l)) =$

$(\lambda ys \in \text{list}(B).$
 $\text{list-case}(\text{Nil}, \%y \text{ } zs. \text{Cons}(<x,y>, \text{zip-aux}(B,l) 'zs), ys))$

definition

$\text{zip} :: [i, i] \Rightarrow i$ **where**
 $\text{zip}(xs, ys) == \text{zip-aux}(\text{set-of-list}(ys), xs) 'ys$

lemma *list-on-set-of-list*: $xs \in \text{list}(A) \Rightarrow xs \in \text{list}(\text{set-of-list}(xs))$
 $\langle \text{proof} \rangle$

lemma *zip-Nil* [*simp*]: $ys:\text{list}(A) \Rightarrow \text{zip}(\text{Nil}, ys) = \text{Nil}$
 $\langle \text{proof} \rangle$

lemma *zip-Nil2* [*simp*]: $xs:\text{list}(A) \Rightarrow \text{zip}(xs, \text{Nil}) = \text{Nil}$
 $\langle \text{proof} \rangle$

lemma *zip-aux-unique* [*rule-format*]:
 $[| B \leq C; xs \in \text{list}(A) |] \Rightarrow \forall ys \in \text{list}(B). \text{zip-aux}(C, xs) 'ys = \text{zip-aux}(B, xs) 'ys$
 $\langle \text{proof} \rangle$

lemma *zip-Cons-Cons* [*simp*]:
 $[| xs:\text{list}(A); ys:\text{list}(B); x:A; y:B |] \Rightarrow$
 $\text{zip}(\text{Cons}(x, xs), \text{Cons}(y, ys)) = \text{Cons}(<x,y>, \text{zip}(xs, ys))$
 $\langle \text{proof} \rangle$

lemma *zip-type* [*rule-format, simp, TC*]:
 $xs:\text{list}(A) \Rightarrow \forall ys \in \text{list}(B). \text{zip}(xs, ys):\text{list}(A*B)$
 $\langle \text{proof} \rangle$

lemma *length-zip* [*rule-format, simp*]:
 $xs:\text{list}(A) \Rightarrow \forall ys \in \text{list}(B). \text{length}(\text{zip}(xs, ys)) =$
 $\text{min}(\text{length}(xs), \text{length}(ys))$
 $\langle \text{proof} \rangle$

lemma *zip-append1* [*rule-format*]:
 $[| ys:\text{list}(A); zs:\text{list}(B) |] \Rightarrow$
 $\forall xs \in \text{list}(A). \text{zip}(xs @ ys, zs) =$
 $\text{zip}(xs, \text{take}(\text{length}(xs), zs)) @ \text{zip}(ys, \text{drop}(\text{length}(xs), zs))$
 $\langle \text{proof} \rangle$

lemma *zip-append2* [*rule-format*]:
 $[| xs:\text{list}(A); zs:\text{list}(B) |] \Rightarrow \forall ys \in \text{list}(B). \text{zip}(xs, ys @ zs) =$
 $\text{zip}(\text{take}(\text{length}(ys), xs), ys) @ \text{zip}(\text{drop}(\text{length}(ys), xs), zs)$
 $\langle \text{proof} \rangle$

lemma *zip-append* [*simp*]:

$$[[\text{length}(xs) = \text{length}(us); \text{length}(ys) = \text{length}(vs);$$

$$xs:\text{list}(A); us:\text{list}(B); ys:\text{list}(A); vs:\text{list}(B)]]$$

$$\implies \text{zip}(xs @ ys, us @ vs) = \text{zip}(xs, us) @ \text{zip}(ys, vs)$$

<proof>

lemma *zip-rev* [*rule-format, simp*]:

$$ys:\text{list}(B) \implies \forall xs \in \text{list}(A).$$

$$\text{length}(xs) = \text{length}(ys) \dashrightarrow \text{zip}(\text{rev}(xs), \text{rev}(ys)) = \text{rev}(\text{zip}(xs, ys))$$

<proof>

lemma *nth-zip* [*rule-format, simp*]:

$$ys:\text{list}(B) \implies \forall i \in \text{nat}. \forall xs \in \text{list}(A).$$

$$i < \text{length}(xs) \dashrightarrow i < \text{length}(ys) \dashrightarrow$$

$$\text{nth}(i, \text{zip}(xs, ys)) = \langle \text{nth}(i, xs), \text{nth}(i, ys) \rangle$$

<proof>

lemma *set-of-list-zip* [*rule-format*]:

$$[[xs:\text{list}(A); ys:\text{list}(B); i:\text{nat}]]$$

$$\implies \text{set-of-list}(\text{zip}(xs, ys)) =$$

$$\{ \langle x, y \rangle : A * B. \text{EX } i:\text{nat}. i < \min(\text{length}(xs), \text{length}(ys))$$

$$\& x = \text{nth}(i, xs) \& y = \text{nth}(i, ys) \}$$

<proof>

lemma *list-update-Nil* [*simp*]: $i:\text{nat} \implies \text{list-update}(\text{Nil}, i, v) = \text{Nil}$
<proof>

lemma *list-update-Cons-0* [*simp*]: $\text{list-update}(\text{Cons}(x, xs), 0, v) = \text{Cons}(v, xs)$
<proof>

lemma *list-update-Cons-succ* [*simp*]:

$$n:\text{nat} \implies$$

$$\text{list-update}(\text{Cons}(x, xs), \text{succ}(n), v) = \text{Cons}(x, \text{list-update}(xs, n, v))$$

<proof>

lemma *list-update-type* [*rule-format, simp, TC*]:

$$[[xs:\text{list}(A); v:A]] \implies \forall n \in \text{nat}. \text{list-update}(xs, n, v) : \text{list}(A)$$

<proof>

lemma *length-list-update* [*rule-format, simp*]:

$$xs:\text{list}(A) \implies \forall i \in \text{nat}. \text{length}(\text{list-update}(xs, i, v)) = \text{length}(xs)$$

<proof>

lemma *nth-list-update* [*rule-format*]:

$$[[xs:\text{list}(A)]] \implies \forall i \in \text{nat}. \forall j \in \text{nat}. i < \text{length}(xs) \dashrightarrow$$

$nth(j, list-update(xs, i, x)) = (if\ i=j\ then\ x\ else\ nth(j, xs))$
 $\langle proof \rangle$

lemma *nth-list-update-eq* [simp]:

$[[\ i < length(xs); xs:list(A) \]] ==> nth(i, list-update(xs, i, x)) = x$
 $\langle proof \rangle$

lemma *nth-list-update-neq* [rule-format,simp]:

$xs:list(A) ==>$
 $\forall i \in nat. \forall j \in nat. i \sim j \dashv\rightarrow nth(j, list-update(xs, i, x)) = nth(j, xs)$
 $\langle proof \rangle$

lemma *list-update-overwrite* [rule-format,simp]:

$xs:list(A) ==> \forall i \in nat. i < length(xs)$
 $\dashv\rightarrow list-update(list-update(xs, i, x), i, y) = list-update(xs, i, y)$
 $\langle proof \rangle$

lemma *list-update-same-conv* [rule-format]:

$xs:list(A) ==>$
 $\forall i \in nat. i < length(xs) \dashv\rightarrow$
 $(list-update(xs, i, x) = xs) <-> (nth(i, xs) = x)$
 $\langle proof \rangle$

lemma *update-zip* [rule-format]:

$ys:list(B) ==>$
 $\forall i \in nat. \forall xy \in A*B. \forall xs \in list(A).$
 $length(xs) = length(ys) \dashv\rightarrow$
 $list-update(zip(xs, ys), i, xy) = zip(list-update(xs, i, fst(xy)),$
 $list-update(ys, i, snd(xy)))$
 $\langle proof \rangle$

lemma *set-update-subset-cons* [rule-format]:

$xs:list(A) ==>$
 $\forall i \in nat. set-of-list(list-update(xs, i, x)) <= cons(x, set-of-list(xs))$
 $\langle proof \rangle$

lemma *set-of-list-update-subsetI*:

$[[\ set-of-list(xs) <= A; xs:list(A); x:A; i:nat \]]$
 $==> set-of-list(list-update(xs, i, x)) <= A$
 $\langle proof \rangle$

lemma *upt-rec*:

$j:nat ==> upt(i, j) = (if\ i<j\ then\ Cons(i, upt(succ(i), j))\ else\ Nil)$
 $\langle proof \rangle$

lemma *upt-conv-Nil* [simp]: $[[\ j\ le\ i; j:nat \]] ==> upt(i, j) = Nil$

$\langle proof \rangle$

lemma *upt-succ-append*:

$\llbracket i \text{ le } j; j:\text{nat} \rrbracket \implies \text{upt}(i, \text{succ}(j)) = \text{upt}(i, j) @ [j]$
 $\langle proof \rangle$

lemma *upt-conv-Cons*:

$\llbracket i < j; j:\text{nat} \rrbracket \implies \text{upt}(i, j) = \text{Cons}(i, \text{upt}(\text{succ}(i), j))$
 $\langle proof \rangle$

lemma *upt-type* $[simp, TC]: j:\text{nat} \implies \text{upt}(i, j):list(\text{nat})$

$\langle proof \rangle$

lemma *upt-add-eq-append*:

$\llbracket i \text{ le } j; j:\text{nat}; k:\text{nat} \rrbracket \implies \text{upt}(i, j \# + k) = \text{upt}(i, j) @ \text{upt}(j, j \# + k)$
 $\langle proof \rangle$

lemma *length-upt* $[simp]: \llbracket i:\text{nat}; j:\text{nat} \rrbracket \implies \text{length}(\text{upt}(i, j)) = j \# - i$

$\langle proof \rangle$

lemma *nth-upt* $[rule-format, simp]:$

$\llbracket i:\text{nat}; j:\text{nat}; k:\text{nat} \rrbracket \implies i \# + k < j \dashv\dashv \text{nth}(k, \text{upt}(i, j)) = i \# + k$
 $\langle proof \rangle$

lemma *take-upt* $[rule-format, simp]:$

$\llbracket m:\text{nat}; n:\text{nat} \rrbracket \implies$
 $\forall i \in \text{nat}. i \# + m \text{ le } n \dashv\dashv \text{take}(m, \text{upt}(i, n)) = \text{upt}(i, i \# + m)$
 $\langle proof \rangle$

lemma *map-succ-upt*:

$\llbracket m:\text{nat}; n:\text{nat} \rrbracket \implies \text{map}(\text{succ}, \text{upt}(m, n)) = \text{upt}(\text{succ}(m), \text{succ}(n))$
 $\langle proof \rangle$

lemma *nth-map* $[rule-format, simp]:$

$xs: list(A) \implies$
 $\forall n \in \text{nat}. n < \text{length}(xs) \dashv\dashv \text{nth}(n, \text{map}(f, xs)) = f(\text{nth}(n, xs))$
 $\langle proof \rangle$

lemma *nth-map-upt* $[rule-format]:$

$\llbracket m:\text{nat}; n:\text{nat} \rrbracket \implies$
 $\forall i \in \text{nat}. i < n \# - m \dashv\dashv \text{nth}(i, \text{map}(f, \text{upt}(m, n))) = f(m \# + i)$
 $\langle proof \rangle$

definition

sublist $:: [i, i] \implies i$ **where**

$sublist(xs, A) ==$
 $map(fst, (filter(\%p. snd(p): A, zip(xs, upt(0, length(xs)))))$

lemma *sublist-0* [simp]: $xs: list(A) ==> sublist(xs, 0) = Nil$
 $\langle proof \rangle$

lemma *sublist-Nil* [simp]: $sublist(Nil, A) = Nil$
 $\langle proof \rangle$

lemma *sublist-shift-lemma*:
 $[| xs: list(B); i: nat |] ==>$
 $map(fst, filter(\%p. snd(p): A, zip(xs, upt(i, i \# + length(xs))))) =$
 $map(fst, filter(\%p. snd(p): nat \& snd(p) \# + i: A, zip(xs, upt(0, length(xs)))))$
 $\langle proof \rangle$

lemma *sublist-type* [simp, TC]:
 $xs: list(B) ==> sublist(xs, A): list(B)$
 $\langle proof \rangle$

lemma *upt-add-eq-append2*:
 $[| i: nat; j: nat |] ==> upt(0, i \# + j) = upt(0, i) @ upt(i, i \# + j)$
 $\langle proof \rangle$

lemma *sublist-append*:
 $[| xs: list(B); ys: list(B) |] ==>$
 $sublist(xs @ ys, A) = sublist(xs, A) @ sublist(ys, \{j: nat. j \# + length(xs): A\})$
 $\langle proof \rangle$

lemma *sublist-Cons*:
 $[| xs: list(B); x: B |] ==>$
 $sublist(Cons(x, xs), A) =$
 $(if 0: A then [x] else []) @ sublist(xs, \{j: nat. succ(j) : A\})$
 $\langle proof \rangle$

lemma *sublist-singleton* [simp]:
 $sublist([x], A) = (if 0 : A then [x] else [])$
 $\langle proof \rangle$

lemma *sublist-upt-eq-take* [rule-format, simp]:
 $xs: list(A) ==> ALL n: nat. sublist(xs, n) = take(n, xs)$
 $\langle proof \rangle$

lemma *sublist-Int-eq*:
 $xs : list(B) ==> sublist(xs, A \cap nat) = sublist(xs, A)$
 $\langle proof \rangle$

Repetition of a List Element

consts *repeat* :: $[i, i] ==> i$

primrec

$repeat(a, 0) = []$

$repeat(a, succ(n)) = Cons(a, repeat(a, n))$

lemma *length-repeat*: $n \in nat \implies length(repeat(a, n)) = n$
 $\langle proof \rangle$

lemma *repeat-succ-app*: $n \in nat \implies repeat(a, succ(n)) = repeat(a, n) @ [a]$
 $\langle proof \rangle$

lemma *repeat-type* [TC]: $[a \in A; n \in nat] \implies repeat(a, n) \in list(A)$
 $\langle proof \rangle$

end

29 EquivClass: Equivalence Relations

theory *EquivClass* **imports** *Trancl Perm* **begin**

definition

quotient :: $[i, i] \Rightarrow i$ (**infixl** $'/'$ 90) **where**
 $A / r == \{r''\{x\} . x:A\}$

definition

congruent :: $[i, i \Rightarrow i] \Rightarrow o$ **where**
 $congruent(r, b) == ALL y z. <y, z>:r \longrightarrow b(y)=b(z)$

definition

congruent2 :: $[i, i, [i, i] \Rightarrow i] \Rightarrow o$ **where**
 $congruent2(r1, r2, b) == ALL y1 z1 y2 z2.$
 $<y1, z1>:r1 \longrightarrow <y2, z2>:r2 \longrightarrow b(y1, y2) = b(z1, z2)$

abbreviation

RESPECTS :: $[i \Rightarrow i, i] \Rightarrow o$ (**infixr** *respects* 80) **where**
 $f respects r == congruent(r, f)$

abbreviation

RESPECTS2 :: $[i \Rightarrow i \Rightarrow i, i] \Rightarrow o$ (**infixr** *respects2* 80) **where**
 $f respects2 r == congruent2(r, r, f)$
 — Abbreviation for the common case where the relations are identical

29.1 Suppes, Theorem 70: r is an equiv relation iff $converse(r) \circ r = r$

lemma *sym-trans-comp-subset*:

$[sym(r); trans(r)] \implies converse(r) \circ r \leq r$
 $\langle proof \rangle$

lemma *refl-comp-subset*:

$\llbracket \text{refl}(A,r); r \leq A * A \rrbracket \implies r \leq \text{converse}(r) \text{ } O \text{ } r$
 $\langle \text{proof} \rangle$

lemma *equiv-comp-eq*:

$\text{equiv}(A,r) \implies \text{converse}(r) \text{ } O \text{ } r = r$
 $\langle \text{proof} \rangle$

lemma *comp-equivI*:

$\llbracket \text{converse}(r) \text{ } O \text{ } r = r; \text{domain}(r) = A \rrbracket \implies \text{equiv}(A,r)$
 $\langle \text{proof} \rangle$

lemma *equiv-class-subset*:

$\llbracket \text{sym}(r); \text{trans}(r); \langle a,b \rangle: r \rrbracket \implies r''\{a\} \leq r''\{b\}$
 $\langle \text{proof} \rangle$

lemma *equiv-class-eq*:

$\llbracket \text{equiv}(A,r); \langle a,b \rangle: r \rrbracket \implies r''\{a\} = r''\{b\}$
 $\langle \text{proof} \rangle$

lemma *equiv-class-self*:

$\llbracket \text{equiv}(A,r); a: A \rrbracket \implies a: r''\{a\}$
 $\langle \text{proof} \rangle$

lemma *subset-equiv-class*:

$\llbracket \text{equiv}(A,r); r''\{b\} \leq r''\{a\}; b: A \rrbracket \implies \langle a,b \rangle: r$
 $\langle \text{proof} \rangle$

lemma *eq-equiv-class*: $\llbracket r''\{a\} = r''\{b\}; \text{equiv}(A,r); b: A \rrbracket \implies \langle a,b \rangle: r$
 $\langle \text{proof} \rangle$

lemma *equiv-class-nondisjoint*:

$\llbracket \text{equiv}(A,r); x: (r''\{a\} \text{ Int } r''\{b\}) \rrbracket \implies \langle a,b \rangle: r$
 $\langle \text{proof} \rangle$

lemma *equiv-type*: $\text{equiv}(A,r) \implies r \leq A * A$

$\langle \text{proof} \rangle$

lemma *equiv-class-eq-iff*:

$\text{equiv}(A,r) \implies \langle x,y \rangle: r \iff r''\{x\} = r''\{y\} \ \& \ x:A \ \& \ y:A$
 $\langle \text{proof} \rangle$

lemma *eq-equiv-class-iff*:

$\llbracket \text{equiv}(A,r); x:A; y:A \rrbracket \implies r''\{x\} = r''\{y\} \iff \langle x,y \rangle : r$
 $\langle \text{proof} \rangle$

lemma *quotientI* [TC]: $x:A \implies r''\{x\} : A//r$

$\langle \text{proof} \rangle$

lemma *quotientE*:

$\llbracket X : A//r; !!x. \llbracket X = r''\{x\}; x:A \rrbracket \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *Union-quotient*:

$\text{equiv}(A,r) \implies \text{Union}(A//r) = A$
 $\langle \text{proof} \rangle$

lemma *quotient-disj*:

$\llbracket \text{equiv}(A,r); X : A//r; Y : A//r \rrbracket \implies X=Y \mid (X \text{ Int } Y \leq 0)$
 $\langle \text{proof} \rangle$

29.2 Defining Unary Operations upon Equivalence Classes

lemma *UN-equiv-class*:

$\llbracket \text{equiv}(A,r); b \text{ respects } r; a:A \rrbracket \implies (\text{UN } x:r''\{a\}. b(x)) = b(a)$
 $\langle \text{proof} \rangle$

lemma *UN-equiv-class-type*:

$\llbracket \text{equiv}(A,r); b \text{ respects } r; X : A//r; !!x. x : A \rrbracket \implies b(x) : B$
 $\implies (\text{UN } x:X. b(x)) : B$
 $\langle \text{proof} \rangle$

lemma *UN-equiv-class-inject*:

$\llbracket \text{equiv}(A,r); b \text{ respects } r;$
 $(\text{UN } x:X. b(x)) = (\text{UN } y:Y. b(y)); X : A//r; Y : A//r;$
 $!!x y. \llbracket x:A; y:A; b(x)=b(y) \rrbracket \implies \langle x,y \rangle : r \rrbracket$
 $\implies X=Y$
 $\langle \text{proof} \rangle$

29.3 Defining Binary Operations upon Equivalence Classes

lemma *congruent2-implies-congruent*:

$\llbracket \text{equiv}(A,r1); \text{congruent2}(r1,r2,b); a:A \rrbracket \implies \text{congruent}(r2,b(a))$
 $\langle \text{proof} \rangle$

lemma *congruent2-implies-congruent-UN*:

$$[\text{equiv}(A1, r1); \text{equiv}(A2, r2); \text{congruent2}(r1, r2, b); a: A2] \implies$$

$$\text{congruent}(r1, \%x1. \bigcup x2 \in r2^{\text{``}\{a\}}. b(x1, x2))$$

$$\langle \text{proof} \rangle$$

lemma *UN-equiv-class2*:

$$[\text{equiv}(A1, r1); \text{equiv}(A2, r2); \text{congruent2}(r1, r2, b); a1: A1; a2: A2]$$

$$\implies (\bigcup x1 \in r1^{\text{``}\{a1\}}. \bigcup x2 \in r2^{\text{``}\{a2\}}. b(x1, x2)) = b(a1, a2)$$

$$\langle \text{proof} \rangle$$

lemma *UN-equiv-class-type2*:

$$[\text{equiv}(A, r); b \text{ respects2 } r;$$

$$X1: A//r; X2: A//r;$$

$$!!x1\ x2. [x1: A; x2: A] \implies b(x1, x2) : B$$

$$] \implies (UN\ x1:X1. UN\ x2:X2. b(x1, x2)) : B$$

$$\langle \text{proof} \rangle$$

lemma *congruent2I*:

$$[\text{equiv}(A1, r1); \text{equiv}(A2, r2);$$

$$!!\ y\ z\ w. [w \in A2; \langle y, z \rangle \in r1] \implies b(y, w) = b(z, w);$$

$$!!\ y\ z\ w. [w \in A1; \langle y, z \rangle \in r2] \implies b(w, y) = b(w, z)$$

$$] \implies \text{congruent2}(r1, r2, b)$$

$$\langle \text{proof} \rangle$$

lemma *congruent2-commuteI*:

assumes *equivA*: $\text{equiv}(A, r)$
and *commute*: $!!\ y\ z. [y: A; z: A] \implies b(y, z) = b(z, y)$
and *cong*: $!!\ y\ z\ w. [w: A; \langle y, z \rangle: r] \implies b(w, y) = b(w, z)$
shows *b respects2 r*

$$\langle \text{proof} \rangle$$

lemma *congruent-commuteI*:

$$[\text{equiv}(A, r); Z: A//r;$$

$$!!w. [w: A] \implies \text{congruent}(r, \%z. b(w, z));$$

$$!!x\ y. [x: A; y: A] \implies b(y, x) = b(x, y)$$

$$] \implies \text{congruent}(r, \%w. UN\ z: Z. b(w, z))$$

$$\langle \text{proof} \rangle$$

end

30 Int-ZF: The Integers as Equivalence Classes Over Pairs of Natural Numbers

theory *Int-ZF* **imports** *EquivClass ArithSimp* **begin**

definition

intrel :: *i* **where**
intrel == {*p* : (*nat***nat*)*(*nat***nat*).
 $\exists x1\ y1\ x2\ y2. p = \langle \langle x1, y1 \rangle, \langle x2, y2 \rangle \rangle \ \& \ x1 \# + y2 = x2 \# + y1$ }

definition

int :: *i* **where**
int == (*nat***nat*)//*intrel*

definition

int-of :: *i* => *i* — coercion from *nat* to *int* (*\$#* - [80] 80) **where**
\$# *m* == *intrel* “ {<*nativify*(*m*), 0>}

definition

intify :: *i* => *i* — coercion from ANYTHING to *int* **where**
intify(*m*) == *if m : int then m else \$#0*

definition

raw-zminus :: *i* => *i* **where**
raw-zminus(*z*) == $\bigcup \langle x, y \rangle \in z. \text{intrel} \{ \langle y, x \rangle \}$

definition

zminus :: *i* => *i* (*\$-* - [80] 80) **where**
\$- *z* == *raw-zminus* (*intify*(*z*))

definition

znegative :: *i* => *o* **where**
znegative(*z*) == $\exists x\ y. x < y \ \& \ y \in \text{nat} \ \& \ \langle x, y \rangle \in z$

definition

iszero :: *i* => *o* **where**
iszero(*z*) == *z* = *\$# 0*

definition

raw-nat-of :: *i* => *i* **where**
raw-nat-of(*z*) == *nativify* ($\bigcup \langle x, y \rangle \in z. x \# - y$)

definition

nat-of :: *i* => *i* **where**
nat-of(*z*) == *raw-nat-of* (*intify*(*z*))

definition

zmagnitude :: *i* => *i* **where**
— could be replaced by an absolute value function from *int* to *int*?
zmagnitude(*z*) ==
THE *m. m* ∈ *nat* & ((\sim *znegative*(*z*) & *z* = *\$# m*) |
(*znegative*(*z*) & *\$-* *z* = *\$# m*))

definition

raw-zmult :: $[i, i] => i$ **where**

raw-zmult(*z1*, *z2*) ==
 $\bigcup p1 \in z1. \bigcup p2 \in z2. \text{split}(\%x1 \ y1. \text{split}(\%x2 \ y2. \\ \text{intrel}''\{<x1 \#*x2 \ \#+ \ y1 \#*y2, \ x1 \#*y2 \ \#+ \ y1 \#*x2>\}, p2), p1)$

definition

zmult :: $[i, i] => i$ (**infixl** \$* 70) **where**
z1 \$* *z2* == *raw-zmult* (*intify*(*z1*), *intify*(*z2*))

definition

raw-zadd :: $[i, i] => i$ **where**
raw-zadd (*z1*, *z2*) ==
 $\bigcup z1 \in z1. \bigcup z2 \in z2. \text{let } <x1, y1> = z1; <x2, y2> = z2 \\ \text{in intrel}''\{<x1 \ \#+ \ x2, \ y1 \ \#+ \ y2>\}$

definition

zadd :: $[i, i] => i$ (**infixl** \$+ 65) **where**
z1 \$+ *z2* == *raw-zadd* (*intify*(*z1*), *intify*(*z2*))

definition

zdiff :: $[i, i] => i$ (**infixl** \$- 65) **where**
z1 \$- *z2* == *z1* \$+ *zminus*(*z2*)

definition

zless :: $[i, i] => o$ (**infixl** \$< 50) **where**
z1 \$< *z2* == *znegative*(*z1* \$- *z2*)

definition

zle :: $[i, i] => o$ (**infixl** \$<= 50) **where**
z1 \$<= *z2* == *z1* \$< *z2* | *intify*(*z1*) = *intify*(*z2*)

notation (*xsymbols*)

zmult (**infixl** \$ \times 70) **and**
zle (**infixl** \$ \leq 50) — less than or equals

notation (*HTML output*)

zmult (**infixl** \$ \times 70) **and**
zle (**infixl** \$ \leq 50)

declare *quotientE* [*elim!*]

30.1 Proving that *intrel* is an equivalence relation

lemma *intrel-iff* [*simp*]:

$<<x1, y1>, <x2, y2>>: \text{intrel } <->$

$x1 \in nat \ \& \ y1 \in nat \ \& \ x2 \in nat \ \& \ y2 \in nat \ \& \ x1 \# + y2 = x2 \# + y1$
 $\langle proof \rangle$

lemma *intrelI* [*intro!*]:
 $[| \ x1 \# + y2 = x2 \# + y1; \ x1 \in nat; \ y1 \in nat; \ x2 \in nat; \ y2 \in nat \ |]$
 $==> \ \langle \langle x1, y1 \rangle, \langle x2, y2 \rangle \rangle : \text{intrel}$
 $\langle proof \rangle$

lemma *intrelE* [*elim!*]:
 $[| \ p : \text{intrel};$
 $!!x1 \ y1 \ x2 \ y2. \ [| \ p = \langle \langle x1, y1 \rangle, \langle x2, y2 \rangle \rangle; \ x1 \# + y2 = x2 \# + y1;$
 $\quad \quad \quad \ x1 \in nat; \ y1 \in nat; \ x2 \in nat; \ y2 \in nat \ |] ==> \ Q \ |]$
 $==> \ Q$
 $\langle proof \rangle$

lemma *int-trans-lemma*:
 $[| \ x1 \# + y2 = x2 \# + y1; \ x2 \# + y3 = x3 \# + y2 \ |] ==> \ x1 \# + y3 = x3 \# + y1$
 $\langle proof \rangle$

lemma *equiv-intrel*: *equiv*(*nat*nat*, *intrel*)
 $\langle proof \rangle$

lemma *image-intrel-int*: $[| \ m \in nat; \ n \in nat \ |] ==> \ \text{intrel} \ \langle \langle m, n \rangle \rangle : \text{int}$
 $\langle proof \rangle$

declare *equiv-intrel* [*THEN eq-equiv-class-iff*, *simp*]
declare *conj-cong* [*cong*]

lemmas *eq-intrelD* = *eq-equiv-class* [*OF - equiv-intrel*]

lemma *int-of-type* [*simp*, *TC*]: $\$ \# m : \text{int}$
 $\langle proof \rangle$

lemma *int-of-eq* [*iff*]: $(\$ \# m = \$ \# n) <-> \ \text{nativify}(m) = \text{nativify}(n)$
 $\langle proof \rangle$

lemma *int-of-inject*: $[| \ \$ \# m = \$ \# n; \ m \in nat; \ n \in nat \ |] ==> \ m = n$
 $\langle proof \rangle$

lemma *intify-in-int* [*iff*, *TC*]: *intify*(*x*) : *int*
 $\langle proof \rangle$

lemma *intify-ident* [*simp*]: $n : \text{int} ==> \ \text{intify}(n) = n$

$\langle proof \rangle$

30.2 Collapsing rules: to remove *intify* from arithmetic expressions

lemma *intify-idem* [simp]: $intify(intify(x)) = intify(x)$
 $\langle proof \rangle$

lemma *int-of-natify* [simp]: $\$ \# (natify(m)) = \$ \# m$
 $\langle proof \rangle$

lemma *zminus-intify* [simp]: $\$ - (intify(m)) = \$ - m$
 $\langle proof \rangle$

lemma *zadd-intify1* [simp]: $intify(x) \$ + y = x \$ + y$
 $\langle proof \rangle$

lemma *zadd-intify2* [simp]: $x \$ + intify(y) = x \$ + y$
 $\langle proof \rangle$

lemma *zdiff-intify1* [simp]: $intify(x) \$ - y = x \$ - y$
 $\langle proof \rangle$

lemma *zdiff-intify2* [simp]: $x \$ - intify(y) = x \$ - y$
 $\langle proof \rangle$

lemma *zmult-intify1* [simp]: $intify(x) \$ * y = x \$ * y$
 $\langle proof \rangle$

lemma *zmult-intify2* [simp]: $x \$ * intify(y) = x \$ * y$
 $\langle proof \rangle$

lemma *zless-intify1* [simp]: $intify(x) \$ < y \leftrightarrow x \$ < y$
 $\langle proof \rangle$

lemma *zless-intify2* [simp]: $x \$ < intify(y) \leftrightarrow x \$ < y$
 $\langle proof \rangle$

lemma *zle-intify1* [simp]: $intify(x) \$ \leq y \leftrightarrow x \$ \leq y$
 $\langle proof \rangle$

lemma *zle-intify2* [simp]: $x \leq \text{intify}(y) \leftrightarrow x \leq y$
 <proof>

30.3 *zminus*: unary negation on *int*

lemma *zminus-congruent*: $(\%<x,y>. \text{intrel} \{<y,x>\})$ respects *intrel*
 <proof>

lemma *raw-zminus-type*: $z : \text{int} \implies \text{raw-zminus}(z) : \text{int}$
 <proof>

lemma *zminus-type* [TC,iff]: $\$-z : \text{int}$
 <proof>

lemma *raw-zminus-inject*:
 $[\text{raw-zminus}(z) = \text{raw-zminus}(w); z : \text{int}; w : \text{int}] \implies z = w$
 <proof>

lemma *zminus-inject-intify* [dest!]: $\$-z = \$-w \implies \text{intify}(z) = \text{intify}(w)$
 <proof>

lemma *zminus-inject*: $[\$-z = \$-w; z : \text{int}; w : \text{int}] \implies z = w$
 <proof>

lemma *raw-zminus*:
 $[\text{raw-zminus}(z) = \text{raw-zminus}(w); z : \text{int}; w : \text{int}] \implies z = w$
 <proof>

lemma *zminus*:
 $[\text{zminus}(z) = \text{zminus}(w); z : \text{int}; w : \text{int}] \implies z = w$
 <proof>

lemma *raw-zminus-zminus*: $z : \text{int} \implies \text{raw-zminus}(\text{raw-zminus}(z)) = z$
 <proof>

lemma *zminus-zminus-intify* [simp]: $\$-(\$-z) = \text{intify}(z)$
 <proof>

lemma *zminus-int0* [simp]: $\$-(\$ \# 0) = \$ \# 0$
 <proof>

lemma *zminus-zminus*: $z : \text{int} \implies \$-(\$-z) = z$
 <proof>

30.4 *znegative*: the test for negative integers

lemma *znegative*: $[\text{znegative}(z) = \text{znegative}(w); z : \text{int}; w : \text{int}] \implies z < w \leftrightarrow w < z$
 <proof>

lemma *not-znegative-int-of* [iff]: $\sim \text{znegative}(\$ \# n)$

<proof>

lemma *znegative-zminus-int-of* [simp]: $\text{znegative}(\$ - \$ \# \text{succ}(n))$

<proof>

lemma *not-znegative-imp-zero*: $\sim \text{znegative}(\$ - \$ \# n) \implies \text{natify}(n)=0$

<proof>

30.5 *nat-of*: Coercion of an Integer to a Natural Number

lemma *nat-of-intify* [simp]: $\text{nat-of}(\text{intify}(z)) = \text{nat-of}(z)$

<proof>

lemma *nat-of-congruent*: $(\lambda x. (\lambda \langle x, y \rangle. x \# - y)(x))$ respects *intrel*

<proof>

lemma *raw-nat-of*:

$[[x \in \text{nat}; y \in \text{nat}]] \implies \text{raw-nat-of}(\text{intrel} \{ \langle x, y \rangle \}) = x \# - y$

<proof>

lemma *raw-nat-of-int-of*: $\text{raw-nat-of}(\$ \# n) = \text{natify}(n)$

<proof>

lemma *nat-of-int-of* [simp]: $\text{nat-of}(\$ \# n) = \text{natify}(n)$

<proof>

lemma *raw-nat-of-type*: $\text{raw-nat-of}(z) \in \text{nat}$

<proof>

lemma *nat-of-type* [iff, TC]: $\text{nat-of}(z) \in \text{nat}$

<proof>

30.6 *zmagnitude*: magnitide of an integer, as a natural number

lemma *zmagnitude-int-of* [simp]: $\text{zmagnitude}(\$ \# n) = \text{natify}(n)$

<proof>

lemma *natify-int-of-eq*: $\text{natify}(x)=n \implies \$ \# x = \$ \# n$

<proof>

lemma *zmagnitude-zminus-int-of* [simp]: $\text{zmagnitude}(\$ - \$ \# n) = \text{natify}(n)$

<proof>

lemma *zmagnitude-type* [iff, TC]: $\text{zmagnitude}(z) \in \text{nat}$

<proof>

lemma *not-zneg-int-of*:

$\llbracket z : \text{int}; \sim \text{znegative}(z) \rrbracket \implies \exists n \in \text{nat}. z = \$\# n$
 $\langle \text{proof} \rangle$

lemma *not-zneg-mag [simp]*:

$\llbracket z : \text{int}; \sim \text{znegative}(z) \rrbracket \implies \$\# (\text{zmagnitude}(z)) = z$
 $\langle \text{proof} \rangle$

lemma *zneg-int-of*:

$\llbracket \text{znegative}(z); z : \text{int} \rrbracket \implies \exists n \in \text{nat}. z = \$- (\$ \# \text{succ}(n))$
 $\langle \text{proof} \rangle$

lemma *zneg-mag [simp]*:

$\llbracket \text{znegative}(z); z : \text{int} \rrbracket \implies \$\# (\text{zmagnitude}(z)) = \$- z$
 $\langle \text{proof} \rangle$

lemma *int-cases*: $z : \text{int} \implies \exists n \in \text{nat}. z = \$\# n \mid z = \$- (\$ \# \text{succ}(n))$

$\langle \text{proof} \rangle$

lemma *not-zneg-raw-nat-of*:

$\llbracket \sim \text{znegative}(z); z : \text{int} \rrbracket \implies \$\# (\text{raw-nat-of}(z)) = z$
 $\langle \text{proof} \rangle$

lemma *not-zneg-nat-of-intify*:

$\sim \text{znegative}(\text{intify}(z)) \implies \$\# (\text{nat-of}(z)) = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *not-zneg-nat-of*: $\llbracket \sim \text{znegative}(z); z : \text{int} \rrbracket \implies \$\# (\text{nat-of}(z)) = z$

$\langle \text{proof} \rangle$

lemma *zneg-nat-of [simp]*: $\text{znegative}(\text{intify}(z)) \implies \text{nat-of}(z) = 0$

$\langle \text{proof} \rangle$

30.7 op \$+: addition on int

Congruence Property for Addition

lemma *zadd-congruent2*:

$(\%z1 \ z2. \text{let } \langle x1, y1 \rangle = z1; \langle x2, y2 \rangle = z2$
 $\quad \text{in } \text{intrel}''\{\langle x1 \# + x2, y1 \# + y2 \rangle\})$
 respects2 intrel
 $\langle \text{proof} \rangle$

lemma *raw-zadd-type*: $\llbracket z : \text{int}; w : \text{int} \rrbracket \implies \text{raw-zadd}(z, w) : \text{int}$

$\langle \text{proof} \rangle$

lemma *zadd-type [iff, TC]*: $z \# + w : \text{int}$

$\langle \text{proof} \rangle$

lemma *raw-zadd*:

$$[| x1 \in nat; y1 \in nat; x2 \in nat; y2 \in nat |]$$

$$\implies \text{raw-zadd } (\text{intrel}''\{\langle x1, y1 \rangle\}, \text{intrel}''\{\langle x2, y2 \rangle\}) =$$

$$\text{intrel}''\{\langle x1 \# + x2, y1 \# + y2 \rangle\}$$

$$\langle \text{proof} \rangle$$

lemma *zadd*:

$$[| x1 \in nat; y1 \in nat; x2 \in nat; y2 \in nat |]$$

$$\implies (\text{intrel}''\{\langle x1, y1 \rangle\} \$+ (\text{intrel}''\{\langle x2, y2 \rangle\}) =$$

$$\text{intrel}''\{\langle x1 \# + x2, y1 \# + y2 \rangle\}$$

$$\langle \text{proof} \rangle$$

lemma *raw-zadd-int0*: $z : int \implies \text{raw-zadd } (\$ \# 0, z) = z$

$$\langle \text{proof} \rangle$$

lemma *zadd-int0-intify [simp]*: $\$ \# 0 \$+ z = \text{intify}(z)$

$$\langle \text{proof} \rangle$$

lemma *zadd-int0*: $z : int \implies \$ \# 0 \$+ z = z$

$$\langle \text{proof} \rangle$$

lemma *raw-zminus-zadd-distrib*:

$$[| z : int; w : int |] \implies \$- \text{raw-zadd}(z, w) = \text{raw-zadd}(\$- z, \$- w)$$

$$\langle \text{proof} \rangle$$

lemma *zminus-zadd-distrib [simp]*: $\$- (z \$+ w) = \$- z \$+ \$- w$

$$\langle \text{proof} \rangle$$

lemma *raw-zadd-commute*:

$$[| z : int; w : int |] \implies \text{raw-zadd}(z, w) = \text{raw-zadd}(w, z)$$

$$\langle \text{proof} \rangle$$

lemma *zadd-commute*: $z \$+ w = w \$+ z$

$$\langle \text{proof} \rangle$$

lemma *raw-zadd-assoc*:

$$[| z1 : int; z2 : int; z3 : int |]$$

$$\implies \text{raw-zadd } (\text{raw-zadd}(z1, z2), z3) = \text{raw-zadd}(z1, \text{raw-zadd}(z2, z3))$$

$$\langle \text{proof} \rangle$$

lemma *zadd-assoc*: $(z1 \$+ z2) \$+ z3 = z1 \$+ (z2 \$+ z3)$

$$\langle \text{proof} \rangle$$

lemma *zadd-left-commute*: $z1 \$+ (z2 \$+ z3) = z2 \$+ (z1 \$+ z3)$

$$\langle \text{proof} \rangle$$

lemmas *zadd-ac = zadd-assoc zadd-commute zadd-left-commute*

lemma *int-of-add*: $\$ \# (m \# + n) = (\$ \# m) \$ + (\$ \# n)$
 $\langle proof \rangle$

lemma *int-succ-int-1*: $\$ \# succ(m) = \$ \# 1 \$ + (\$ \# m)$
 $\langle proof \rangle$

lemma *int-of-diff*:
 $[[m \in nat; \ n \leq m]] ==> \$ \# (m \# - n) = (\$ \# m) \$ - (\$ \# n)$
 $\langle proof \rangle$

lemma *raw-zadd-zminus-inverse*: $z : int ==> raw-zadd (z, \$ - z) = \$ \# 0$
 $\langle proof \rangle$

lemma *zadd-zminus-inverse* [simp]: $z \$ + (\$ - z) = \$ \# 0$
 $\langle proof \rangle$

lemma *zadd-zminus-inverse2* [simp]: $(\$ - z) \$ + z = \$ \# 0$
 $\langle proof \rangle$

lemma *zadd-int0-right-intify* [simp]: $z \$ + \$ \# 0 = intify(z)$
 $\langle proof \rangle$

lemma *zadd-int0-right*: $z : int ==> z \$ + \$ \# 0 = z$
 $\langle proof \rangle$

30.8 *op* \times : Integer Multiplication

Congruence property for multiplication

lemma *zmult-congruent2*:
 $(\%p1 \ p2. \ split(\%x1 \ y1. \ split(\%x2 \ y2. \$
 $\quad \quad \quad intrel\{\{<x1 \# * x2 \# + y1 \# * y2, x1 \# * y2 \# + y1 \# * x2>\}, p2), p1))$
 $\quad \quad \quad respects2 \ intrel$
 $\langle proof \rangle$

lemma *raw-zmult-type*: $[[z : int; \ w : int]] ==> raw-zmult(z, w) : int$
 $\langle proof \rangle$

lemma *zmult-type* [iff, TC]: $z \$ * w : int$
 $\langle proof \rangle$

lemma *raw-zmult*:
 $[[x1 \in nat; \ y1 \in nat; \ x2 \in nat; \ y2 \in nat]]$
 $==> raw-zmult(intrel\{\{<x1, y1>\}, intrel\{\{<x2, y2>\}\}) =$
 $\quad \quad \quad intrel\{\{<x1 \# * x2 \# + y1 \# * y2, x1 \# * y2 \# + y1 \# * x2>\}$
 $\langle proof \rangle$

lemma *zmult*:
 $[[x1 \in nat; \ y1 \in nat; \ x2 \in nat; \ y2 \in nat]]$

$$\implies (\text{intrel}^{\prime\prime}\{\langle x1, y1 \rangle\}) \$* (\text{intrel}^{\prime\prime}\{\langle x2, y2 \rangle\}) =$$

$$\text{intrel}^{\prime\prime} \{ \langle x1 \#* x2 \# + y1 \#* y2, x1 \#* y2 \# + y1 \#* x2 \rangle \}$$

$$\langle \text{proof} \rangle$$

lemma *raw-zmult-int0*: $z : \text{int} \implies \text{raw-zmult } (\$ \# 0, z) = \$ \# 0$
 $\langle \text{proof} \rangle$

lemma *zmult-int0 [simp]*: $\$ \# 0 \$* z = \$ \# 0$
 $\langle \text{proof} \rangle$

lemma *raw-zmult-int1*: $z : \text{int} \implies \text{raw-zmult } (\$ \# 1, z) = z$
 $\langle \text{proof} \rangle$

lemma *zmult-int1-intify [simp]*: $\$ \# 1 \$* z = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zmult-int1*: $z : \text{int} \implies \$ \# 1 \$* z = z$
 $\langle \text{proof} \rangle$

lemma *raw-zmult-commute*:

$$[\mid z : \text{int}; \ w : \text{int} \mid] \implies \text{raw-zmult}(z, w) = \text{raw-zmult}(w, z)$$
 $\langle \text{proof} \rangle$

lemma *zmult-commute*: $z \$* w = w \$* z$
 $\langle \text{proof} \rangle$

lemma *raw-zmult-zminus*:

$$[\mid z : \text{int}; \ w : \text{int} \mid] \implies \text{raw-zmult}(\$ - z, w) = \$ - \text{raw-zmult}(z, w)$$
 $\langle \text{proof} \rangle$

lemma *zmult-zminus [simp]*: $(\$ - z) \$* w = \$ - (z \$* w)$
 $\langle \text{proof} \rangle$

lemma *zmult-zminus-right [simp]*: $w \$* (\$ - z) = \$ - (w \$* z)$
 $\langle \text{proof} \rangle$

lemma *raw-zmult-assoc*:

$$[\mid z1 : \text{int}; \ z2 : \text{int}; \ z3 : \text{int} \mid]$$

$$\implies \text{raw-zmult } (\text{raw-zmult}(z1, z2), z3) = \text{raw-zmult}(z1, \text{raw-zmult}(z2, z3))$$
 $\langle \text{proof} \rangle$

lemma *zmult-assoc*: $(z1 \$* z2) \$* z3 = z1 \$* (z2 \$* z3)$
 $\langle \text{proof} \rangle$

lemma *zmult-left-commute*: $z1 \$*(z2 \$* z3) = z2 \$*(z1 \$* z3)$
 $\langle \text{proof} \rangle$

lemmas *zmult-ac = zmult-assoc zmult-commute zmult-left-commute*

lemma *raw-zadd-zmult-distrib:*

$$\begin{aligned} & [| z1: int; z2: int; w: int |] \\ & \implies raw-zmult(raw-zadd(z1, z2), w) = \\ & \quad raw-zadd (raw-zmult(z1, w), raw-zmult(z2, w)) \end{aligned}$$

 $\langle proof \rangle$

lemma *zadd-zmult-distrib:* $(z1 \ \$+ \ z2) \ \$* \ w = (z1 \ \$* \ w) \ \$+ \ (z2 \ \$* \ w)$
 $\langle proof \rangle$

lemma *zadd-zmult-distrib2:* $w \ \$* \ (z1 \ \$+ \ z2) = (w \ \$* \ z1) \ \$+ \ (w \ \$* \ z2)$
 $\langle proof \rangle$

lemmas *int-typechecks =*
int-of-type zminus-type zmagnitude-type zadd-type zmult-type

lemma *zdiff-type [iff, TC]:* $z \ \$- \ w : int$
 $\langle proof \rangle$

lemma *zminus-zdiff-eq [simp]:* $\$- \ (z \ \$- \ y) = y \ \$- \ z$
 $\langle proof \rangle$

lemma *zdiff-zmult-distrib:* $(z1 \ \$- \ z2) \ \$* \ w = (z1 \ \$* \ w) \ \$- \ (z2 \ \$* \ w)$
 $\langle proof \rangle$

lemma *zdiff-zmult-distrib2:* $w \ \$* \ (z1 \ \$- \ z2) = (w \ \$* \ z1) \ \$- \ (w \ \$* \ z2)$
 $\langle proof \rangle$

lemma *zadd-zdiff-eq:* $x \ \$+ \ (y \ \$- \ z) = (x \ \$+ \ y) \ \$- \ z$
 $\langle proof \rangle$

lemma *zdiff-zadd-eq:* $(x \ \$- \ y) \ \$+ \ z = (x \ \$+ \ z) \ \$- \ y$
 $\langle proof \rangle$

30.9 The "Less Than" Relation

lemma *zless-linear-lemma:*

$$[| z: int; w: int |] \implies z \ \$< \ w \mid z = w \mid w \ \$< \ z$$

 $\langle proof \rangle$

lemma *zless-linear:* $z \ \$< \ w \mid intify(z) = intify(w) \mid w \ \$< \ z$
 $\langle proof \rangle$

lemma *zless-not-refl [iff]:* $\sim (z \ \$< \ z)$
 $\langle proof \rangle$

lemma *neq-iff-zless*: $[| x: \text{int}; y: \text{int} |] \implies (x \sim y) \iff (x \$< y \mid y \$< x)$
 $\langle \text{proof} \rangle$

lemma *zless-imp-intify-neq*: $w \$< z \implies \text{intify}(w) \sim \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zless-imp-succ-zadd-lemma*:
 $[| w \$< z; w: \text{int}; z: \text{int} |] \implies (\exists n \in \text{nat}. z = w \$+ \$\#(\text{succ}(n)))$
 $\langle \text{proof} \rangle$

lemma *zless-imp-succ-zadd*:
 $w \$< z \implies (\exists n \in \text{nat}. w \$+ \$\#(\text{succ}(n)) = \text{intify}(z))$
 $\langle \text{proof} \rangle$

lemma *zless-succ-zadd-lemma*:
 $w : \text{int} \implies w \$< w \$+ \$\# \text{succ}(n)$
 $\langle \text{proof} \rangle$

lemma *zless-succ-zadd*: $w \$< w \$+ \$\# \text{succ}(n)$
 $\langle \text{proof} \rangle$

lemma *zless-iff-succ-zadd*:
 $w \$< z \iff (\exists n \in \text{nat}. w \$+ \$\#(\text{succ}(n)) = \text{intify}(z))$
 $\langle \text{proof} \rangle$

lemma *zless-int-of [simp]*: $[| m \in \text{nat}; n \in \text{nat} |] \implies (\$ \# m \$< \$ \# n) \iff (m < n)$
 $\langle \text{proof} \rangle$

lemma *zless-trans-lemma*:
 $[| x \$< y; y \$< z; x: \text{int}; y: \text{int}; z: \text{int} |] \implies x \$< z$
 $\langle \text{proof} \rangle$

lemma *zless-trans*: $[| x \$< y; y \$< z |] \implies x \$< z$
 $\langle \text{proof} \rangle$

lemma *zless-not-sym*: $z \$< w \implies \sim (w \$< z)$
 $\langle \text{proof} \rangle$

lemmas *zless-asm* = *zless-not-sym* [*THEN swap, standard*]

lemma *zless-imp-zle*: $z \$< w \implies z \$<= w$
 $\langle \text{proof} \rangle$

lemma *zle-linear*: $z \$<= w \mid w \$<= z$
 $\langle \text{proof} \rangle$

30.10 Less Than or Equals

lemma *zle-refl*: $z \leq z$
 $\langle proof \rangle$

lemma *zle-eq-refl*: $x=y \implies x \leq y$
 $\langle proof \rangle$

lemma *zle-anti-sym-intify*: $[| x \leq y; y \leq x |] \implies \text{intify}(x) = \text{intify}(y)$
 $\langle proof \rangle$

lemma *zle-anti-sym*: $[| x \leq y; y \leq x; x: \text{int}; y: \text{int} |] \implies x=y$
 $\langle proof \rangle$

lemma *zle-trans-lemma*:
 $[| x: \text{int}; y: \text{int}; z: \text{int}; x \leq y; y \leq z |] \implies x \leq z$
 $\langle proof \rangle$

lemma *zle-trans*: $[| x \leq y; y \leq z |] \implies x \leq z$
 $\langle proof \rangle$

lemma *zle-zless-trans*: $[| i \leq j; j < k |] \implies i < k$
 $\langle proof \rangle$

lemma *zless-zle-trans*: $[| i < j; j \leq k |] \implies i < k$
 $\langle proof \rangle$

lemma *not-zless-iff-zle*: $\sim (z < w) \iff (w \leq z)$
 $\langle proof \rangle$

lemma *not-zle-iff-zless*: $\sim (z \leq w) \iff (w < z)$
 $\langle proof \rangle$

30.11 More subtraction laws (for *zcompare-rls*)

lemma *zdiff-zdiff-eq*: $(x - y) - z = x - (y + z)$
 $\langle proof \rangle$

lemma *zdiff-zdiff-eq2*: $x - (y - z) = (x + z) - y$
 $\langle proof \rangle$

lemma *zdiff-zless-iff*: $(x - y < z) \iff (x < z + y)$
 $\langle proof \rangle$

lemma *zless-zdiff-iff*: $(x < z - y) \iff (x + y < z)$
 $\langle proof \rangle$

lemma *zdiff-eq-iff*: $[| x: \text{int}; z: \text{int} |] \implies (x - y = z) \iff (x = z + y)$
 $\langle proof \rangle$

lemma *eq-zdiff-iff*: $[[\ x: \text{int}; z: \text{int} \]] \implies (x = z\$-y) <-> (x \$+ y = z)$
 $\langle \text{proof} \rangle$

lemma *zdiff-zle-iff-lemma*:
 $[[\ x: \text{int}; z: \text{int} \]] \implies (x\$-y \$<= z) <-> (x \$<= z \$+ y)$
 $\langle \text{proof} \rangle$

lemma *zdiff-zle-iff*: $(x\$-y \$<= z) <-> (x \$<= z \$+ y)$
 $\langle \text{proof} \rangle$

lemma *zle-zdiff-iff-lemma*:
 $[[\ x: \text{int}; z: \text{int} \]] \implies (x \$<= z\$-y) <-> (x \$+ y \$<= z)$
 $\langle \text{proof} \rangle$

lemma *zle-zdiff-iff*: $(x \$<= z\$-y) <-> (x \$+ y \$<= z)$
 $\langle \text{proof} \rangle$

This list of rewrites simplifies (in)equalities by bringing subtractions to the top and then moving negative terms to the other side. Use with *zadd-ac*

lemmas *zcompare-rls* =
zdiff-def [*symmetric*]
zadd-zdiff-eq *zdiff-zadd-eq* *zdiff-zdiff-eq* *zdiff-zdiff-eq2*
zdiff-zless-iff *zless-zdiff-iff* *zdiff-zle-iff* *zle-zdiff-iff*
zdiff-eq-iff *eq-zdiff-iff*

30.12 Monotonicity and Cancellation Results for Instantiation of the CancelNumerals Simprocs

lemma *zadd-left-cancel*:
 $[[\ w: \text{int}; w': \text{int} \]] \implies (z \$+ w' = z \$+ w) <-> (w' = w)$
 $\langle \text{proof} \rangle$

lemma *zadd-left-cancel-intify* [*simp*]:
 $(z \$+ w' = z \$+ w) <-> \text{intify}(w') = \text{intify}(w)$
 $\langle \text{proof} \rangle$

lemma *zadd-right-cancel*:
 $[[\ w: \text{int}; w': \text{int} \]] \implies (w' \$+ z = w \$+ z) <-> (w' = w)$
 $\langle \text{proof} \rangle$

lemma *zadd-right-cancel-intify* [*simp*]:
 $(w' \$+ z = w \$+ z) <-> \text{intify}(w') = \text{intify}(w)$
 $\langle \text{proof} \rangle$

lemma *zadd-right-cancel-zless* [*simp*]: $(w' \$+ z \$< w \$+ z) <-> (w' \$< w)$
 $\langle \text{proof} \rangle$

lemma *zadd-left-cancel-zless* [*simp*]: $(z \$+ w' \$< z \$+ w) <-> (w' \$< w)$
 $\langle \text{proof} \rangle$

lemma *zadd-right-cancel-zle* [*simp*]: $(w' \$+ z \$\leq w \$+ z) <-> w' \$\leq w$
 $\langle proof \rangle$

lemma *zadd-left-cancel-zle* [*simp*]: $(z \$+ w' \$\leq z \$+ w) <-> w' \$\leq w$
 $\langle proof \rangle$

lemmas *zadd-zless-mono1* = *zadd-right-cancel-zless* [*THEN iffD2, standard*]

lemmas *zadd-zless-mono2* = *zadd-left-cancel-zless* [*THEN iffD2, standard*]

lemmas *zadd-zle-mono1* = *zadd-right-cancel-zle* [*THEN iffD2, standard*]

lemmas *zadd-zle-mono2* = *zadd-left-cancel-zle* [*THEN iffD2, standard*]

lemma *zadd-zle-mono*: $[| w' \$\leq w; z' \$\leq z |] ==> w' \$+ z' \$\leq w \$+ z$
 $\langle proof \rangle$

lemma *zadd-zless-mono*: $[| w' \$< w; z' \$\leq z |] ==> w' \$+ z' \$< w \$+ z$
 $\langle proof \rangle$

30.13 Comparison laws

lemma *zminus-zless-zminus* [*simp*]: $(\$- x \$< \$- y) <-> (y \$< x)$
 $\langle proof \rangle$

lemma *zminus-zle-zminus* [*simp*]: $(\$- x \$\leq \$- y) <-> (y \$\leq x)$
 $\langle proof \rangle$

30.13.1 More inequality lemmas

lemma *equation-zminus*: $[| x: int; y: int |] ==> (x = \$- y) <-> (y = \$- x)$
 $\langle proof \rangle$

lemma *zminus-equation*: $[| x: int; y: int |] ==> (\$- x = y) <-> (\$- y = x)$
 $\langle proof \rangle$

lemma *equation-zminus-intify*: $(intify(x) = \$- y) <-> (intify(y) = \$- x)$
 $\langle proof \rangle$

lemma *zminus-equation-intify*: $(\$- x = intify(y)) <-> (\$- y = intify(x))$
 $\langle proof \rangle$

30.13.2 The next several equations are permutative: watch out!

lemma *zless-zminus*: $(x \text{ \$< } \$- y) \text{ <-> } (y \text{ \$< } \$- x)$
 $\langle \text{proof} \rangle$

lemma *zminus-zless*: $(\$- x \text{ \$< } y) \text{ <-> } (\$- y \text{ \$< } x)$
 $\langle \text{proof} \rangle$

lemma *zle-zminus*: $(x \text{ \$<= } \$- y) \text{ <-> } (y \text{ \$<= } \$- x)$
 $\langle \text{proof} \rangle$

lemma *zminus-zle*: $(\$- x \text{ \$<= } y) \text{ <-> } (\$- y \text{ \$<= } x)$
 $\langle \text{proof} \rangle$

end

31 Bin: Arithmetic on Binary Integers

theory *Bin*
imports *Int-ZF Datatype-ZF*
uses (*Tools/numeral-syntax.ML*)
begin

consts *bin* :: *i*
datatype
 bin = *Pls*
 | *Min*
 | *Bit* (*w*: *bin*, *b*: *bool*) (**infixl** *BIT* 90)

consts
 integ-of :: *i* => *i*
 NCons :: [*i*,*i*] => *i*
 bin-succ :: *i* => *i*
 bin-pred :: *i* => *i*
 bin-minus :: *i* => *i*
 bin-adder :: *i* => *i*
 bin-mult :: [*i*,*i*] => *i*

primrec
 integ-of-Pls: *integ-of* (*Pls*) = $\$ \# 0$
 integ-of-Min: *integ-of* (*Min*) = $\$ - (\$ \# 1)$
 integ-of-BIT: *integ-of* (*w BIT b*) = $\$ \# b \text{ \$+ } \text{integ-of}(w) \text{ \$+ } \text{integ-of}(w)$

primrec
 NCons-Pls: *NCons* (*Pls*,*b*) = *cond*(*b*,*Pls BIT b*,*Pls*)
 NCons-Min: *NCons* (*Min*,*b*) = *cond*(*b*,*Min*,*Min BIT b*)

NCons-BIT: $NCons (w \text{ BIT } c, b) = w \text{ BIT } c \text{ BIT } b$

primrec

bin-succ-Pls: $bin-succ (Pls) = Pls \text{ BIT } 1$

bin-succ-Min: $bin-succ (Min) = Pls$

bin-succ-BIT: $bin-succ (w \text{ BIT } b) = cond(b, bin-succ(w) \text{ BIT } 0, NCons(w, 1))$

primrec

bin-pred-Pls: $bin-pred (Pls) = Min$

bin-pred-Min: $bin-pred (Min) = Min \text{ BIT } 0$

bin-pred-BIT: $bin-pred (w \text{ BIT } b) = cond(b, NCons(w, 0), bin-pred(w) \text{ BIT } 1)$

primrec

bin-minus-Pls:

$bin-minus (Pls) = Pls$

bin-minus-Min:

$bin-minus (Min) = Pls \text{ BIT } 1$

bin-minus-BIT:

$bin-minus (w \text{ BIT } b) = cond(b, bin-pred(NCons(bin-minus(w), 0)), bin-minus(w) \text{ BIT } 0)$

primrec

bin-adder-Pls:

$bin-adder (Pls) = (lam w:bin. w)$

bin-adder-Min:

$bin-adder (Min) = (lam w:bin. bin-pred(w))$

bin-adder-BIT:

$bin-adder (v \text{ BIT } x) =$
 $(lam w:bin.$
 $bin-case (v \text{ BIT } x, bin-pred(v \text{ BIT } x),$
 $\%w y. NCons(bin-adder (v) ' cond(x and y, bin-succ(w), w),$
 $x xor y),$
 $w))$

definition

bin-add :: $[i, i] \Rightarrow i$ **where**

$bin-add(v, w) == bin-adder(v) ' w$

primrec

bin-mult-Pls:

$bin-mult (Pls, w) = Pls$

bin-mult-Min:

$bin-mult (Min, w) = bin-minus(w)$

bin-mult-BIT:

$bin-mult (v \text{ BIT } b, w) = cond(b, bin-add(NCons(bin-mult(v, w), 0), w), NCons(bin-mult(v, w), 0))$

syntax

-Int :: *xnum* ==> *i* (-)

$\langle ML \rangle$

declare *bin.intros* [*simp*, *TC*]

lemma *NCons-Pls-0*: *NCons*(*Pls*,0) = *Pls*
 $\langle proof \rangle$

lemma *NCons-Pls-1*: *NCons*(*Pls*,1) = *Pls BIT 1*
 $\langle proof \rangle$

lemma *NCons-Min-0*: *NCons*(*Min*,0) = *Min BIT 0*
 $\langle proof \rangle$

lemma *NCons-Min-1*: *NCons*(*Min*,1) = *Min*
 $\langle proof \rangle$

lemma *NCons-BIT*: *NCons*(*w BIT x*,*b*) = *w BIT x BIT b*
 $\langle proof \rangle$

lemmas *NCons-simps* [*simp*] =
NCons-Pls-0 NCons-Pls-1 NCons-Min-0 NCons-Min-1 NCons-BIT

lemma *integ-of-type* [*TC*]: *w*: *bin* ==> *integ-of*(*w*) : *int*
 $\langle proof \rangle$

lemma *NCons-type* [*TC*]: [*w*: *bin*; *b*: *bool*] ==> *NCons*(*w*,*b*) : *bin*
 $\langle proof \rangle$

lemma *bin-succ-type* [*TC*]: *w*: *bin* ==> *bin-succ*(*w*) : *bin*
 $\langle proof \rangle$

lemma *bin-pred-type* [*TC*]: *w*: *bin* ==> *bin-pred*(*w*) : *bin*
 $\langle proof \rangle$

lemma *bin-minus-type* [*TC*]: *w*: *bin* ==> *bin-minus*(*w*) : *bin*
 $\langle proof \rangle$

lemma *bin-add-type* [*rule-format*, *TC*]:
v: *bin* ==> *ALL w*: *bin*. *bin-add*(*v*,*w*) : *bin*

$\langle \text{proof} \rangle$

lemma *bin-mult-type* [TC]: $[| v: \text{bin}; w: \text{bin} |] \implies \text{bin-mult}(v, w) : \text{bin}$
 $\langle \text{proof} \rangle$

31.0.3 The Carry and Borrow Functions, *bin-succ* and *bin-pred*

lemma *integ-of-NCons* [simp]:
 $[| w: \text{bin}; b: \text{bool} |] \implies \text{integ-of}(\text{NCons}(w, b)) = \text{integ-of}(w \text{ BIT } b)$
 $\langle \text{proof} \rangle$

lemma *integ-of-succ* [simp]:
 $w: \text{bin} \implies \text{integ-of}(\text{bin-succ}(w)) = \$\#1 \$+ \text{integ-of}(w)$
 $\langle \text{proof} \rangle$

lemma *integ-of-pred* [simp]:
 $w: \text{bin} \implies \text{integ-of}(\text{bin-pred}(w)) = \$- (\$ \# 1) \$+ \text{integ-of}(w)$
 $\langle \text{proof} \rangle$

31.0.4 *bin-minus*: Unary Negation of Binary Integers

lemma *integ-of-minus*: $w: \text{bin} \implies \text{integ-of}(\text{bin-minus}(w)) = \$- \text{integ-of}(w)$
 $\langle \text{proof} \rangle$

31.0.5 *bin-add*: Binary Addition

lemma *bin-add-Pls* [simp]: $w: \text{bin} \implies \text{bin-add}(\text{Pls}, w) = w$
 $\langle \text{proof} \rangle$

lemma *bin-add-Pls-right*: $w: \text{bin} \implies \text{bin-add}(w, \text{Pls}) = w$
 $\langle \text{proof} \rangle$

lemma *bin-add-Min* [simp]: $w: \text{bin} \implies \text{bin-add}(\text{Min}, w) = \text{bin-pred}(w)$
 $\langle \text{proof} \rangle$

lemma *bin-add-Min-right*: $w: \text{bin} \implies \text{bin-add}(w, \text{Min}) = \text{bin-pred}(w)$
 $\langle \text{proof} \rangle$

lemma *bin-add-BIT-Pls* [simp]: $\text{bin-add}(v \text{ BIT } x, \text{Pls}) = v \text{ BIT } x$
 $\langle \text{proof} \rangle$

lemma *bin-add-BIT-Min* [simp]: $\text{bin-add}(v \text{ BIT } x, \text{Min}) = \text{bin-pred}(v \text{ BIT } x)$
 $\langle \text{proof} \rangle$

lemma *bin-add-BIT-BIT* [simp]:
 $[| w: \text{bin}; y: \text{bool} |]$
 $\implies \text{bin-add}(v \text{ BIT } x, w \text{ BIT } y) =$
 $\text{NCons}(\text{bin-add}(v, \text{cond}(x \text{ and } y, \text{bin-succ}(w), w)), x \text{ xor } y)$
 $\langle \text{proof} \rangle$

lemma *integ-of-add* [rule-format]:

$v: \text{bin} \implies$
 $ALL\ w: \text{bin}. \text{integ-of}(\text{bin-add}(v, w)) = \text{integ-of}(v) \$+ \text{integ-of}(w)$
 $\langle \text{proof} \rangle$

lemma *diff-integ-of-eq*:

$[| v: \text{bin};\ w: \text{bin} |]$
 $\implies \text{integ-of}(v) \$- \text{integ-of}(w) = \text{integ-of}(\text{bin-add}(v, \text{bin-minus}(w)))$
 $\langle \text{proof} \rangle$

31.0.6 *bin-mult*: Binary Multiplication

lemma *integ-of-mult*:

$[| v: \text{bin};\ w: \text{bin} |]$
 $\implies \text{integ-of}(\text{bin-mult}(v, w)) = \text{integ-of}(v) \$* \text{integ-of}(w)$
 $\langle \text{proof} \rangle$

31.1 Computations

lemma *bin-succ-1*: $\text{bin-succ}(w\ \text{BIT}\ 1) = \text{bin-succ}(w)\ \text{BIT}\ 0$

$\langle \text{proof} \rangle$

lemma *bin-succ-0*: $\text{bin-succ}(w\ \text{BIT}\ 0) = \text{NCons}(w, 1)$

$\langle \text{proof} \rangle$

lemma *bin-pred-1*: $\text{bin-pred}(w\ \text{BIT}\ 1) = \text{NCons}(w, 0)$

$\langle \text{proof} \rangle$

lemma *bin-pred-0*: $\text{bin-pred}(w\ \text{BIT}\ 0) = \text{bin-pred}(w)\ \text{BIT}\ 1$

$\langle \text{proof} \rangle$

lemma *bin-minus-1*: $\text{bin-minus}(w\ \text{BIT}\ 1) = \text{bin-pred}(\text{NCons}(\text{bin-minus}(w), 0))$

$\langle \text{proof} \rangle$

lemma *bin-minus-0*: $\text{bin-minus}(w\ \text{BIT}\ 0) = \text{bin-minus}(w)\ \text{BIT}\ 0$

$\langle \text{proof} \rangle$

lemma *bin-add-BIT-11*: $w: \text{bin} \implies \text{bin-add}(v\ \text{BIT}\ 1, w\ \text{BIT}\ 1) =$

$\text{NCons}(\text{bin-add}(v, \text{bin-succ}(w)), 0)$

$\langle \text{proof} \rangle$

lemma *bin-add-BIT-10*: $w: \text{bin} \implies \text{bin-add}(v\ \text{BIT}\ 1, w\ \text{BIT}\ 0) =$

$\text{NCons}(\text{bin-add}(v, w), 1)$

$\langle \text{proof} \rangle$

lemma *bin-add-BIT-0*: $[\mid w: \text{bin}; \ y: \text{bool} \mid]$
 $\implies \text{bin-add}(v \text{ BIT } 0, w \text{ BIT } y) = \text{NCons}(\text{bin-add}(v, w), y)$
 $\langle \text{proof} \rangle$

lemma *bin-mult-1*: $\text{bin-mult}(v \text{ BIT } 1, w) = \text{bin-add}(\text{NCons}(\text{bin-mult}(v, w), 0), w)$
 $\langle \text{proof} \rangle$

lemma *bin-mult-0*: $\text{bin-mult}(v \text{ BIT } 0, w) = \text{NCons}(\text{bin-mult}(v, w), 0)$
 $\langle \text{proof} \rangle$

lemma *int-of-0*: $\$ \# 0 = \# 0$
 $\langle \text{proof} \rangle$

lemma *int-of-succ*: $\$ \# \text{succ}(n) = \# 1 \$ + \$ \# n$
 $\langle \text{proof} \rangle$

lemma *zminus-0* [simp]: $\$ - \# 0 = \# 0$
 $\langle \text{proof} \rangle$

lemma *zadd-0-intify* [simp]: $\# 0 \$ + z = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zadd-0-right-intify* [simp]: $z \$ + \# 0 = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zmult-1-intify* [simp]: $\# 1 \$ * z = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zmult-1-right-intify* [simp]: $z \$ * \# 1 = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zmult-0* [simp]: $\# 0 \$ * z = \# 0$
 $\langle \text{proof} \rangle$

lemma *zmult-0-right* [simp]: $z \$ * \# 0 = \# 0$
 $\langle \text{proof} \rangle$

lemma *zmult-minus1* [simp]: $\# -1 \$ * z = \$ - z$
 $\langle \text{proof} \rangle$

lemma *zmult-minus1-right* [simp]: $z \$ * \# -1 = \$ - z$
 $\langle \text{proof} \rangle$

31.2 Simplification Rules for Comparison of Binary Numbers

Thanks to Norbert Voelker

lemma *eq-integ-of-eq*:

$$\begin{aligned} & [[\ v: \text{bin};\ w: \text{bin}\]] \\ & \implies ((\text{integ-of}(v)) = \text{integ-of}(w)) \leftrightarrow \\ & \quad \text{iszero}(\text{integ-of}(\text{bin-add}(v, \text{bin-minus}(w)))) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *iszero-integ-of-Pls*: $\text{iszero}(\text{integ-of}(Pls))$

$\langle \text{proof} \rangle$

lemma *nonzero-integ-of-Min*: $\sim \text{iszero}(\text{integ-of}(Min))$

$\langle \text{proof} \rangle$

lemma *iszero-integ-of-BIT*:

$$\begin{aligned} & [[\ w: \text{bin};\ x: \text{bool}\]] \\ & \implies \text{iszero}(\text{integ-of}(w \text{ BIT } x)) \leftrightarrow (x=0 \ \& \ \text{iszero}(\text{integ-of}(w))) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *iszero-integ-of-0*:

$$\begin{aligned} & w: \text{bin} \implies \text{iszero}(\text{integ-of}(w \text{ BIT } 0)) \leftrightarrow \text{iszero}(\text{integ-of}(w)) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *iszero-integ-of-1*: $w: \text{bin} \implies \sim \text{iszero}(\text{integ-of}(w \text{ BIT } 1))$

$\langle \text{proof} \rangle$

lemma *less-integ-of-eq-neg*:

$$\begin{aligned} & [[\ v: \text{bin};\ w: \text{bin}\]] \\ & \implies \text{integ-of}(v) \$< \text{integ-of}(w) \\ & \quad \leftrightarrow \text{znegative}(\text{integ-of}(\text{bin-add}(v, \text{bin-minus}(w)))) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *not-neg-integ-of-Pls*: $\sim \text{znegative}(\text{integ-of}(Pls))$

$\langle \text{proof} \rangle$

lemma *neg-integ-of-Min*: $\text{znegative}(\text{integ-of}(Min))$

$\langle \text{proof} \rangle$

lemma *neg-integ-of-BIT*:

$$\begin{aligned} & [[\ w: \text{bin};\ x: \text{bool}\]] \\ & \implies \text{znegative}(\text{integ-of}(w \text{ BIT } x)) \leftrightarrow \text{znegative}(\text{integ-of}(w)) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *le-integ-of-eq-not-less*:
 $(\text{integ-of}(x) \$<= (\text{integ-of}(w))) <-> \sim (\text{integ-of}(w) \$< (\text{integ-of}(x)))$
 $\langle \text{proof} \rangle$

declare *bin-succ-BIT* [*simp del*]
bin-pred-BIT [*simp del*]
bin-minus-BIT [*simp del*]
NCons-Pls [*simp del*]
NCons-Min [*simp del*]
bin-adder-BIT [*simp del*]
bin-mult-BIT [*simp del*]

declare *integ-of-Pls* [*simp del*] *integ-of-Min* [*simp del*] *integ-of-BIT* [*simp del*]

lemmas *bin-arith-extra-simps* =
integ-of-add [*symmetric*]
integ-of-minus [*symmetric*]
integ-of-mult [*symmetric*]
bin-succ-1 bin-succ-0
bin-pred-1 bin-pred-0
bin-minus-1 bin-minus-0
bin-add-Pls-right bin-add-Min-right
bin-add-BIT-0 bin-add-BIT-10 bin-add-BIT-11
diff-integ-of-eq
bin-mult-1 bin-mult-0 NCons-simps

lemmas *bin-arith-simps* =
bin-pred-Pls bin-pred-Min
bin-succ-Pls bin-succ-Min
bin-add-Pls bin-add-Min
bin-minus-Pls bin-minus-Min
bin-mult-Pls bin-mult-Min
bin-arith-extra-simps

lemmas *bin-rel-simps* =
eq-integ-of-eq iszero-integ-of-Pls nonzero-integ-of-Min
iszero-integ-of-0 iszero-integ-of-1
less-integ-of-eq-neg
not-neg-integ-of-Pls neg-integ-of-Min neg-integ-of-BIT

le-integ-of-eq-not-less

declare *bin-arith-simps* [*simp*]
declare *bin-rel-simps* [*simp*]

lemma *add-integ-of-left* [*simp*]:

$$[| v: \text{bin}; w: \text{bin} |] \\
\implies \text{integ-of}(v) \$+ (\text{integ-of}(w) \$+ z) = (\text{integ-of}(\text{bin-add}(v,w)) \$+ z)$$
 $\langle \text{proof} \rangle$

lemma *mult-integ-of-left* [*simp*]:

$$[| v: \text{bin}; w: \text{bin} |] \\
\implies \text{integ-of}(v) \$* (\text{integ-of}(w) \$* z) = (\text{integ-of}(\text{bin-mult}(v,w)) \$* z)$$
 $\langle \text{proof} \rangle$

lemma *add-integ-of-diff1* [*simp*]:

$$[| v: \text{bin}; w: \text{bin} |] \\
\implies \text{integ-of}(v) \$+ (\text{integ-of}(w) \$- c) = \text{integ-of}(\text{bin-add}(v,w)) \$- (c)$$
 $\langle \text{proof} \rangle$

lemma *add-integ-of-diff2* [*simp*]:

$$[| v: \text{bin}; w: \text{bin} |] \\
\implies \text{integ-of}(v) \$+ (c \$- \text{integ-of}(w)) = \\
\text{integ-of}(\text{bin-add}(v, \text{bin-minus}(w))) \$+ (c)$$
 $\langle \text{proof} \rangle$

declare *int-of-0* [*simp*] *int-of-succ* [*simp*]

lemma *zdiff0* [*simp*]: $\#0 \$- x = \$-x$
 $\langle \text{proof} \rangle$

lemma *zdiff0-right* [*simp*]: $x \$- \#0 = \text{intify}(x)$
 $\langle \text{proof} \rangle$

lemma *zdiff-self* [*simp*]: $x \$- x = \#0$
 $\langle \text{proof} \rangle$

lemma *znegative-iff-zless-0*: $k: \text{int} \implies \text{znegative}(k) <-> k \$< \#0$
 $\langle \text{proof} \rangle$

lemma *zero-zless-imp-znegative-zminus*: $[| \#0 \$< k; k: \text{int} |] \implies \text{znegative}(\$-k)$
 $\langle \text{proof} \rangle$

lemma *zero-zle-int-of* [simp]: $\#0 \leq \#n$
 $\langle \text{proof} \rangle$

lemma *nat-of-0* [simp]: $\text{nat-of}(\#0) = 0$
 $\langle \text{proof} \rangle$

lemma *nat-le-int0-lemma*: $[z \leq \#0; z: \text{int}] \implies \text{nat-of}(z) = 0$
 $\langle \text{proof} \rangle$

lemma *nat-le-int0*: $z \leq \#0 \implies \text{nat-of}(z) = 0$
 $\langle \text{proof} \rangle$

lemma *int-of-eq-0-imp-natify-eq-0*: $\#n = \#0 \implies \text{natify}(n) = 0$
 $\langle \text{proof} \rangle$

lemma *nat-of-zminus-int-of*: $\text{nat-of}(\$- \#n) = 0$
 $\langle \text{proof} \rangle$

lemma *int-of-nat-of*: $\#0 \leq z \implies \# \text{nat-of}(z) = \text{intify}(z)$
 $\langle \text{proof} \rangle$

declare *int-of-nat-of* [simp] *nat-of-zminus-int-of* [simp]

lemma *int-of-nat-of-if*: $\# \text{nat-of}(z) = (\text{if } \#0 \leq z \text{ then } \text{intify}(z) \text{ else } \#0)$
 $\langle \text{proof} \rangle$

lemma *zless-nat-iff-int-zless*: $[m: \text{nat}; z: \text{int}] \implies (m < \text{nat-of}(z)) \iff (\#m \leq z)$
 $\langle \text{proof} \rangle$

lemma *zless-nat-conj-lemma*: $\#0 \leq z \implies (\text{nat-of}(w) < \text{nat-of}(z)) \iff (w \leq z)$
 $\langle \text{proof} \rangle$

lemma *zless-nat-conj*: $(\text{nat-of}(w) < \text{nat-of}(z)) \iff (\#0 \leq z \ \& \ w \leq z)$
 $\langle \text{proof} \rangle$

lemma *integ-of-minus-reorient* [simp]:
 $(\text{integ-of}(w) = \$- x) \iff (\$- x = \text{integ-of}(w))$
 $\langle \text{proof} \rangle$

lemma *integ-of-add-reorient* [simp]:
 $(\text{integ-of}(w) = x \$+ y) \iff (x \$+ y = \text{integ-of}(w))$

$\langle proof \rangle$

lemma *integ-of-diff-reorient* [simp]:

$$(integ-of(w) = x \$- y) <-> (x \$- y = integ-of(w))$$

$\langle proof \rangle$

lemma *integ-of-mult-reorient* [simp]:

$$(integ-of(w) = x \$* y) <-> (x \$* y = integ-of(w))$$

$\langle proof \rangle$

end

theory *IntArith* **imports** *Bin*

uses (*int-arith.ML*)

begin

lemmas [simp] =

zminus-equation [where $y = integ-of(w)$, standard]

equation-zminus [where $x = integ-of(w)$, standard]

lemmas [iff] =

zminus-zless [where $y = integ-of(w)$, standard]

zless-zminus [where $x = integ-of(w)$, standard]

lemmas [iff] =

zminus-zle [where $y = integ-of(w)$, standard]

zle-zminus [where $x = integ-of(w)$, standard]

lemmas [simp] =

Let-def [where $s = integ-of(w)$, standard]

lemma *zless-iff-zdiff-zless-0*: $(x \$< y) <-> (x \$- y \$< \#0)$

$\langle proof \rangle$

lemma *eq-iff-zdiff-eq-0*: $[| x: int; y: int |] ==> (x = y) <-> (x \$- y = \#0)$

$\langle proof \rangle$

lemma *zle-iff-zdiff-zle-0*: $(x \$<= y) <-> (x \$- y \$<= \#0)$

$\langle proof \rangle$

lemma *left-zadd-zmult-distrib*: $i\$*u \$+ (j\$*u \$+ k) = (i\$+j)\$*u \$+ k$
 $\langle proof \rangle$

lemmas *rel-iff-rel-0-rls* =
zless-iff-zdiff-zless-0 [where $y = u \$+ v$, *standard*]
eq-iff-zdiff-eq-0 [where $y = u \$+ v$, *standard*]
zle-iff-zdiff-zle-0 [where $y = u \$+ v$, *standard*]
zless-iff-zdiff-zless-0 [where $y = n$]
eq-iff-zdiff-eq-0 [where $y = n$]
zle-iff-zdiff-zle-0 [where $y = n$]

lemma *eq-add-iff1*: $(i\$*u \$+ m = j\$*u \$+ n) <-> ((i\$-j)\$*u \$+ m = intify(n))$
 $\langle proof \rangle$

lemma *eq-add-iff2*: $(i\$*u \$+ m = j\$*u \$+ n) <-> (intify(m) = (j\$-i)\$*u \$+ n)$
 $\langle proof \rangle$

lemma *less-add-iff1*: $(i\$*u \$+ m \$< j\$*u \$+ n) <-> ((i\$-j)\$*u \$+ m \$< n)$
 $\langle proof \rangle$

lemma *less-add-iff2*: $(i\$*u \$+ m \$< j\$*u \$+ n) <-> (m \$< (j\$-i)\$*u \$+ n)$
 $\langle proof \rangle$

lemma *le-add-iff1*: $(i\$*u \$+ m \$<= j\$*u \$+ n) <-> ((i\$-j)\$*u \$+ m \$<= n)$
 $\langle proof \rangle$

lemma *le-add-iff2*: $(i\$*u \$+ m \$<= j\$*u \$+ n) <-> (m \$<= (j\$-i)\$*u \$+ n)$
 $\langle proof \rangle$

$\langle ML \rangle$

end

32 IntDiv-ZF: The Division Operators Div and Mod

theory *IntDiv-ZF* **imports** *IntArith OrderArith* **begin**

definition

$quorem :: [i, i] \Rightarrow o$ **where**
 $quorem == \%<a, b> <q, r>.$
 $a = b\$*q \$+ r \ \&$
 $(\#0\$<b \ \& \ \#0\$<=r \ \& \ r\$<b \mid \sim(\#0\$<b) \ \& \ b\$<r \ \& \ r \$<= \#0)$

definition

$adjust :: [i, i] \Rightarrow i$ **where**
 $adjust(b) == \%<q, r>. \text{ if } \#0 \$<= r\$-b \text{ then } <\#2\$*q \$+ \#1, r\$-b>$
 $\text{ else } <\#2\$*q, r>$

definition

$posDivAlg :: i \Rightarrow i$ **where**
 $posDivAlg(ab) ==$
 $wfrec(measure(int*int, \%<a, b>. \text{ nat-of } (a \$- b \$+ \#1)),$
 $ab,$
 $\%<a, b> f. \text{ if } (a\$<b \mid b\$<=\#0) \text{ then } <\#0, a>$
 $\text{ else } adjust(b, f \text{ ‘ } <a, \#2\$*b>))$

definition

$negDivAlg :: i \Rightarrow i$ **where**
 $negDivAlg(ab) ==$
 $wfrec(measure(int*int, \%<a, b>. \text{ nat-of } (\$- a \$- b)),$
 $ab,$
 $\%<a, b> f. \text{ if } (\#0 \$<= a\$+b \mid b\$<=\#0) \text{ then } <\#-1, a\$+b>$
 $\text{ else } adjust(b, f \text{ ‘ } <a, \#2\$*b>))$

definition

$negateSnd :: i \Rightarrow i$ **where**
 $negateSnd == \%<q, r>. <q, \$-r>$

definition

$divAlg :: i \Rightarrow i$ **where**
 $divAlg ==$
 $\%<a, b>. \text{ if } \#0 \$<= a \text{ then}$
 $\text{ if } \#0 \$<= b \text{ then } posDivAlg (<a, b>)$
 $\text{ else if } a=\#0 \text{ then } <\#0, \#0>$
 $\text{ else } negateSnd (negDivAlg (<\$-a, \$-b>))$

else
if #0\$<*b* *then* *negDivAlg* (<*a*,*b*>)
else *negateSnd* (*posDivAlg* (<\$-*a*,*\$-b*>))

definition

zdiv :: [*i*,*i*]==>*i* (infixl *zdiv* 70) **where**
a zdiv b == *fst* (*divAlg* (<*intify*(*a*), *intify*(*b*)>))

definition

zmod :: [*i*,*i*]==>*i* (infixl *zmod* 70) **where**
a zmod b == *snd* (*divAlg* (<*intify*(*a*), *intify*(*b*)>))

lemma *zspos-add-zspos-imp-zspos*: [| #0 \$< *x*; #0 \$< *y* |] ==> #0 \$< *x* \$+ *y*
 <proof>

lemma *zpos-add-zpos-imp-zpos*: [| #0 \$<= *x*; #0 \$<= *y* |] ==> #0 \$<= *x* \$+
y
 <proof>

lemma *zneg-add-zneg-imp-zneg*: [| *x* \$< #0; *y* \$< #0 |] ==> *x* \$+ *y* \$< #0
 <proof>

lemma *zneg-or-0-add-zneg-or-0-imp-zneg-or-0*:
 [| *x* \$<= #0; *y* \$<= #0 |] ==> *x* \$+ *y* \$<= #0
 <proof>

lemma *zero-lt-zmagnitude*: [| #0 \$< *k*; *k* ∈ *int* |] ==> 0 < *zmagnitude*(*k*)
 <proof>

lemma *zless-add-succ-iff*:
 (*w* \$< *z* \$+ \$# *succ*(*m*)) <-> (*w* \$< *z* \$+ \$#*m* | *intify*(*w*) = *z* \$+ \$#*m*)
 <proof>

lemma *zadd-succ-lemma*:
z ∈ *int* ==> (*w* \$+ \$# *succ*(*m*) \$<= *z*) <-> (*w* \$+ \$#*m* \$< *z*)
 <proof>

lemma *zadd-succ-zle-iff*: (*w* \$+ \$# *succ*(*m*) \$<= *z*) <-> (*w* \$+ \$#*m* \$< *z*)
 <proof>

lemma *zless-add1-iff-zle*: $(w \leq z + \#1) \leftrightarrow (w \leq z)$
 $\langle proof \rangle$

lemma *add1-zle-iff*: $(w + \#1 \leq z) \leftrightarrow (w \leq z)$
 $\langle proof \rangle$

lemma *add1-left-zle-iff*: $(\#1 + w \leq z) \leftrightarrow (w \leq z)$
 $\langle proof \rangle$

lemma *zmult-mono-lemma*: $k \in nat \implies i \leq j \implies i * \#k \leq j * \#k$
 $\langle proof \rangle$

lemma *zmult-zle-mono1*: $[i \leq j; \#0 \leq k] \implies i * k \leq j * k$
 $\langle proof \rangle$

lemma *zmult-zle-mono1-neg*: $[i \leq j; k \leq \#0] \implies j * k \leq i * k$
 $\langle proof \rangle$

lemma *zmult-zle-mono2*: $[i \leq j; \#0 \leq k] \implies k * i \leq k * j$
 $\langle proof \rangle$

lemma *zmult-zle-mono2-neg*: $[i \leq j; k \leq \#0] \implies k * j \leq k * i$
 $\langle proof \rangle$

lemma *zmult-zle-mono*:
 $[i \leq j; k \leq l; \#0 \leq j; \#0 \leq k] \implies i * k \leq j * l$
 $\langle proof \rangle$

lemma *zmult-zless-mono2-lemma* [rule-format]:
 $[i < j; k \in nat] \implies 0 < k \longrightarrow \#k * i < \#k * j$
 $\langle proof \rangle$

lemma *zmult-zless-mono2*: $[i < j; \#0 < k] \implies k * i < k * j$
 $\langle proof \rangle$

lemma *zmult-zless-mono1*: $[i < j; \#0 < k] \implies i * k < j * k$
 $\langle proof \rangle$

lemma *zmult-zless-mono*:
 $[i < j; k < l; \#0 < j; \#0 < k] \implies i * k < j * l$
 $\langle proof \rangle$

lemma *zmult-zless-mono1-neg*: $[[i \$< j; k \$< \#0]] ==> j\$*k \$< i\$*k$
 $\langle proof \rangle$

lemma *zmult-zless-mono2-neg*: $[[i \$< j; k \$< \#0]] ==> k\$*j \$< k\$*i$
 $\langle proof \rangle$

lemma *zmult-eq-lemma*:
 $[[m \in int; n \in int]] ==> (m = \#0 \mid n = \#0) <-> (m\$*n = \#0)$
 $\langle proof \rangle$

lemma *zmult-eq-0-iff* [iff]: $(m\$*n = \#0) <-> (intify(m) = \#0 \mid intify(n) = \#0)$
 $\langle proof \rangle$

lemma *zmult-zless-lemma*:
 $[[k \in int; m \in int; n \in int]] ==> (m\$*k \$< n\$*k) <-> ((\#0 \$< k \ \& \ m\$<n) \mid (k \$< \#0 \ \& \ n\$<m))$
 $\langle proof \rangle$

lemma *zmult-zless-cancel2*:
 $(m\$*k \$< n\$*k) <-> ((\#0 \$< k \ \& \ m\$<n) \mid (k \$< \#0 \ \& \ n\$<m))$
 $\langle proof \rangle$

lemma *zmult-zless-cancel1*:
 $(k\$*m \$< k\$*n) <-> ((\#0 \$< k \ \& \ m\$<n) \mid (k \$< \#0 \ \& \ n\$<m))$
 $\langle proof \rangle$

lemma *zmult-zle-cancel2*:
 $(m\$*k \$<= n\$*k) <-> ((\#0 \$< k --> m\$<=n) \ \& \ (k \$< \#0 --> n\$<=m))$
 $\langle proof \rangle$

lemma *zmult-zle-cancel1*:
 $(k\$*m \$<= k\$*n) <-> ((\#0 \$< k --> m\$<=n) \ \& \ (k \$< \#0 --> n\$<=m))$
 $\langle proof \rangle$

lemma *int-eq-iff-zle*: $[[m \in int; n \in int]] ==> m=n <-> (m \$<= n \ \& \ n \$<= m)$
 $\langle proof \rangle$

lemma *zmult-cancel2-lemma*:

$$[[k \in \text{int}; m \in \text{int}; n \in \text{int}]] \implies (m * k = n * k) \leftrightarrow (k = \#0 \mid m = n)$$

 $\langle \text{proof} \rangle$

lemma *zmult-cancel2* [simp]:

$$(m * k = n * k) \leftrightarrow (\text{intify}(k) = \#0 \mid \text{intify}(m) = \text{intify}(n))$$

 $\langle \text{proof} \rangle$

lemma *zmult-cancel1* [simp]:

$$(k * m = k * n) \leftrightarrow (\text{intify}(k) = \#0 \mid \text{intify}(m) = \text{intify}(n))$$

 $\langle \text{proof} \rangle$

32.1 Uniqueness and monotonicity of quotients and remainders

lemma *unique-quotient-lemma*:

$$[[b * q' \$+ r' \$\leq b * q \$+ r; \#0 \$\leq r'; \#0 \$\leq b; r \$\leq b]]$$

$$\implies q' \$\leq q$$

 $\langle \text{proof} \rangle$

lemma *unique-quotient-lemma-neg*:

$$[[b * q' \$+ r' \$\leq b * q \$+ r; r \$\leq \#0; b \$\leq \#0; b \$\leq r']]$$

$$\implies q \$\leq q'$$

 $\langle \text{proof} \rangle$

lemma *unique-quotient*:

$$[[\text{quorem}(<a, b>, <q, r>); \text{quorem}(<a, b>, <q', r'>); b \in \text{int}; b \sim \#0;$$

$$q \in \text{int}; q' \in \text{int}]] \implies q = q'$$

 $\langle \text{proof} \rangle$

lemma *unique-remainder*:

$$[[\text{quorem}(<a, b>, <q, r>); \text{quorem}(<a, b>, <q', r'>); b \in \text{int}; b \sim \#0;$$

$$q \in \text{int}; q' \in \text{int};$$

$$r \in \text{int}; r' \in \text{int}]] \implies r = r'$$

 $\langle \text{proof} \rangle$

32.2 Correctness of posDivAlg, the Division Algorithm for $a \geq 0$ and $b > 0$

lemma *adjust-eq* [simp]:

$$\text{adjust}(b, <q, r>) = (\text{let } \text{diff} = r \$- b \text{ in}$$

$$\text{if } \#0 \$\leq \text{diff} \text{ then } <\#2 * q \$+ \#1, \text{diff}>$$

$$\text{else } <\#2 * q, r>)$$

 $\langle \text{proof} \rangle$

lemma *posDivAlg-termination*:

$$[[\#0 \$\leq b; \sim a \$\leq b]]$$

$$\implies \text{nat-of}(a \$- \#2 \$\times b \$+ \#1) < \text{nat-of}(a \$- b \$+ \#1)$$

$\langle proof \rangle$

lemmas *posDivAlg-unfold* = *def-wfrec* [*OF posDivAlg-def wf-measure*]

lemma *posDivAlg-eqn*:

$[[\#0 \ \$< b; a \in int; b \in int]] ==>$

$posDivAlg(<a,b>) =$

$(if\ a\$<b\ then\ <\#0,a>\ else\ adjust(b,\ posDivAlg\ (<a,\ \#2\$*b>)))$

$\langle proof \rangle$

lemma *posDivAlg-induct-lemma* [*rule-format*]:

assumes *prem*:

$!!a\ b. [[\ a \in int; b \in int;$

$\sim (a\ \$< b \mid b\ \$\leq \#0) \dashrightarrow P(<a,\ \#2\ \$* b>)]] ==> P(<a,b>)$

shows $<u,v> \in int*int \dashrightarrow P(<u,v>)$

$\langle proof \rangle$

lemma *posDivAlg-induct* [*consumes 2*]:

assumes *u-int*: $u \in int$

and *v-int*: $v \in int$

and *ih*: $!!a\ b. [[\ a \in int; b \in int;$

$\sim (a\ \$< b \mid b\ \$\leq \#0) \dashrightarrow P(a,\ \#2\ \$* b)]] ==> P(a,b)$

shows $P(u,v)$

$\langle proof \rangle$

lemma *intify-eq-0-iff-zle*: $intify(m) = \#0 \leftrightarrow (m\ \$\leq \#0 \ \& \ \#0\ \$\leq m)$

$\langle proof \rangle$

32.3 Some convenient biconditionals for products of signs

lemma *zmult-pos*: $[[\#0 \ \$< i; \#0 \ \$< j]] ==> \#0 \ \$< i\ \$* j$

$\langle proof \rangle$

lemma *zmult-neg*: $[[i\ \$< \#0; j\ \$< \#0]] ==> \#0 \ \$< i\ \$* j$

$\langle proof \rangle$

lemma *zmult-pos-neg*: $[[\#0 \ \$< i; j\ \$< \#0]] ==> i\ \$* j\ \$< \#0$

$\langle proof \rangle$

lemma *int-0-less-lemma*:

$[[x \in int; y \in int]]$

$==> (\#0 \ \$< x\ \$* y) \leftrightarrow (\#0 \ \$< x \ \& \ \#0 \ \$< y \mid x\ \$< \#0 \ \& \ y\ \$< \#0)$

$\langle proof \rangle$

lemma *int-0-less-mult-iff*:

$(\#0 \ \$< x \ \$* y) <-> (\#0 \ \$< x \ \& \ \#0 \ \$< y \mid x \ \$< \#0 \ \& \ y \ \$< \#0)$
 $\langle proof \rangle$

lemma *int-0-le-lemma*:

$[\mid x \in int; y \in int \mid]$
 $==> (\#0 \ \$<= x \ \$* y) <-> (\#0 \ \$<= x \ \& \ \#0 \ \$<= y \mid x \ \$<= \#0 \ \& \ y \ \$<= \#0)$
 $\langle proof \rangle$

lemma *int-0-le-mult-iff*:

$(\#0 \ \$<= x \ \$* y) <-> ((\#0 \ \$<= x \ \& \ \#0 \ \$<= y) \mid (x \ \$<= \#0 \ \& \ y \ \$<= \#0))$
 $\langle proof \rangle$

lemma *zmult-less-0-iff*:

$(x \ \$* y \ \$< \#0) <-> (\#0 \ \$< x \ \& \ y \ \$< \#0 \mid x \ \$< \#0 \ \& \ \#0 \ \$< y)$
 $\langle proof \rangle$

lemma *zmult-le-0-iff*:

$(x \ \$* y \ \$<= \#0) <-> (\#0 \ \$<= x \ \& \ y \ \$<= \#0 \mid x \ \$<= \#0 \ \& \ \#0 \ \$<= y)$
 $\langle proof \rangle$

lemma *posDivAlg-type* [rule-format]:

$[\mid a \in int; b \in int \mid] ==> posDivAlg(<a,b>) \in int * int$
 $\langle proof \rangle$

lemma *posDivAlg-correct* [rule-format]:

$[\mid a \in int; b \in int \mid]$
 $==> \#0 \ \$<= a \ ---> \#0 \ \$< b \ ---> quorem (<a,b>, posDivAlg(<a,b>))$
 $\langle proof \rangle$

32.4 Correctness of negDivAlg, the division algorithm for $a \neq 0$ and $b \neq 0$

lemma *negDivAlg-termination*:

$[\mid \#0 \ \$< b; a \ \$+ b \ \$< \#0 \mid]$
 $==> nat-of(\$- a \ \$- \#2 \ \$* b) < nat-of(\$- a \ \$- b)$
 $\langle proof \rangle$

lemmas *negDivAlg-unfold* = def-wfrec [OF negDivAlg-def wf-measure]

lemma *negDivAlg-eqn*:

$[\mid \#0 \ \$< b; a : int; b : int \mid] ==>$
 $negDivAlg(<a,b>) =$
 $(if \ \#0 \ \$<= a \$+ b \ then \ <\#-1, a \$+ b>$
 $\quad \text{else } adjust(b, negDivAlg (<a, \#2 \$* b>)))$

$\langle proof \rangle$

lemma *negDivAlg-induct-lemma* [rule-format]:

assumes *prem*:

!!*a b*. [| *a* ∈ *int*; *b* ∈ *int*;
 $\sim (\#0 \ \$\leq a \ \$+ b \mid b \ \$\leq \#0) \dashrightarrow P(<a, \#2 \ \$* b>)$ |]
 $\implies P(<a, b>)$

shows $<u, v> \in int * int \dashrightarrow P(<u, v>)$

$\langle proof \rangle$

lemma *negDivAlg-induct* [consumes 2]:

assumes *u-int*: *u* ∈ *int*

and *v-int*: *v* ∈ *int*

and *ih*: !!*a b*. [| *a* ∈ *int*; *b* ∈ *int*;
 $\sim (\#0 \ \$\leq a \ \$+ b \mid b \ \$\leq \#0) \dashrightarrow P(a, \#2 \ \$* b)$ |]
 $\implies P(a, b)$

shows $P(u, v)$

$\langle proof \rangle$

lemma *negDivAlg-type*:

[| *a* ∈ *int*; *b* ∈ *int* |] $\implies negDivAlg(<a, b>) \in int * int$

$\langle proof \rangle$

lemma *negDivAlg-correct* [rule-format]:

[| *a* ∈ *int*; *b* ∈ *int* |]
 $\implies a \ \$< \#0 \dashrightarrow \#0 \ \$< b \dashrightarrow quorem(<a, b>, negDivAlg(<a, b>))$

$\langle proof \rangle$

32.5 Existence shown by proving the division algorithm to be correct

lemma *quorem-0*: [| *b* ≠ #0; *b* ∈ *int* |] $\implies quorem(<\#0, b>, <\#0, \#0>)$

$\langle proof \rangle$

lemma *posDivAlg-zero-divisor*: $posDivAlg(<a, \#0>) = <\#0, a>$

$\langle proof \rangle$

lemma *posDivAlg-0* [simp]: $posDivAlg(<\#0, b>) = <\#0, \#0>$

$\langle proof \rangle$

lemma *linear-arith-lemma*: $\sim (\#0 \ \$\leq \#-1 \ \$+ b) \implies (b \ \$\leq \#0)$

$\langle proof \rangle$

lemma *negDivAlg-minus1* [simp]: $\text{negDivAlg } (<\#-1, b>) = <\#-1, b\$-\#1>$
 $\langle \text{proof} \rangle$

lemma *negateSnd-eq* [simp]: $\text{negateSnd } (<q, r>) = <q, \$-r>$
 $\langle \text{proof} \rangle$

lemma *negateSnd-type*: $qr \in \text{int} * \text{int} \implies \text{negateSnd } (qr) \in \text{int} * \text{int}$
 $\langle \text{proof} \rangle$

lemma *quorem-neg*:
 $[[\text{quorem } (<\$-a, \$-b>, qr); a \in \text{int}; b \in \text{int}; qr \in \text{int} * \text{int}]]$
 $\implies \text{quorem } (<a, b>, \text{negateSnd}(qr))$
 $\langle \text{proof} \rangle$

lemma *divAlg-correct*:
 $[[b \neq \#0; a \in \text{int}; b \in \text{int}]] \implies \text{quorem } (<a, b>, \text{divAlg } (<a, b>))$
 $\langle \text{proof} \rangle$

lemma *divAlg-type*: $[[a \in \text{int}; b \in \text{int}]] \implies \text{divAlg } (<a, b>) \in \text{int} * \text{int}$
 $\langle \text{proof} \rangle$

lemma *zdiv-intify1* [simp]: $\text{intify}(x) \text{ zdiv } y = x \text{ zdiv } y$
 $\langle \text{proof} \rangle$

lemma *zdiv-intify2* [simp]: $x \text{ zdiv } \text{intify}(y) = x \text{ zdiv } y$
 $\langle \text{proof} \rangle$

lemma *zdiv-type* [iff, TC]: $z \text{ zdiv } w \in \text{int}$
 $\langle \text{proof} \rangle$

lemma *zmod-intify1* [simp]: $\text{intify}(x) \text{ zmod } y = x \text{ zmod } y$
 $\langle \text{proof} \rangle$

lemma *zmod-intify2* [simp]: $x \text{ zmod } \text{intify}(y) = x \text{ zmod } y$
 $\langle \text{proof} \rangle$

lemma *zmod-type* [iff, TC]: $z \text{ zmod } w \in \text{int}$
 $\langle \text{proof} \rangle$

lemma *DIVISION-BY-ZERO-ZDIV*: $a \text{ zdiv } \#0 = \#0$
 $\langle \text{proof} \rangle$

lemma *DIVISION-BY-ZERO-ZMOD*: $a \text{ zmod } \#0 = \text{intify}(a)$

$\langle proof \rangle$

lemma *raw-zmod-zdiv-equality*:

$[[a \in int; b \in int]] \implies a = b \$* (a \text{ zdiv } b) \$+ (a \text{ zmod } b)$
 $\langle proof \rangle$

lemma *zmod-zdiv-equality*: $intify(a) = b \$* (a \text{ zdiv } b) \$+ (a \text{ zmod } b)$
 $\langle proof \rangle$

lemma *pos-mod*: $\#0 \$< b \implies \#0 \$<= a \text{ zmod } b \ \& \ a \text{ zmod } b \$< b$
 $\langle proof \rangle$

lemmas *pos-mod-sign* = *pos-mod* [*THEN* *conjunct1*, *standard*]
and *pos-mod-bound* = *pos-mod* [*THEN* *conjunct2*, *standard*]

lemma *neg-mod*: $b \$< \#0 \implies a \text{ zmod } b \$<= \#0 \ \& \ b \$< a \text{ zmod } b$
 $\langle proof \rangle$

lemmas *neg-mod-sign* = *neg-mod* [*THEN* *conjunct1*, *standard*]
and *neg-mod-bound* = *neg-mod* [*THEN* *conjunct2*, *standard*]

lemma *quorem-div-mod*:

$[[b \neq \#0; a \in int; b \in int]]$
 $\implies quorem (<a,b>, <a \text{ zdiv } b, a \text{ zmod } b>)$
 $\langle proof \rangle$

lemma *quorem-div*:

$[[quorem (<a,b>, <q,r>); b \neq \#0; a \in int; b \in int; q \in int]]$
 $\implies a \text{ zdiv } b = q$
 $\langle proof \rangle$

lemma *quorem-mod*:

$[[quorem (<a,b>, <q,r>); b \neq \#0; a \in int; b \in int; q \in int; r \in int]]$
 $\implies a \text{ zmod } b = r$
 $\langle proof \rangle$

lemma *zdiv-pos-pos-trivial-raw*:

$[[a \in int; b \in int; \#0 \$<= a; a \$< b]]$ $\implies a \text{ zdiv } b = \#0$
 $\langle proof \rangle$

lemma *zdiv-pos-pos-trivial*: $[[\#0 \$<= a; a \$< b]]$ $\implies a \text{ zdiv } b = \#0$

$\langle proof \rangle$

lemma *zdiv-neg-neg-trivial-raw*:

$\llbracket a \in int; b \in int; a \leq 0; b < a \rrbracket \implies a \text{ zdiv } b = 0$
 $\langle proof \rangle$

lemma *zdiv-neg-neg-trivial*: $\llbracket a \leq 0; b < a \rrbracket \implies a \text{ zdiv } b = 0$

$\langle proof \rangle$

lemma *zadd-le-0-lemma*: $\llbracket a+b \leq 0; 0 < a; 0 < b \rrbracket \implies False$

$\langle proof \rangle$

lemma *zdiv-pos-neg-trivial-raw*:

$\llbracket a \in int; b \in int; 0 < a; a+b \leq 0 \rrbracket \implies a \text{ zdiv } b = -1$
 $\langle proof \rangle$

lemma *zdiv-pos-neg-trivial*: $\llbracket 0 < a; a+b \leq 0 \rrbracket \implies a \text{ zdiv } b = -1$

$\langle proof \rangle$

lemma *zmod-pos-pos-trivial-raw*:

$\llbracket a \in int; b \in int; 0 \leq a; a < b \rrbracket \implies a \text{ zmod } b = a$
 $\langle proof \rangle$

lemma *zmod-pos-pos-trivial*: $\llbracket 0 \leq a; a < b \rrbracket \implies a \text{ zmod } b = \text{intify}(a)$

$\langle proof \rangle$

lemma *zmod-neg-neg-trivial-raw*:

$\llbracket a \in int; b \in int; a \leq 0; b < a \rrbracket \implies a \text{ zmod } b = a$
 $\langle proof \rangle$

lemma *zmod-neg-neg-trivial*: $\llbracket a \leq 0; b < a \rrbracket \implies a \text{ zmod } b = \text{intify}(a)$

$\langle proof \rangle$

lemma *zmod-pos-neg-trivial-raw*:

$\llbracket a \in int; b \in int; 0 < a; a+b \leq 0 \rrbracket \implies a \text{ zmod } b = a+b$
 $\langle proof \rangle$

lemma *zmod-pos-neg-trivial*: $\llbracket 0 < a; a+b \leq 0 \rrbracket \implies a \text{ zmod } b = a+b$

$\langle proof \rangle$

lemma *zdiv-zminus-zminus-raw*:

$[|a \in \text{int}; b \in \text{int}|] \implies (\$-a) \text{ zdiv } (\$-b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zminus-zminus [simp]*: $(\$-a) \text{ zdiv } (\$-b) = a \text{ zdiv } b$

$\langle \text{proof} \rangle$

lemma *zmod-zminus-zminus-raw*:

$[|a \in \text{int}; b \in \text{int}|] \implies (\$-a) \text{ zmod } (\$-b) = \$- (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zminus-zminus [simp]*: $(\$-a) \text{ zmod } (\$-b) = \$- (a \text{ zmod } b)$

$\langle \text{proof} \rangle$

32.6 division of a number by itself

lemma *self-quotient-aux1*: $[| \#0 \$< a; a = r \$+ a\$*q; r \$< a |] \implies \#1 \$<= q$
 $\langle \text{proof} \rangle$

lemma *self-quotient-aux2*: $[| \#0 \$< a; a = r \$+ a\$*q; \#0 \$<= r |] \implies q \$<= \#1$
 $\langle \text{proof} \rangle$

lemma *self-quotient*:

$[| \text{quorem}(<a,a>,<q,r>); a \in \text{int}; q \in \text{int}; a \neq \#0 |] \implies q = \#1$
 $\langle \text{proof} \rangle$

lemma *self-remainder*:

$[| \text{quorem}(<a,a>,<q,r>); a \in \text{int}; q \in \text{int}; r \in \text{int}; a \neq \#0 |] \implies r = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-self-raw*: $[|a \neq \#0; a \in \text{int}|] \implies a \text{ zdiv } a = \#1$
 $\langle \text{proof} \rangle$

lemma *zdiv-self [simp]*: $\text{intify}(a) \neq \#0 \implies a \text{ zdiv } a = \#1$
 $\langle \text{proof} \rangle$

lemma *zmod-self-raw*: $a \in \text{int} \implies a \text{ zmod } a = \#0$
 $\langle \text{proof} \rangle$

lemma *zmod-self [simp]*: $a \text{ zmod } a = \#0$
 $\langle \text{proof} \rangle$

32.7 Computation of division and remainder

lemma *zdiv-zero [simp]*: $\#0 \text{ zdiv } b = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-eq-minus1*: $\#0 \ \$< \ b \implies \#-1 \ zdiv \ b = \#-1$
 $\langle proof \rangle$

lemma *zmod-zero* [*simp*]: $\#0 \ zmod \ b = \#0$
 $\langle proof \rangle$

lemma *zdiv-minus1*: $\#0 \ \$< \ b \implies \#-1 \ zdiv \ b = \#-1$
 $\langle proof \rangle$

lemma *zmod-minus1*: $\#0 \ \$< \ b \implies \#-1 \ zmod \ b = b \ \$- \ #1$
 $\langle proof \rangle$

lemma *zdiv-pos-pos*: $[\#0 \ \$< \ a; \ #0 \ \$\leq \ b]$
 $\implies a \ zdiv \ b = fst \ (posDivAlg(<intify(a), \ intify(b)>))$
 $\langle proof \rangle$

lemma *zmod-pos-pos*:
 $[\#0 \ \$< \ a; \ #0 \ \$\leq \ b]$
 $\implies a \ zmod \ b = snd \ (posDivAlg(<intify(a), \ intify(b)>))$
 $\langle proof \rangle$

lemma *zdiv-neg-pos*:
 $[a \ \$< \ \#0; \ \#0 \ \$< \ b]$
 $\implies a \ zdiv \ b = fst \ (negDivAlg(<intify(a), \ intify(b)>))$
 $\langle proof \rangle$

lemma *zmod-neg-pos*:
 $[a \ \$< \ \#0; \ \#0 \ \$< \ b]$
 $\implies a \ zmod \ b = snd \ (negDivAlg(<intify(a), \ intify(b)>))$
 $\langle proof \rangle$

lemma *zdiv-pos-neg*:
 $[\#0 \ \$< \ a; \ b \ \$< \ \#0]$
 $\implies a \ zdiv \ b = fst \ (negateSnd(negDivAlg \ (<\$-a, \ \$-b>)))$
 $\langle proof \rangle$

lemma *zmod-pos-neg*:
 $[\#0 \ \$< \ a; \ b \ \$< \ \#0]$
 $\implies a \ zmod \ b = snd \ (negateSnd(negDivAlg \ (<\$-a, \ \$-b>)))$
 $\langle proof \rangle$

lemma *zdiv-neg-neg*:

$$[| a \$ < \#0; b \$ \leq \#0 |] \implies a \text{ zdiv } b = \text{fst } (\text{negateSnd}(\text{posDivAlg}(<\$-a, \$-b>)))$$
 $\langle \text{proof} \rangle$

lemma *zmod-neg-neg*:

$$[| a \$ < \#0; b \$ \leq \#0 |] \implies a \text{ zmod } b = \text{snd } (\text{negateSnd}(\text{posDivAlg}(<\$-a, \$-b>)))$$
 $\langle \text{proof} \rangle$

declare *zdiv-pos-pos* [of integ-of (v) integ-of (w), standard, simp]
declare *zdiv-neg-pos* [of integ-of (v) integ-of (w), standard, simp]
declare *zdiv-pos-neg* [of integ-of (v) integ-of (w), standard, simp]
declare *zdiv-neg-neg* [of integ-of (v) integ-of (w), standard, simp]
declare *zmod-pos-pos* [of integ-of (v) integ-of (w), standard, simp]
declare *zmod-neg-pos* [of integ-of (v) integ-of (w), standard, simp]
declare *zmod-pos-neg* [of integ-of (v) integ-of (w), standard, simp]
declare *zmod-neg-neg* [of integ-of (v) integ-of (w), standard, simp]
declare *posDivAlg-eqn* [of concl: integ-of (v) integ-of (w), standard, simp]
declare *negDivAlg-eqn* [of concl: integ-of (v) integ-of (w), standard, simp]

lemma *zmod-1* [simp]: $a \text{ zmod } \#1 = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-1* [simp]: $a \text{ zdiv } \#1 = \text{intify}(a)$
 $\langle \text{proof} \rangle$

lemma *zmod-minus1-right* [simp]: $a \text{ zmod } \#-1 = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-minus1-right-raw*: $a \in \text{int} \implies a \text{ zdiv } \#-1 = \$-a$
 $\langle \text{proof} \rangle$

lemma *zdiv-minus1-right*: $a \text{ zdiv } \#-1 = \$-a$
 $\langle \text{proof} \rangle$

declare *zdiv-minus1-right* [simp]

32.8 Monotonicity in the first argument (divisor)

lemma *zdiv-mono1*: $[| a \$ \leq a'; \#0 \$ < b |] \implies a \text{ zdiv } b \$ \leq a' \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-mono1-neg*: $[| a \$ \leq a'; b \$ < \#0 |] \implies a' \text{ zdiv } b \$ \leq a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

32.9 Monotonicity in the second argument (dividend)

lemma *q-pos-lemma*:

$[[\#0 \leq b' * q' + r'; r' < b'; \#0 < b']] \implies \#0 \leq q'$
 $\langle proof \rangle$

lemma *zdiv-mono2-lemma*:

$[[b * q + r = b' * q' + r'; \#0 \leq b' * q' + r';$
 $r' < b'; \#0 \leq r; \#0 < b'; b' \leq b]]$
 $\implies q \leq q'$
 $\langle proof \rangle$

lemma *zdiv-mono2-raw*:

$[[\#0 \leq a; \#0 < b'; b' \leq b; a \in int]]$
 $\implies a \text{ zdiv } b \leq a \text{ zdiv } b'$
 $\langle proof \rangle$

lemma *zdiv-mono2*:

$[[\#0 \leq a; \#0 < b'; b' \leq b]]$
 $\implies a \text{ zdiv } b \leq a \text{ zdiv } b'$
 $\langle proof \rangle$

lemma *q-neg-lemma*:

$[[b' * q' + r' < \#0; \#0 \leq r'; \#0 < b']] \implies q' < \#0$
 $\langle proof \rangle$

lemma *zdiv-mono2-neg-lemma*:

$[[b * q + r = b' * q' + r'; b' * q' + r' < \#0;$
 $r < b; \#0 \leq r'; \#0 < b'; b' \leq b]]$
 $\implies q' \leq q$
 $\langle proof \rangle$

lemma *zdiv-mono2-neg-raw*:

$[[a < \#0; \#0 < b'; b' \leq b; a \in int]]$
 $\implies a \text{ zdiv } b' \leq a \text{ zdiv } b$
 $\langle proof \rangle$

lemma *zdiv-mono2-neg*: $[[a < \#0; \#0 < b'; b' \leq b]]$

$\implies a \text{ zdiv } b' \leq a \text{ zdiv } b$
 $\langle proof \rangle$

32.10 More algebraic laws for zdiv and zmod

lemma *zmult1-lemma*:

$[[\text{quorem}(\langle b, c \rangle, \langle q, r \rangle); c \in int; c \neq \#0]]$
 $\implies \text{quorem}(\langle a * b, c \rangle, \langle a * q + (a * r) \text{ zdiv } c, (a * r) \text{ zmod } c \rangle)$
 $\langle proof \rangle$

lemma *zdiv-zmult1-eq-raw*:

$[[b \in \text{int}; c \in \text{int}]]$
 $\implies (a\$*b) \text{ zdiv } c = a\$*(b \text{ zdiv } c) \$+ a\$*(b \text{ zmod } c) \text{ zdiv } c$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult1-eq*: $(a\$*b) \text{ zdiv } c = a\$*(b \text{ zdiv } c) \$+ a\$*(b \text{ zmod } c) \text{ zdiv } c$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult1-eq-raw*:

$[[b \in \text{int}; c \in \text{int}]] \implies (a\$*b) \text{ zmod } c = a\$*(b \text{ zmod } c) \text{ zmod } c$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult1-eq*: $(a\$*b) \text{ zmod } c = a\$*(b \text{ zmod } c) \text{ zmod } c$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult1-eq'*: $(a\$*b) \text{ zmod } c = ((a \text{ zmod } c) \$* b) \text{ zmod } c$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-distrib*: $(a\$*b) \text{ zmod } c = ((a \text{ zmod } c) \$* (b \text{ zmod } c)) \text{ zmod } c$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-self1* [simp]: $\text{intify}(b) \neq \#0 \implies (a\$*b) \text{ zdiv } b = \text{intify}(a)$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-self2* [simp]: $\text{intify}(b) \neq \#0 \implies (b\$*a) \text{ zdiv } b = \text{intify}(a)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-self1* [simp]: $(a\$*b) \text{ zmod } b = \#0$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-self2* [simp]: $(b\$*a) \text{ zmod } b = \#0$
 $\langle \text{proof} \rangle$

lemma *zadd1-lemma*:

$[[\text{quorem}(\langle a, c \rangle, \langle aq, ar \rangle); \text{quorem}(\langle b, c \rangle, \langle bq, br \rangle);$
 $c \in \text{int}; c \neq \#0]]$
 $\implies \text{quorem}(\langle a\$+b, c \rangle, \langle aq \$+ bq \$+ (ar\$+br) \text{ zdiv } c, (ar\$+br) \text{ zmod } c \rangle)$
 $\langle \text{proof} \rangle$

lemma *zdiv-zadd1-eq-raw*:

$[[a \in \text{int}; b \in \text{int}; c \in \text{int}]] \implies$
 $(a\$+b) \text{ zdiv } c = a \text{ zdiv } c \$+ b \text{ zdiv } c \$+ ((a \text{ zmod } c \$+ b \text{ zmod } c) \text{ zdiv } c)$
 $\langle \text{proof} \rangle$

lemma *zdiv-zadd1-eq*:

$$(a\$+b) \text{ zdiv } c = a \text{ zdiv } c \$+ b \text{ zdiv } c \$+ ((a \text{ zmod } c \$+ b \text{ zmod } c) \text{ zdiv } c)$$

<proof>

lemma *zmod-zadd1-eq-raw*:

$$[[a \in \text{int}; b \in \text{int}; c \in \text{int}]] \\ \implies (a\$+b) \text{ zmod } c = (a \text{ zmod } c \$+ b \text{ zmod } c) \text{ zmod } c$$

<proof>

lemma *zmod-zadd1-eq*: $(a\$+b) \text{ zmod } c = (a \text{ zmod } c \$+ b \text{ zmod } c) \text{ zmod } c$
<proof>

lemma *zmod-div-trivial-raw*:

$$[[a \in \text{int}; b \in \text{int}]] \implies (a \text{ zmod } b) \text{ zdiv } b = \#0$$

<proof>

lemma *zmod-div-trivial* [simp]: $(a \text{ zmod } b) \text{ zdiv } b = \#0$
<proof>

lemma *zmod-mod-trivial-raw*:

$$[[a \in \text{int}; b \in \text{int}]] \implies (a \text{ zmod } b) \text{ zmod } b = a \text{ zmod } b$$

<proof>

lemma *zmod-mod-trivial* [simp]: $(a \text{ zmod } b) \text{ zmod } b = a \text{ zmod } b$
<proof>

lemma *zmod-zadd-left-eq*: $(a\$+b) \text{ zmod } c = ((a \text{ zmod } c) \$+ b) \text{ zmod } c$
<proof>

lemma *zmod-zadd-right-eq*: $(a\$+b) \text{ zmod } c = (a \$+ (b \text{ zmod } c)) \text{ zmod } c$
<proof>

lemma *zdiv-zadd-self1* [simp]:

$$\text{intify}(a) \neq \#0 \implies (a\$+b) \text{ zdiv } a = b \text{ zdiv } a \$+ \#1$$

<proof>

lemma *zdiv-zadd-self2* [simp]:

$$\text{intify}(a) \neq \#0 \implies (b\$+a) \text{ zdiv } a = b \text{ zdiv } a \$+ \#1$$

<proof>

lemma *zmod-zadd-self1* [simp]: $(a\$+b) \text{ zmod } a = b \text{ zmod } a$
<proof>

lemma *zmod-zadd-self2* [simp]: $(b\$+a) \text{ zmod } a = b \text{ zmod } a$
<proof>

32.11 proving a zdiv (b*c) = (a zdiv b) zdiv c

lemma *zdiv-zmult2-aux1*:

$[| \#0 \$< c; \ b \$< r; \ r \$\leq \#0 \ |] \implies b\$*c \$< b\$*(q \text{ zmod } c) \$+ r$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-aux2*:

$[| \#0 \$< c; \ \ b \$< r; \ r \$\leq \#0 \ |] \implies b \$* (q \text{ zmod } c) \$+ r \$\leq \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-aux3*:

$[| \#0 \$< c; \ \#0 \$\leq r; \ r \$< b \ |] \implies \#0 \$\leq b \$* (q \text{ zmod } c) \$+ r$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-aux4*:

$[| \#0 \$< c; \ \#0 \$\leq r; \ r \$< b \ |] \implies b \$* (q \text{ zmod } c) \$+ r \$< b \$* c$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-lemma*:

$[| \text{quorem } (<a,b>, <q,r>); \ a \in \text{int}; \ b \in \text{int}; \ b \neq \#0; \ \#0 \$< c \ |]$
 $\implies \text{quorem } (<a,b\$*c>, <q \text{ zdiv } c, b\$*(q \text{ zmod } c) \$+ r>)$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-eq-row*:

$[| \#0 \$< c; \ a \in \text{int}; \ b \in \text{int} \ |] \implies a \text{ zdiv } (b\$*c) = (a \text{ zdiv } b) \text{ zdiv } c$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-eq*: $\#0 \$< c \implies a \text{ zdiv } (b\$*c) = (a \text{ zdiv } b) \text{ zdiv } c$

$\langle \text{proof} \rangle$

lemma *zmod-zmult2-eq-row*:

$[| \#0 \$< c; \ a \in \text{int}; \ b \in \text{int} \ |]$
 $\implies a \text{ zmod } (b\$*c) = b\$*(a \text{ zdiv } b \text{ zmod } c) \$+ a \text{ zmod } b$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult2-eq*:

$\#0 \$< c \implies a \text{ zmod } (b\$*c) = b\$*(a \text{ zdiv } b \text{ zmod } c) \$+ a \text{ zmod } b$
 $\langle \text{proof} \rangle$

32.12 Cancellation of common factors in "zdiv"

lemma *zdiv-zmult-zmult1-aux1*:

$[| \#0 \$< b; \ \text{intify}(c) \neq \#0 \ |] \implies (c\$*a) \text{ zdiv } (c\$*b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult1-aux2*:

$[| b \$< \#0; \ \text{intify}(c) \neq \#0 \ |] \implies (c\$*a) \text{ zdiv } (c\$*b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult1-row*:

$$[[\text{intify}(c) \neq \#0; b \in \text{int}]] \implies (c\$*a) \text{ zdiv } (c\$*b) = a \text{ zdiv } b$$

 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult1*: $\text{intify}(c) \neq \#0 \implies (c\$*a) \text{ zdiv } (c\$*b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult2*: $\text{intify}(c) \neq \#0 \implies (a\$*c) \text{ zdiv } (b\$*c) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

32.13 Distribution of factors over "zmod"

lemma *zmod-zmult-zmult1-aux1*:

$$[[\#0 \$< b; \text{intify}(c) \neq \#0]] \implies (c\$*a) \text{ zmod } (c\$*b) = c \$* (a \text{ zmod } b)$$

 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult1-aux2*:

$$[[b \$< \#0; \text{intify}(c) \neq \#0]] \implies (c\$*a) \text{ zmod } (c\$*b) = c \$* (a \text{ zmod } b)$$

 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult1-raw*:

$$[[b \in \text{int}; c \in \text{int}]] \implies (c\$*a) \text{ zmod } (c\$*b) = c \$* (a \text{ zmod } b)$$

 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult1*: $(c\$*a) \text{ zmod } (c\$*b) = c \$* (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult2*: $(a\$*c) \text{ zmod } (b\$*c) = (a \text{ zmod } b) \$* c$
 $\langle \text{proof} \rangle$

lemma *zdiv-neg-pos-less0*: $[[a \$< \#0; \#0 \$< b]] \implies a \text{ zdiv } b \$< \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-nonneg-neg-le0*: $[[\#0 \$<= a; b \$< \#0]] \implies a \text{ zdiv } b \$<= \#0$
 $\langle \text{proof} \rangle$

lemma *pos-imp-zdiv-nonneg-iff*: $\#0 \$< b \implies (\#0 \$<= a \text{ zdiv } b) <-> (\#0 \$<= a)$
 $\langle \text{proof} \rangle$

lemma *neg-imp-zdiv-nonneg-iff*: $b \$< \#0 \implies (\#0 \$<= a \text{ zdiv } b) <-> (a \$<= \#0)$
 $\langle \text{proof} \rangle$

lemma *pos-imp-zdiv-neg-iff*: $\#0 \leq b \implies (a \text{ zdiv } b \leq \#0) \iff (a \leq \#0)$
 $\langle \text{proof} \rangle$

lemma *neg-imp-zdiv-neg-iff*: $b \leq \#0 \implies (a \text{ zdiv } b \leq \#0) \iff (\#0 \leq a)$
 $\langle \text{proof} \rangle$

end

33 CardinalArith: Cardinal Arithmetic Without the Axiom of Choice

theory *CardinalArith* **imports** *Cardinal OrderArith ArithSimp Finite* **begin**

definition

InfCard $:: i \Rightarrow o$ **where**
InfCard(*i*) == *Card*(*i*) & nat le *i*

definition

cmult $:: [i, i] \Rightarrow i$ (**infixl** $|*|$ 70) **where**
 $i \text{ } |*| \text{ } j == |i*j|$

definition

cadd $:: [i, i] \Rightarrow i$ (**infixl** $|+|$ 65) **where**
 $i \text{ } |+| \text{ } j == |i+j|$

definition

csquare-rel $:: i \Rightarrow i$ **where**
csquare-rel(*K*) ==
 $\text{rvimage}(K*K,$
 $\text{lam } \langle x, y \rangle : K*K. \langle x \text{ Un } y, x, y \rangle,$
 $\text{rmult}(K, \text{Memrel}(K), K*K, \text{rmult}(K, \text{Memrel}(K), K, \text{Memrel}(K))))$

definition

jump-cardinal $:: i \Rightarrow i$ **where**
— This def is more complex than Kunen’s but it more easily proved to be a cardinal
jump-cardinal(*K*) ==
 $\bigcup X \in \text{Pow}(K). \{z. r: \text{Pow}(K*K), \text{well-ord}(X, r) \ \& \ z = \text{ordertype}(X, r)\}$

definition

csucc $:: i \Rightarrow i$ **where**
— needed because *jump-cardinal*(*K*) might not be the successor of *K*
csucc(*K*) == *LEAST* *L*. *Card*(*L*) & *K* < *L*

notation (*xsymbols output*)

cadd (**infixl** \oplus 65) **and**

cmult (**infixl** \otimes 70)

notation (*HTML output*)

cadd (**infixl** \oplus 65) **and**

cmult (**infixl** \otimes 70)

lemma *Card-Union* [*simp,intro,TC*]: $(\text{ALL } x:A. \text{Card}(x)) \implies \text{Card}(\text{Union}(A))$
<proof>

lemma *Card-UN*: $(!\!x. x:A \implies \text{Card}(K(x))) \implies \text{Card}(\bigcup_{x \in A} K(x))$
<proof>

lemma *Card-OUN* [*simp,intro,TC*]:
 $(!\!x. x:A \implies \text{Card}(K(x))) \implies \text{Card}(\bigcup_{x < A} K(x))$
<proof>

lemma *n-lesspoll-nat*: $n \in \text{nat} \implies n \prec \text{nat}$
<proof>

lemma *in-Card-imp-lesspoll*: $[\mid \text{Card}(K); b \in K \mid] \implies b \prec K$
<proof>

lemma *lesspoll-lemma*: $[\mid \sim A \prec B; C \prec B \mid] \implies A - C \neq 0$
<proof>

33.1 Cardinal addition

Note: Could omit proving the algebraic laws for cardinal addition and multiplication. On finite cardinals these operations coincide with addition and multiplication of natural numbers; on infinite cardinals they coincide with union (maximum). Either way we get most laws for free.

33.1.1 Cardinal addition is commutative

lemma *sum-commute-epoll*: $A+B \approx B+A$
<proof>

lemma *cadd-commute*: $i \mid + \mid j = j \mid + \mid i$
<proof>

33.1.2 Cardinal addition is associative

lemma *sum-assoc-epoll*: $(A+B)+C \approx A+(B+C)$
<proof>

lemma *well-ord-cadd-assoc*:

$$[\text{well-ord}(i, ri); \text{well-ord}(j, rj); \text{well-ord}(k, rk)]$$

$$\implies (i \mid\mid j) \mid\mid k = i \mid\mid (j \mid\mid k)$$

$$\langle \text{proof} \rangle$$

33.1.3 0 is the identity for addition

lemma *sum-0-epoll*: $0 + A \approx A$

$\langle \text{proof} \rangle$

lemma *cadd-0 [simp]*: $\text{Card}(K) \implies 0 \mid\mid K = K$

$\langle \text{proof} \rangle$

33.1.4 Addition by another cardinal

lemma *sum-lepoll-self*: $A \lesssim A + B$

$\langle \text{proof} \rangle$

lemma *cadd-le-self*:

$$[\text{Card}(K); \text{Ord}(L)] \implies K \text{ le } (K \mid\mid L)$$

$$\langle \text{proof} \rangle$$

33.1.5 Monotonicity of addition

lemma *sum-lepoll-mono*:

$$[A \lesssim C; B \lesssim D] \implies A + B \lesssim C + D$$

$$\langle \text{proof} \rangle$$

lemma *cadd-le-mono*:

$$[K' \text{ le } K; L' \text{ le } L] \implies (K' \mid\mid L') \text{ le } (K \mid\mid L)$$

$$\langle \text{proof} \rangle$$

33.1.6 Addition of finite cardinals is "ordinary" addition

lemma *sum-succ-epoll*: $\text{succ}(A) + B \approx \text{succ}(A + B)$

$\langle \text{proof} \rangle$

lemma *cadd-succ-lemma*:

$$[\text{Ord}(m); \text{Ord}(n)] \implies \text{succ}(m) \mid\mid n = |\text{succ}(m \mid\mid n)|$$

$$\langle \text{proof} \rangle$$

lemma *nat-cadd-eq-add*: $[m: \text{nat}; n: \text{nat}] \implies m \mid\mid n = m \# + n$

$\langle \text{proof} \rangle$

33.2 Cardinal multiplication

33.2.1 Cardinal multiplication is commutative

lemma *prod-commute-epoll*: $A * B \approx B * A$
<proof>

lemma *cmult-commute*: $i \mid * \mid j = j \mid * \mid i$
<proof>

33.2.2 Cardinal multiplication is associative

lemma *prod-assoc-epoll*: $(A * B) * C \approx A * (B * C)$
<proof>

lemma *well-ord-cmult-assoc*:
[[*well-ord*(i, ri); *well-ord*(j, rj); *well-ord*(k, rk)]]
==> ($i \mid * \mid j$) $\mid * \mid k = i \mid * \mid (j \mid * \mid k)$
<proof>

33.2.3 Cardinal multiplication distributes over addition

lemma *sum-prod-distrib-epoll*: $(A + B) * C \approx (A * C) + (B * C)$
<proof>

lemma *well-ord-cadd-cmult-distrib*:
[[*well-ord*(i, ri); *well-ord*(j, rj); *well-ord*(k, rk)]]
==> ($i \mid + \mid j$) $\mid * \mid k = (i \mid * \mid k) \mid + \mid (j \mid * \mid k)$
<proof>

33.2.4 Multiplication by 0 yields 0

lemma *prod-0-epoll*: $0 * A \approx 0$
<proof>

lemma *cmult-0 [simp]*: $0 \mid * \mid i = 0$
<proof>

33.2.5 1 is the identity for multiplication

lemma *prod-singleton-epoll*: $\{x\} * A \approx A$
<proof>

lemma *cmult-1 [simp]*: $\text{Card}(K) ==> 1 \mid * \mid K = K$
<proof>

33.3 Some inequalities for multiplication

lemma *prod-square-lepoll*: $A \lesssim A * A$
<proof>

lemma *cmult-square-le*: $\text{Card}(K) \implies K \text{ le } K \mid * \mid K$
 $\langle \text{proof} \rangle$

33.3.1 Multiplication by a non-zero cardinal

lemma *prod-lepoll-self*: $b: B \implies A \lesssim A * B$
 $\langle \text{proof} \rangle$

lemma *cmult-le-self*:
 $\llbracket \text{Card}(K); \text{Ord}(L); 0 < L \rrbracket \implies K \text{ le } (K \mid * \mid L)$
 $\langle \text{proof} \rangle$

33.3.2 Monotonicity of multiplication

lemma *prod-lepoll-mono*:
 $\llbracket A \lesssim C; B \lesssim D \rrbracket \implies A * B \lesssim C * D$
 $\langle \text{proof} \rangle$

lemma *cmult-le-mono*:
 $\llbracket K' \text{ le } K; L' \text{ le } L \rrbracket \implies (K' \mid * \mid L') \text{ le } (K \mid * \mid L)$
 $\langle \text{proof} \rangle$

33.4 Multiplication of finite cardinals is "ordinary" multiplication

lemma *prod-succ-epoll*: $\text{succ}(A) * B \approx B + A * B$
 $\langle \text{proof} \rangle$

lemma *cmult-succ-lemma*:
 $\llbracket \text{Ord}(m); \text{Ord}(n) \rrbracket \implies \text{succ}(m) \mid * \mid n = n \mid + \mid (m \mid * \mid n)$
 $\langle \text{proof} \rangle$

lemma *nat-cmult-eq-mult*: $\llbracket m: \text{nat}; n: \text{nat} \rrbracket \implies m \mid * \mid n = m \# * n$
 $\langle \text{proof} \rangle$

lemma *cmult-2*: $\text{Card}(n) \implies 2 \mid * \mid n = n \mid + \mid n$
 $\langle \text{proof} \rangle$

lemma *sum-lepoll-prod*: $2 \lesssim C \implies B + B \lesssim C * B$
 $\langle \text{proof} \rangle$

lemma *lepoll-imp-sum-lepoll-prod*: $\llbracket A \lesssim B; 2 \lesssim A \rrbracket \implies A + B \lesssim A * B$
 $\langle \text{proof} \rangle$

33.5 Infinite Cardinals are Limit Ordinals

lemma *nat-cons-lepoll*: $\text{nat} \lesssim A \implies \text{cons}(u, A) \lesssim A$
 $\langle \text{proof} \rangle$

lemma *nat-cons-epoll*: $\text{nat} \lesssim A \implies \text{cons}(u, A) \approx A$
 $\langle \text{proof} \rangle$

lemma *nat-succ-epoll*: $\text{nat} \leq A \implies \text{succ}(A) \approx A$
 $\langle \text{proof} \rangle$

lemma *InfCard-nat*: $\text{InfCard}(\text{nat})$
 $\langle \text{proof} \rangle$

lemma *InfCard-is-Card*: $\text{InfCard}(K) \implies \text{Card}(K)$
 $\langle \text{proof} \rangle$

lemma *InfCard-Un*:
 $\llbracket \text{InfCard}(K); \text{Card}(L) \rrbracket \implies \text{InfCard}(K \text{ Un } L)$
 $\langle \text{proof} \rangle$

lemma *InfCard-is-Limit*: $\text{InfCard}(K) \implies \text{Limit}(K)$
 $\langle \text{proof} \rangle$

lemma *ordermap-epoll-pred*:
 $\llbracket \text{well-ord}(A, r); x:A \rrbracket \implies \text{ordermap}(A, r) \text{ ' } x \approx \text{Order.pred}(A, x, r)$
 $\langle \text{proof} \rangle$

33.5.1 Establishing the well-ordering

lemma *csquare-lam-inj*:
 $\text{Ord}(K) \implies (\text{lam } \langle x, y \rangle : K * K. \langle x \text{ Un } y, x, y \rangle) : \text{inj}(K * K, K * K * K)$
 $\langle \text{proof} \rangle$

lemma *well-ord-csquare*: $\text{Ord}(K) \implies \text{well-ord}(K * K, \text{csquare-rel}(K))$
 $\langle \text{proof} \rangle$

33.5.2 Characterising initial segments of the well-ordering

lemma *csquareD*:
 $\llbracket \langle \langle x, y \rangle, \langle z, z \rangle \rangle : \text{csquare-rel}(K); x < K; y < K; z < K \rrbracket \implies x \text{ le } z \ \& \ y \text{ le } z$
 $\langle \text{proof} \rangle$

lemma *pred-csquare-subset*:

$z < K \implies \text{Order.pred}(K * K, <z, z>, \text{csquare-rel}(K)) \leq \text{succ}(z) * \text{succ}(z)$
 $\langle \text{proof} \rangle$

lemma *csquare-ltI*:

$[| x < z; y < z; z < K |] \implies <<x, y>, <z, z>> : \text{csquare-rel}(K)$
 $\langle \text{proof} \rangle$

lemma *csquare-or-eqI*:

$[| x \leq z; y \leq z; z < K |] \implies <<x, y>, <z, z>> : \text{csquare-rel}(K) \mid x = z \ \& \ y = z$
 $\langle \text{proof} \rangle$

33.5.3 The cardinality of initial segments

lemma *ordermap-z-lt*:

$[| \text{Limit}(K); x < K; y < K; z = \text{succ}(x \text{ Un } y) |] \implies$
 $\text{ordermap}(K * K, \text{csquare-rel}(K)) \text{ ' } <x, y> <$
 $\text{ordermap}(K * K, \text{csquare-rel}(K)) \text{ ' } <z, z>$
 $\langle \text{proof} \rangle$

lemma *ordermap-csquare-le*:

$[| \text{Limit}(K); x < K; y < K; z = \text{succ}(x \text{ Un } y) |]$
 $\implies | \text{ordermap}(K * K, \text{csquare-rel}(K)) \text{ ' } <x, y> | \leq | \text{succ}(z) | * | \text{succ}(z) |$
 $\langle \text{proof} \rangle$

lemma *ordertype-csquare-le*:

$[| \text{InfCard}(K); \text{ALL } y:K. \text{InfCard}(y) \dashrightarrow y * y = y |]$
 $\implies \text{ordertype}(K * K, \text{csquare-rel}(K)) \leq K$
 $\langle \text{proof} \rangle$

lemma *InfCard-csquare-eq*: $\text{InfCard}(K) \implies K * K = K$

$\langle \text{proof} \rangle$

lemma *well-ord-InfCard-square-eq*:

$[| \text{well-ord}(A, r); \text{InfCard}(|A|) |] \implies A * A \approx A$
 $\langle \text{proof} \rangle$

lemma *InfCard-square-eqpoll*: $\text{InfCard}(K) \implies K \times K \approx K$

$\langle \text{proof} \rangle$

lemma *Inf-Card-is-InfCard*: $[| \sim \text{Finite}(i); \text{Card}(i) |] \implies \text{InfCard}(i)$

$\langle \text{proof} \rangle$

33.5.4 Toward's Kunen's Corollary 10.13 (1)

lemma *InfCard-le-cmult-eq*: $[[\text{InfCard}(K); L \leq K; 0 < L]] \implies K \mid * \mid L = K$
 $\langle \text{proof} \rangle$

lemma *InfCard-cmult-eq*: $[[\text{InfCard}(K); \text{InfCard}(L)]] \implies K \mid * \mid L = K \cup_n L$
 $\langle \text{proof} \rangle$

lemma *InfCard-cdouble-eq*: $\text{InfCard}(K) \implies K \mid + \mid K = K$
 $\langle \text{proof} \rangle$

lemma *InfCard-le-cadd-eq*: $[[\text{InfCard}(K); L \leq K]] \implies K \mid + \mid L = K$
 $\langle \text{proof} \rangle$

lemma *InfCard-cadd-eq*: $[[\text{InfCard}(K); \text{InfCard}(L)]] \implies K \mid + \mid L = K \cup_n L$
 $\langle \text{proof} \rangle$

33.6 For Every Cardinal Number There Exists A Greater One

This result is Kunen's Theorem 10.16, which would be trivial using AC

lemma *Ord-jump-cardinal*: $\text{Ord}(\text{jump-cardinal}(K))$
 $\langle \text{proof} \rangle$

lemma *jump-cardinal-iff*:
 $i : \text{jump-cardinal}(K) < - >$
 $(\exists X \ r \ X. \ r \leq K * K \ \& \ X \leq K \ \& \ \text{well-ord}(X, r) \ \& \ i = \text{ordertype}(X, r))$
 $\langle \text{proof} \rangle$

lemma *K-lt-jump-cardinal*: $\text{Ord}(K) \implies K < \text{jump-cardinal}(K)$
 $\langle \text{proof} \rangle$

lemma *Card-jump-cardinal-lemma*:
 $[[\text{well-ord}(X, r); \ r \leq K * K; \ X \leq K;$
 $\quad f : \text{bij}(\text{ordertype}(X, r), \text{jump-cardinal}(K)) \]]$
 $\implies \text{jump-cardinal}(K) : \text{jump-cardinal}(K)$
 $\langle \text{proof} \rangle$

lemma *Card-jump-cardinal*: $\text{Card}(\text{jump-cardinal}(K))$
 $\langle \text{proof} \rangle$

33.7 Basic Properties of Successor Cardinals

lemma *csucc-basic*: $\text{Ord}(K) \implies \text{Card}(\text{csucc}(K)) \ \& \ K < \text{csucc}(K)$
 $\langle \text{proof} \rangle$

lemmas *Card-csucc* = *csucc-basic* [THEN conjunct1, standard]

lemmas *lt-csucc* = *csucc-basic* [THEN conjunct2, standard]

lemma *Ord-0-lt-csucc*: $\text{Ord}(K) \implies 0 < \text{csucc}(K)$
 $\langle \text{proof} \rangle$

lemma *csucc-le*: $[\text{Card}(L); K < L] \implies \text{csucc}(K) \text{ le } L$
 $\langle \text{proof} \rangle$

lemma *lt-csucc-iff*: $[\text{Ord}(i); \text{Card}(K)] \implies i < \text{csucc}(K) \iff |i| \text{ le } K$
 $\langle \text{proof} \rangle$

lemma *Card-lt-csucc-iff*:
 $[\text{Card}(K'); \text{Card}(K)] \implies K' < \text{csucc}(K) \iff K' \text{ le } K$
 $\langle \text{proof} \rangle$

lemma *InfCard-csucc*: $\text{InfCard}(K) \implies \text{InfCard}(\text{csucc}(K))$
 $\langle \text{proof} \rangle$

33.7.1 Removing elements from a finite set decreases its cardinality

lemma *Fin-imp-not-cons-lepoll*: $A: \text{Fin}(U) \implies x \sim : A \dashrightarrow \sim \text{cons}(x, A) \lesssim A$
 $\langle \text{proof} \rangle$

lemma *Finite-imp-cardinal-cons* [simp]:
 $[\text{Finite}(A); a \sim : A] \implies |\text{cons}(a, A)| = \text{succ}(|A|)$
 $\langle \text{proof} \rangle$

lemma *Finite-imp-succ-cardinal-Diff*:
 $[\text{Finite}(A); a : A] \implies \text{succ}(|A - \{a\}|) = |A|$
 $\langle \text{proof} \rangle$

lemma *Finite-imp-cardinal-Diff*: $[\text{Finite}(A); a : A] \implies |A - \{a\}| < |A|$
 $\langle \text{proof} \rangle$

lemma *Finite-cardinal-in-nat* [simp]: $\text{Finite}(A) \implies |A| : \text{nat}$
 $\langle \text{proof} \rangle$

lemma *card-Un-Int*:
 $[\text{Finite}(A); \text{Finite}(B)] \implies |A| \# + |B| = |A \text{ Un } B| \# + |A \text{ Int } B|$
 $\langle \text{proof} \rangle$

lemma *card-Un-disjoint*:

$[|Finite(A); Finite(B); A \text{ Int } B = 0|] ==> |A \text{ Un } B| = |A| \# + |B|$
 $\langle proof \rangle$

lemma *card-partition* [rule-format]:

$Finite(C) ==>$
 $Finite(\bigcup C) -->$
 $(\forall c \in C. |c| = k) -->$
 $(\forall c1 \in C. \forall c2 \in C. c1 \neq c2 --> c1 \cap c2 = 0) -->$
 $k \# * |C| = |\bigcup C|$
 $\langle proof \rangle$

33.7.2 Theorems by Krzysztof Grabczewski, proofs by lcp

lemmas *nat-implies-well-ord* = *nat-into-Ord* [THEN *well-ord-Memrel*, *standard*]

lemma *nat-sum-egpoll-sum*: $[| m:nat; n:nat |] ==> m + n \approx m \# + n$
 $\langle proof \rangle$

lemma *Ord-subset-natD* [rule-format]: $Ord(i) ==> i \leq nat --> i : nat \mid i=nat$
 $\langle proof \rangle$

lemma *Ord-nat-subset-into-Card*: $[| Ord(i); i \leq nat |] ==> Card(i)$
 $\langle proof \rangle$

lemma *Finite-Diff-sing-eq-diff-1*: $[| Finite(A); x:A |] ==> |A - \{x\}| = |A| \# - 1$
 $\langle proof \rangle$

lemma *cardinal-lt-imp-Diff-not-0* [rule-format]:

$Finite(B) ==> ALL A. |B| < |A| --> A - B \neq 0$
 $\langle proof \rangle$

$\langle ML \rangle$

end

34 Main-ZF: Theory Main: Everything Except AC

theory *Main-ZF* **imports** *List-ZF IntDiv-ZF CardinalArith* **begin**

34.1 Iteration of the function F

consts *iterates* :: $[i=>i,i,i] ==> i \quad ((-^{\wedge} - '(-')) [60,1000,1000] 60)$

primrec

$F^{\wedge} 0 (x) = x$
 $F^{\wedge} (succ(n)) (x) = F(F^{\wedge} n (x))$

definition

$iterates\text{-}\omega :: [i \Rightarrow i, i] \Rightarrow i$ **where**
 $iterates\text{-}\omega(F, x) == \bigcup_{n \in nat.} F^n(x)$

notation (*xsymbols*)

$iterates\text{-}\omega \quad ((-)^{\omega} \text{ '(-)}) \quad [60, 1000] \quad 60$

notation (*HTML output*)

$iterates\text{-}\omega \quad ((-)^{\omega} \text{ '(-)}) \quad [60, 1000] \quad 60$

lemma *iterates-triv*:

$[| \ n \in nat; \ F(x) = x \ |] \Rightarrow F^n(x) = x$
 $\langle proof \rangle$

lemma *iterates-type* $[TC]$:

$[| \ n \in nat; \ a : A; \ !!x. x : A \Rightarrow F(x) : A \ |]$
 $\Rightarrow F^n(a) : A$
 $\langle proof \rangle$

lemma *iterates-omega-triv*:

$F(x) = x \Rightarrow F^{\omega}(x) = x$
 $\langle proof \rangle$

lemma *Ord-iterates* $[simp]$:

$[| \ n \in nat; \ !!i. Ord(i) \Rightarrow Ord(F(i)); \ Ord(x) \ |]$
 $\Rightarrow Ord(F^n(x))$
 $\langle proof \rangle$

lemma *iterates-commute*: $n \in nat \Rightarrow F(F^n(x)) = F^n(F(x))$

$\langle proof \rangle$

34.2 Transfinite Recursion

Transfinite recursion for definitions based on the three cases of ordinals

definition

$transrec3 :: [i, i, [i, i] \Rightarrow i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $transrec3(k, a, b, c) ==$
 $transrec(k, \lambda x. r.$
 $\quad \text{if } x=0 \text{ then } a$
 $\quad \text{else if } Limit(x) \text{ then } c(x, \lambda y \in x. r'y)$
 $\quad \text{else } b(Arith.pred(x), r \text{ ' } Arith.pred(x)))$

lemma *transrec3-0* $[simp]$: $transrec3(0, a, b, c) = a$

$\langle proof \rangle$

lemma *transrec3-succ* $[simp]$:

$transrec3(succ(i), a, b, c) = b(i, transrec3(i, a, b, c))$
 $\langle proof \rangle$

lemma *transrec3-Limit*:
 $Limit(i) ==>$
 $transrec3(i, a, b, c) = c(i, \lambda j \in i. transrec3(j, a, b, c))$
 $\langle proof \rangle$

$\langle ML \rangle$

end

theory *Main*
imports *Main-ZF*
begin

end

35 AC: The Axiom of Choice

theory *AC* **imports** *Main-ZF* **begin**

This definition comes from Halmos (1960), page 59.

axiomatization **where**

$AC: [\mid a: A; \mid \mid x. x:A ==> (EX y. y:B(x)) \mid] ==> EX z. z : Pi(A,B)$

lemma *AC-Pi*: $[\mid \mid x. x \in A ==> (\exists y. y \in B(x)) \mid] ==> \exists z. z \in Pi(A,B)$
 $\langle proof \rangle$

lemma *AC-ball-Pi*: $\forall x \in A. \exists y. y \in B(x) ==> \exists y. y \in Pi(A,B)$
 $\langle proof \rangle$

lemma *AC-Pi-Pow*: $\exists f. f \in (\Pi X \in Pow(C) - \{0\}. X)$
 $\langle proof \rangle$

lemma *AC-func*:
 $[\mid \mid x. x \in A ==> (\exists y. y \in x) \mid] ==> \exists f \in A \rightarrow Union(A). \forall x \in A. f'x \in x$
 $\langle proof \rangle$

lemma *non-empty-family*: $[\mid 0 \notin A; \mid x \in A \mid] ==> \exists y. y \in x$
 $\langle proof \rangle$

lemma *AC-func0*: $0 \notin A ==> \exists f \in A \rightarrow Union(A). \forall x \in A. f'x \in x$
 $\langle proof \rangle$

lemma *AC-func-Pow*: $\exists f \in (Pow(C) - \{0\}) \rightarrow C. \forall x \in Pow(C) - \{0\}. f'x \in x$
 $\langle proof \rangle$

lemma *AC-Pi0*: $0 \notin A \implies \exists f. f \in (\prod x \in A. x)$
 $\langle proof \rangle$

end

36 Zorn: Zorn's Lemma

theory *Zorn* **imports** *OrderArith AC Inductive-ZF* **begin**

Based upon the unpublished article “Towards the Mechanization of the Proofs of Some Classical Theorems of Set Theory,” by Abrial and Laffitte.

definition

$Subset-rel :: i \Rightarrow i$ **where**
 $Subset-rel(A) == \{z \in A * A . \exists x y. z = \langle x, y \rangle \ \& \ x \leq y \ \& \ x \neq y\}$

definition

$chain :: i \Rightarrow i$ **where**
 $chain(A) == \{F \in Pow(A). \forall X \in F. \forall Y \in F. X \leq Y \mid Y \leq X\}$

definition

$super :: [i, i] \Rightarrow i$ **where**
 $super(A, c) == \{d \in chain(A). c \leq d \ \& \ c \neq d\}$

definition

$maxchain :: i \Rightarrow i$ **where**
 $maxchain(A) == \{c \in chain(A). super(A, c) = 0\}$

definition

$increasing :: i \Rightarrow i$ **where**
 $increasing(A) == \{f \in Pow(A) \rightarrow Pow(A). \forall x. x \leq A \implies x \leq f'x\}$

Lemma for the inductive definition below

lemma *Union-in-Pow*: $Y \in Pow(Pow(A)) \implies Union(Y) \in Pow(A)$
 $\langle proof \rangle$

We could make the inductive definition conditional on $next \in increasing(S)$ but instead we make this a side-condition of an introduction rule. Thus the induction rule lets us assume that condition! Many inductive proofs are therefore unconditional.

consts

$TFin :: [i, i] \Rightarrow i$

inductive

domains $TFin(S, next) \leq Pow(S)$

intros

$nextI: \quad [| x \in TFin(S, next); next \in increasing(S) |]$
 $\implies next'x \in TFin(S, next)$

Pow-UnionI: $Y \in \text{Pow}(\text{TFin}(S, \text{next})) \implies \text{Union}(Y) \in \text{TFin}(S, \text{next})$

monos *Pow-mono*
con-defs *increasing-def*
type-intros *CollectD1 [THEN apply-funtype] Union-in-Pow*

36.1 Mathematical Preamble

lemma *Union-lemma0*: $(\forall x \in C. x \leq A \mid B \leq x) \implies \text{Union}(C) \leq A \mid B \leq \text{Union}(C)$
 $\langle \text{proof} \rangle$

lemma *Inter-lemma0*:
 $\llbracket c \in C; \forall x \in C. A \leq x \mid x \leq B \rrbracket \implies A \leq \text{Inter}(C) \mid \text{Inter}(C) \leq B$
 $\langle \text{proof} \rangle$

36.2 The Transfinite Construction

lemma *increasingD1*: $f \in \text{increasing}(A) \implies f \in \text{Pow}(A) \multimap \text{Pow}(A)$
 $\langle \text{proof} \rangle$

lemma *increasingD2*: $\llbracket f \in \text{increasing}(A); x \leq A \rrbracket \implies x \leq f'x$
 $\langle \text{proof} \rangle$

lemmas *TFin-UnionI* = *PowI [THEN TFin.Pow-UnionI, standard]*

lemmas *TFin-is-subset* = *TFin.dom-subset [THEN subsetD, THEN PowD, standard]*

Structural induction on $\text{TFin}(S, \text{next})$

lemma *TFin-induct*:
 $\llbracket n \in \text{TFin}(S, \text{next});$
 $\quad \text{!!}x. \llbracket x \in \text{TFin}(S, \text{next}); P(x); \text{next} \in \text{increasing}(S) \rrbracket \implies P(\text{next}'x);$
 $\quad \text{!!}Y. \llbracket Y \leq \text{TFin}(S, \text{next}); \forall y \in Y. P(y) \rrbracket \implies P(\text{Union}(Y))$
 $\quad \rrbracket \implies P(n)$
 $\langle \text{proof} \rangle$

36.3 Some Properties of the Transfinite Construction

lemmas *increasing-trans* = *subset-trans [OF - increasingD2,*
 $\quad \text{OF - - TFin-is-subset}]$

Lemma 1 of section 3.1

lemma *TFin-linear-lemma1*:
 $\llbracket n \in \text{TFin}(S, \text{next}); m \in \text{TFin}(S, \text{next});$
 $\quad \forall x \in \text{TFin}(S, \text{next}). x \leq m \dashv\vdash x = m \mid \text{next}'x \leq m \rrbracket$
 $\implies n \leq m \mid \text{next}'m \leq n$
 $\langle \text{proof} \rangle$

Lemma 2 of section 3.2. Interesting in its own right! Requires $next \in increasing(S)$ in the second induction step.

lemma *TFin-linear-lemma2*:

$$[| m \in TFin(S, next); next \in increasing(S) |]$$

$$==> \forall n \in TFin(S, next). n \leq m \leftrightarrow n = m \mid next'n \leq m$$

 $\langle proof \rangle$

a more convenient form for Lemma 2

lemma *TFin-subsetD*:

$$[| n \leq m; m \in TFin(S, next); n \in TFin(S, next); next \in increasing(S) |]$$

$$==> n = m \mid next'n \leq m$$

 $\langle proof \rangle$

Consequences from section 3.3 – Property 3.2, the ordering is total

lemma *TFin-subset-linear*:

$$[| m \in TFin(S, next); n \in TFin(S, next); next \in increasing(S) |]$$

$$==> n \leq m \mid m \leq n$$

 $\langle proof \rangle$

Lemma 3 of section 3.3

lemma *equal-next-upper*:

$$[| n \in TFin(S, next); m \in TFin(S, next); m = next'm |] ==> n \leq m$$

 $\langle proof \rangle$

Property 3.3 of section 3.3

lemma *equal-next-Union*:

$$[| m \in TFin(S, next); next \in increasing(S) |]$$

$$==> m = next'm \leftrightarrow m = Union(TFin(S, next))$$

 $\langle proof \rangle$

36.4 Hausdorff's Theorem: Every Set Contains a Maximal Chain

NOTE: We assume the partial ordering is \subseteq , the subset relation!

* Defining the "next" operation for Hausdorff's Theorem *

lemma *chain-subset-Pow*: $chain(A) \leq Pow(A)$

$\langle proof \rangle$

lemma *super-subset-chain*: $super(A, c) \leq chain(A)$

$\langle proof \rangle$

lemma *maxchain-subset-chain*: $maxchain(A) \leq chain(A)$

$\langle proof \rangle$

lemma *choice-super*:

$$[| ch \in (\Pi X \in Pow(chain(S)) - \{0\}. X); X \in chain(S); X \notin maxchain(S) |]$$

$\implies ch \text{ ' } super(S,X) \in super(S,X)$
 $\langle proof \rangle$

lemma *choice-not-equals*:

$[[ch \in (\Pi X \in Pow(chain(S)) - \{0\}. X); X \in chain(S); X \notin maxchain(S)$
 $]]$
 $\implies ch \text{ ' } super(S,X) \neq X$
 $\langle proof \rangle$

This justifies Definition 4.4

lemma *Hausdorff-next-exists*:

$ch \in (\Pi X \in Pow(chain(S)) - \{0\}. X) \implies$
 $\exists next \in increasing(S). \forall X \in Pow(S).$
 $next \text{ ' } X = if(X \in chain(S) - maxchain(S), ch \text{ ' } super(S,X), X)$
 $\langle proof \rangle$

Lemma 4

lemma *TFin-chain-lemma4*:

$[[c \in TFin(S,next);$
 $ch \in (\Pi X \in Pow(chain(S)) - \{0\}. X);$
 $next \in increasing(S);$
 $\forall X \in Pow(S). next \text{ ' } X =$
 $if(X \in chain(S) - maxchain(S), ch \text{ ' } super(S,X), X)]]$
 $\implies c \in chain(S)$
 $\langle proof \rangle$

theorem *Hausdorff*: $\exists c. c \in maxchain(S)$
 $\langle proof \rangle$

36.5 Zorn's Lemma: If All Chains in S Have Upper Bounds In S, then S contains a Maximal Element

Used in the proof of Zorn's Lemma

lemma *chain-extend*:

$[[c \in chain(A); z \in A; \forall x \in c. x \leq z]]$ $\implies cons(z,c) \in chain(A)$
 $\langle proof \rangle$

lemma *Zorn*: $\forall c \in chain(S). Union(c) \in S \implies \exists y \in S. \forall z \in S. y \leq z \implies y=z$
 $\langle proof \rangle$

Alternative version of Zorn's Lemma

theorem *Zorn2*:

$\forall c \in chain(S). \exists y \in S. \forall x \in c. x \leq y \implies \exists y \in S. \forall z \in S. y \leq z \implies y=z$
 $\langle proof \rangle$

36.6 Zermelo's Theorem: Every Set can be Well-Ordered

Lemma 5

lemma *TFin-well-lemma5*:

$$[[n \in TFin(S, next); Z \leq TFin(S, next); z:Z; \sim Inter(Z) \in Z]]$$

$$\implies \forall m \in Z. n \leq m$$

<proof>

Well-ordering of $TFin(S, next)$

lemma *well-ord-TFin-lemma*: $[[Z \leq TFin(S, next); z \in Z]] \implies Inter(Z) \in Z$

<proof>

This theorem just packages the previous result

lemma *well-ord-TFin*:

$$next \in increasing(S)$$

$$\implies well\text{-}ord(TFin(S, next), Subset\text{-}rel(TFin(S, next)))$$

<proof>

* Defining the "next" operation for Zermelo's Theorem *

lemma *choice-Diff*:

$$[[ch \in (\Pi X \in Pow(S) - \{0\}. X); X \subseteq S; X \neq S]] \implies ch'(S - X) \in S - X$$

<proof>

This justifies Definition 6.1

lemma *Zermelo-next-exists*:

$$ch \in (\Pi X \in Pow(S) - \{0\}. X) \implies$$

$$\exists next \in increasing(S). \forall X \in Pow(S).$$

$$next'X = (if X=S then S else cons(ch'(S-X), X))$$

<proof>

The construction of the injection

lemma *choice-imp-injection*:

$$[[ch \in (\Pi X \in Pow(S) - \{0\}. X);$$

$$next \in increasing(S);$$

$$\forall X \in Pow(S). next'X = if(X=S, S, cons(ch'(S-X), X))]]$$

$$\implies (\lambda x \in S. Union(\{y \in TFin(S, next). x \notin y\}))$$

$$\in inj(S, TFin(S, next) - \{S\})$$

<proof>

The wellordering theorem

theorem *AC-well-ord*: $\exists r. well\text{-}ord(S, r)$

<proof>

36.7 Zorn's Lemma for Partial Orders

Reimported from HOL by Clemens Ballarin.

definition *Chain* :: $i \Rightarrow i$ **where**

$Chain(r) = \{A : Pow(field(r)). \text{ ALL } a:A. \text{ ALL } b:A. \langle a, b \rangle : r \mid \langle b, a \rangle : r\}$

lemma *mono-Chain*:

$r \subseteq s \Rightarrow Chain(r) \subseteq Chain(s)$

$\langle proof \rangle$

theorem *Zorn-po*:

assumes *po*: *Partial-order*(r)

and u : *ALL* $C:Chain(r)$. *EX* $u:field(r)$. *ALL* $a:C$. $\langle a, u \rangle : r$

shows *EX* $m:field(r)$. *ALL* $a:field(r)$. $\langle m, a \rangle : r \dashv\dashv a = m$

$\langle proof \rangle$

end

37 Cardinal-AC: Cardinal Arithmetic Using AC

theory *Cardinal-AC* **imports** *CardinalArith* *Zorn* **begin**

37.1 Strengthened Forms of Existing Theorems on Cardinals

lemma *cardinal-eqpoll*: $|A| \text{ eqpoll } A$

$\langle proof \rangle$

The theorem $||A|| = |A|$

lemmas *cardinal-idem* = *cardinal-eqpoll* [*THEN* *cardinal-cong*, *standard*, *simp*]

lemma *cardinal-eqE*: $|X| = |Y| \Rightarrow X \text{ eqpoll } Y$

$\langle proof \rangle$

lemma *cardinal-eqpoll-iff*: $|X| = |Y| \Leftrightarrow X \text{ eqpoll } Y$

$\langle proof \rangle$

lemma *cardinal-disjoint-Un*:

$[| |A|=|B|; |C|=|D|; A \text{ Int } C = 0; B \text{ Int } D = 0]$

$\Rightarrow |A \text{ Un } C| = |B \text{ Un } D|$

$\langle proof \rangle$

lemma *lepoll-imp-Card-le*: $A \text{ lepoll } B \Rightarrow |A| \text{ le } |B|$

$\langle proof \rangle$

lemma *cadd-assoc*: $(i \mid + \mid j) \mid + \mid k = i \mid + \mid (j \mid + \mid k)$

$\langle proof \rangle$

lemma *cmult-assoc*: $(i \mid * \mid j) \mid * \mid k = i \mid * \mid (j \mid * \mid k)$

$\langle proof \rangle$

lemma *cadd-cmult-distrib*: $(i \mid + \mid j) \mid * \mid k = (i \mid * \mid k) \mid + \mid (j \mid * \mid k)$

$\langle proof \rangle$

lemma *InfCard-square-eq*: $InfCard(|A|) ==> A * A \text{ eqpoll } A$
 $\langle proof \rangle$

37.2 The relationship between cardinality and le-pollence

lemma *Card-le-imp-lepoll*: $|A| \text{ le } |B| ==> A \text{ lepoll } B$
 $\langle proof \rangle$

lemma *le-Card-iff*: $Card(K) ==> |A| \text{ le } K <-> A \text{ lepoll } K$
 $\langle proof \rangle$

lemma *cardinal-0-iff-0 [simp]*: $|A| = 0 <-> A = 0$
 $\langle proof \rangle$

lemma *cardinal-lt-iff-lesspoll*: $Ord(i) ==> i < |A| <-> i \text{ lesspoll } A$
 $\langle proof \rangle$

lemma *cardinal-le-imp-lepoll*: $i \leq |A| ==> i \lesssim A$
 $\langle proof \rangle$

37.3 Other Applications of AC

lemma *surj-implies-inj*: $f: \text{surj}(X, Y) ==> \exists X \text{ g. } g: \text{inj}(Y, X)$
 $\langle proof \rangle$

lemma *surj-implies-cardinal-le*: $f: \text{surj}(X, Y) ==> |Y| \text{ le } |X|$
 $\langle proof \rangle$

lemma *cardinal-UN-le*:
 $[| \text{InfCard}(K); \text{ALL } i:K. |X(i)| \text{ le } K |] ==> |\bigcup i \in K. X(i)| \text{ le } K$
 $\langle proof \rangle$

lemma *cardinal-UN-lt-csucc*:
 $[| \text{InfCard}(K); \text{ALL } i:K. |X(i)| < \text{csucc}(K) |]$
 $==> |\bigcup i \in K. X(i)| < \text{csucc}(K)$
 $\langle proof \rangle$

lemma *cardinal-UN-Ord-lt-csucc*:
 $[| \text{InfCard}(K); \text{ALL } i:K. j(i) < \text{csucc}(K) |]$
 $==> (\bigcup i \in K. j(i)) < \text{csucc}(K)$
 $\langle proof \rangle$

lemma *inj-UN-subset*:

$$[| f: \text{inj}(A,B); \ a:A |] ==>$$

$$(\bigcup_{x \in A}. C(x)) \leq (\bigcup_{y \in B}. C(\text{if } y: \text{range}(f) \text{ then } \text{converse}(f) 'y \text{ else } a))$$

$$\langle \text{proof} \rangle$$

lemma *le-UN-Ord-lt-csucc*:

$$[| \text{InfCard}(K); \ |W| \text{ le } K; \ \text{ALL } w:W. j(w) < \text{csucc}(K) |]$$

$$==> (\bigcup_{w \in W}. j(w)) < \text{csucc}(K)$$

$$\langle \text{proof} \rangle$$

$$\langle ML \rangle$$

end

38 InfDatatype: Infinite-Branching Datatype Definitions

theory *InfDatatype* **imports** *Datatype-ZF Univ Finite Cardinal-AC* **begin**

lemmas *fun-Limit-VfromE* =

$$\text{Limit-VfromE} \ [OF \ \text{apply-funtype} \ \text{InfCard-csucc} \ [THEN \ \text{InfCard-is-Limit}]]$$

lemma *fun-Vcsucc-lemma*:

$$[| f: D \rightarrow Vfrom(A, \text{csucc}(K)); \ |D| \text{ le } K; \ \text{InfCard}(K) |]$$

$$==> \text{EX } j. f: D \rightarrow Vfrom(A, j) \ \& \ j < \text{csucc}(K)$$

$$\langle \text{proof} \rangle$$

lemma *subset-Vcsucc*:

$$[| D \leq Vfrom(A, \text{csucc}(K)); \ |D| \text{ le } K; \ \text{InfCard}(K) |]$$

$$==> \text{EX } j. D \leq Vfrom(A, j) \ \& \ j < \text{csucc}(K)$$

$$\langle \text{proof} \rangle$$

lemma *fun-Vcsucc*:

$$[| |D| \text{ le } K; \ \text{InfCard}(K); \ D \leq Vfrom(A, \text{csucc}(K)) |] ==>$$

$$D \rightarrow Vfrom(A, \text{csucc}(K)) \leq Vfrom(A, \text{csucc}(K))$$

$$\langle \text{proof} \rangle$$

lemma *fun-in-Vcsucc*:

$$[| f: D \rightarrow Vfrom(A, \text{csucc}(K)); \ |D| \text{ le } K; \ \text{InfCard}(K);$$

$$D \leq Vfrom(A, \text{csucc}(K)) |]$$

$$==> f: D \rightarrow Vfrom(A, \text{csucc}(K))$$

$$\langle \text{proof} \rangle$$

lemmas *fun-in-Vcsucc' = fun-in-Vcsucc* [*OF - - - subsetI*]

lemma *Card-fun-Vcsucc:*

InfCard(K) ==> K -> Vfrom(A,csucc(K)) <= Vfrom(A,csucc(K))
<proof>

lemma *Card-fun-in-Vcsucc:*

[| f: K -> Vfrom(A, csucc(K)); InfCard(K) |] ==> f: Vfrom(A,csucc(K))
<proof>

lemma *Limit-csucc: InfCard(K) ==> Limit(csucc(K))*

<proof>

lemmas *Pair-in-Vcsucc = Pair-in-VLimit* [*OF - - Limit-csucc*]

lemmas *Inl-in-Vcsucc = Inl-in-VLimit* [*OF - Limit-csucc*]

lemmas *Inr-in-Vcsucc = Inr-in-VLimit* [*OF - Limit-csucc*]

lemmas *zero-in-Vcsucc = Limit-csucc* [*THEN zero-in-VLimit*]

lemmas *nat-into-Vcsucc = nat-into-VLimit* [*OF - Limit-csucc*]

lemmas *InfCard-nat-Un-cardinal = InfCard-Un* [*OF InfCard-nat Card-cardinal*]

lemmas *le-nat-Un-cardinal =*

Un-upper2-le [*OF Ord-nat Card-cardinal* [*THEN Card-is-Ord*]]

lemmas *UN-upper-cardinal = UN-upper* [*THEN subset-imp-lepoll, THEN lepoll-imp-Card-le*]

lemmas *Data-Arg-intros =*

SigmaI InlI InrI

Pair-in-univ Inl-in-univ Inr-in-univ

zero-in-univ A-into-univ nat-into-univ UnCI

lemmas *inf-datatype-intros =*

InfCard-nat InfCard-nat-Un-cardinal

Pair-in-Vcsucc Inl-in-Vcsucc Inr-in-Vcsucc

zero-in-Vcsucc A-into-Vfrom nat-into-Vcsucc

Card-fun-in-Vcsucc fun-in-Vcsucc' UN-I

end

theory *Main-ZFC imports Main-ZF InfDatatype*

begin

end