

HOL-Z

Achim D. Brucker Burkhard Wolf

February 6, 2007

Copyright (C) 2000–2006 Achim D. Brucker and Burkhard Wolff

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Free Software Foundation.

Important note:

This manual describes

Contents

A. HOL-Z Commands	5
A.1. Proof Commands	5
A.2. Support of the Z Refinement Method	5
B. Isabelle Theories	7
B.1. Core of the Mathematical Toolkit	7
B.2. Integers in the Mathematical Toolkit	31
B.3. Functions in the Mathematical Toolkit	37
B.3.1. Relation Apply	38
B.3.2. Relations	39
B.4. Finite Sets for the Mathematical Toolkit	48
B.4.1. Finite Sets	50
B.4.2. Finite Functions	52
B.4.3. Finite Number Range	54
B.5. Sequences in the Mathematical Toolkit	55
B.6. Support for a rudimentary lifter	64
B.7. A rudimentary Theory-Morphism from List to Sequences	68
B.8. Bags from the Mathematical Toolkit	79
B.8.1. Basic Theorems	81
B.8.2. Bag Union	83
B.9. Definitions of Syntax and Core-Combinators of Z	85
B.9.1. Initialize Z-Environment, provide Access	85
B.9.2. Syntax	85
B.9.3. Semantic Representation	89
B.9.4. Syntax: The parse-translation setup	91
B.9.5. Derived Rules	91
B.9.6. Proof Support: Z-tactics	94
B.10. Method Package for Z Representation and Z Refinement	94
B.10.1. Configuring the Generic PO Manager	94
B.10.2. The Z Method Package	95
B.11. Main Theory for the HOL-Z Environment	95
B.11.1. Proof Support: The .holz-loader (generated by ZETA)	96
B.11.2. Proof Support: Toplevel-Command zlemma	96

Contents

B.11.3. Baustelle: Code von DARMA zum integrieren 96

Appendix A.

HOL-Z Commands

A.1. Proof Commands

Attributes:

- tcsimp
- deltcsimp
- 2Z
- 2HOL
- stripZ

Methods:

- zrule_tac
- zfrule_tac
- zdrule_tac
- zerule_tac
- sowie zrule, zfrule, zdrule, zerule

- 2HOL_tac
- 2Z_tac

Toplevel:

- zlemma

A.2. Support of the Z Refinement Method

Generic part:

- list_po [except <po-class-name>]
- show_po <po-name>+
- check_po [except <po-class-name>]

Appendix A. HOL-Z Commands

```
discharge_po <po-name>
```

Z Method specific part:

```
gen_state_cc <name>
```

```
gen_op_cc    <name>
```

```
gen_thm_cc   <name>
```

```
set_abs <name> ["[" functional "]" ]
```

```
refine_init <name> <name>
```

```
refine_op   <name> <name>
```

Appendix B.

Isabelle Theories

The main dependencies are shown in Fig. B.1 on the following page. is build on top of Isabelle/HOL.¹

B.1. Core of the Mathematical Toolkit

theory *ZMathTool* **imports** *Finite-Set Sum-Type Main* **begin**

types (*'a*,*'b*) $\langle \Rightarrow \rangle$ = (*'a***'b*) *set* (**infixr** 20)

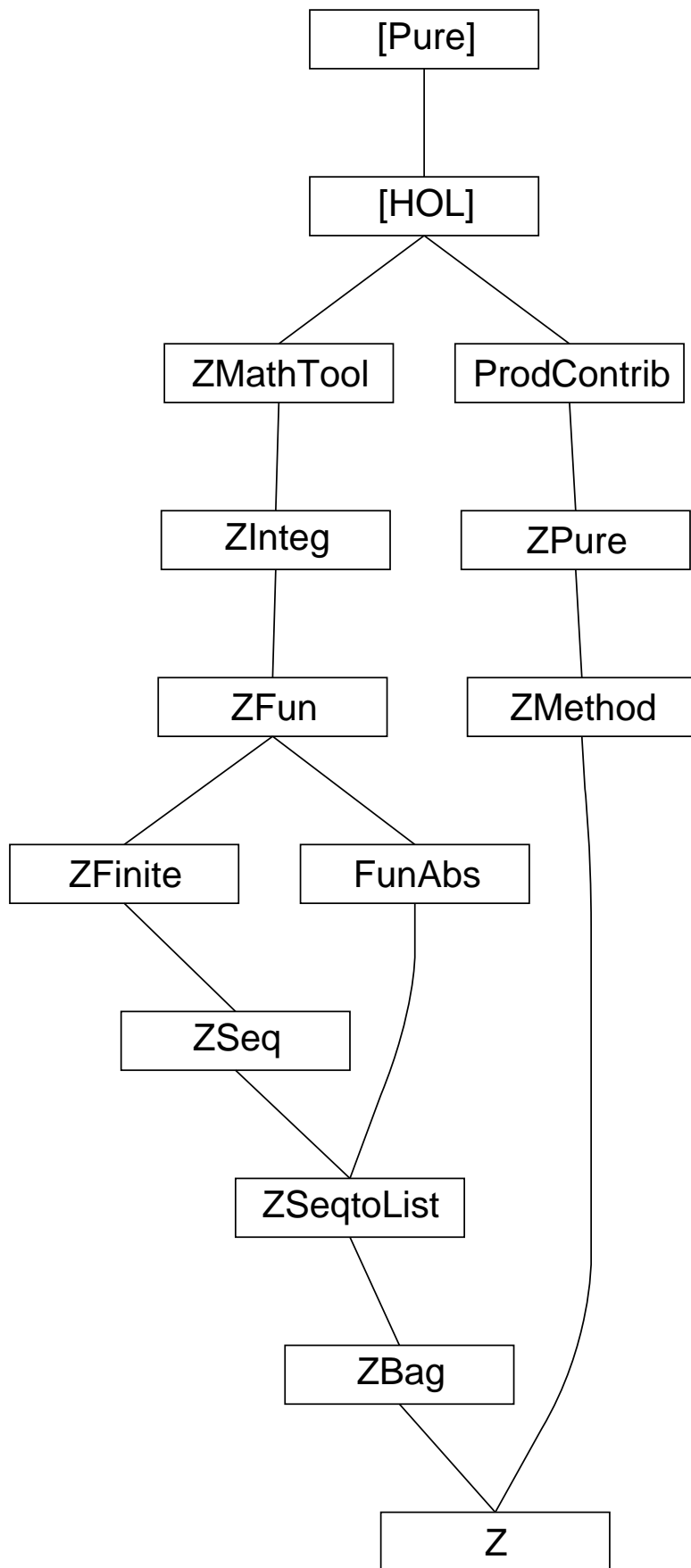
consts

idZ ::*'a set* \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'a*)
forw-comp ::[*'a* $\langle \Rightarrow \rangle$ *'b*, *'b* $\langle \Rightarrow \rangle$ *'c*] \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'c*) (- %; - [60,61] 60)
dom-restr ::[*'a set* , *'a* $\langle \Rightarrow \rangle$ *'b*] \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'b*) (- <: - [71,70] 70)
ran-restr ::[*'a* $\langle \Rightarrow \rangle$ *'b*, (*'b*)*set*] \Rightarrow *'a* $\langle \Rightarrow \rangle$ *'b* (- := - [65,66] 65)
dom-substr ::[(*'a*)*set* , *'a* $\langle \Rightarrow \rangle$ *'b*] \Rightarrow *'a* $\langle \Rightarrow \rangle$ *'b* (- <-: - [71,70] 70)
ran-substr ::[*'a* $\langle \Rightarrow \rangle$ *'b*, (*'b*) *set*] \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'b*) (- :-> - [65,66] 65)

trans-clos ::(*'a* $\langle \Rightarrow \rangle$ *'a*) \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'a*) (- %+ [1000] 999)
ref-trans-clos::(*'a* $\langle \Rightarrow \rangle$ *'a*) \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'a*) (- %* [1000] 999)

partial-func ::[*'a set*,*'b set*] \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'b*) *set* (- -|> - [54,53] 53)
total-func ::[*'a set*,*'b set*] \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'b*) *set* (- ----> - [54,53] 53)
partial-inj ::[*'a set*,*'b set*] \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'b*) *set* (- >-|> - [54,53] 53)
total-inj ::[*'a set*,*'b set*] \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'b*) *set* (- >--> - [54,53] 53)
partial-surj ::[*'a set*,*'b set*] \Rightarrow (*'a* $\langle \Rightarrow \rangle$ *'b*) *set* (- -|>> - [54,53] 53)

¹More precise, is build on top of Isabelle/HOL-Complex but only to support real numbers.
If you dont't need them, should be build just fine on top of a plain Isabelle/HOL.



B.1. Core of the Mathematical Toolkit

total-surj $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
biject $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
fin-part-func $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- || -> - [54,53] 53)$
fin-part-inj $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- > - || -> - [54,53] 53)$
rel $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
rel-appl $::['a \Leftrightarrow 'b, 'a] \Rightarrow 'b$ $(- \% ^ - [90,91] 90)$
override $::['a \Leftrightarrow 'b, 'a \Leftrightarrow 'b] \Rightarrow ('a \Leftrightarrow 'b)$ $(- (' +) - [55,56] 55)$
iter $::['a \Leftrightarrow 'a, \text{int}] \Rightarrow ('a \Leftrightarrow 'a)$
numb-range $::[\text{int}, \text{int}] \Rightarrow (\text{int}) \text{ set}$ $(- .. - [50,51] 50)$

syntax (*xsymbols*)

idZ $:: 'a \text{ set} \Rightarrow ('a \Leftrightarrow 'a)$ (id)
forw-comp $::['a \Leftrightarrow 'b, 'b \Leftrightarrow 'c] \Rightarrow ('a \Leftrightarrow 'c)$ $(- \circ - [60,61] 60)$
dom-restr $::['a \text{ set}, 'a \Leftrightarrow 'b] \Rightarrow ('a \Leftrightarrow 'b)$ $(- \triangleleft - [71,70] 70)$
ran-restr $::['a \Leftrightarrow 'b, ('b) \text{ set}] \Rightarrow 'a \Leftrightarrow 'b$ $(- \triangleright - [65,66] 65)$
dom-substr $::[('a) \text{ set}, 'a \Leftrightarrow 'b] \Rightarrow 'a \Leftrightarrow 'b$ $(- \triangleleft - [71,70] 70)$
ran-substr $::['a \Leftrightarrow 'b, ('b) \text{ set}] \Rightarrow ('a \Leftrightarrow 'b)$ $(- \triangleright - [65,66] 65)$

trans-clos $::('a \Leftrightarrow 'a) \Rightarrow ('a \Leftrightarrow 'a)$ $(- + [1000] 999)$
ref-trans-clos $::('a \Leftrightarrow 'a) \Rightarrow ('a \Leftrightarrow 'a)$ $(- * [1000] 999)$

partial-func $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
total-func $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \rightarrow - [54,53] 53)$
partial-inj $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
total-inj $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
partial-surj $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
total-surj $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
biject $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
override $::['a \Leftrightarrow 'b, 'a \Leftrightarrow 'b] \Rightarrow ('a \Leftrightarrow 'b)$ $(- \oplus - [55,56] 55)$
numb-range $::[\text{int}, \text{int}] \Rightarrow (\text{int}) \text{ set}$ $(- .. - [50,51] 50)$
fin-part-func $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
fin-part-inj $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \dashrightarrow - [54,53] 53)$
rel $::['a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Leftrightarrow 'b) \text{ set}$ $(- \leftrightarrow - [54,53] 53)$
rel-appl $::['a \Leftrightarrow 'b, 'a] \Rightarrow 'b$ $(- (. .) [90,91] 90)$

syntax

dom $::('a \Leftrightarrow 'b) \Rightarrow 'a \text{ set}$

Appendix B. Isabelle Theories

```

ran      ::('a <=> 'b) => 'b set
zinverse ::('a <=> 'b) => ('b <=> 'a)      (- %~ [1000]999)
rel-image ::('a <=> 'b) => ('a set) => 'b set (-'(|-')[1000,0]999)
back-comp ::['b <=> 'c, 'a <=> 'b] => ('a <=> 'c) (- %o - [60,61] 60)
prodZ    ::['a set, 'b set] => ('a <=> 'b)    (- %x - [81,80] 80)
sumZ     ::['a set, 'b set] => ('a + 'b) set  (- %+ - [66,65] 65)
gen-un   ::'a set set => 'a set
gen-int  ::'a set set => 'a set
zint     :: int => nat                        (§i)

```

syntax (*xsymbols*)

```

dom      ::('a <=> 'b) => 'a set             (dom)
ran      ::('a <=> 'b) => 'b set             (ran)
zinverse ::('a <=> 'b) => ('b <=> 'a)      (~ [1000]999)
rel-image ::('a <=> 'b) => ('a set) => 'b set (-(|-')[1000,0]999)
back-comp ::['b <=> 'c, 'a <=> 'b] => ('a <=> 'c) (- o - [60,61] 60)
prodZ    ::['a set, 'b set] => ('a <=> 'b)    (- x - [81,80] 80)
sumZ     ::['a set, 'b set] => ('a + 'b) set  (- + - [66,65] 65)
gen-un   ::'a set set => 'a set             (U -)
gen-int  ::'a set set => 'a set             (∩ -)

```

translations

```

dom r      == Domain r
ran r      == Range r
r %~       == converse r
rel-image r s == r+s
r %o s     == r O s
a %x b     == a <*> b
a %+ b     == a Plus b
gen-un s   == Union s
gen-int s  == Inter s
zint       == nat

```

consts

```

zsize     ::('a set) => int                 (#)
String    ::string set
Integers  ::int set                         (%Z)
Naturals  ::int set                         (%N)
Naturals-1 ::int set
<->      ::[bool, bool] => bool            (infixr 25)
maplet    ::['a, 'b] => ('a * 'b)          ((2- |-->| -) [60,61] 60)
disjoint  ::('a <=> ('b set)) => bool

```

B.1. Core of the Mathematical Toolkit

partition ::(['a <=> ('b set)), ('b set)] => bool (**infixl** 60)

syntax(*xsymbols*)

zsize ::('a set) => int (#)
String ::string set (String)
Integers ::int set (\mathbb{Z})
Naturals ::int set (\mathbb{N})
Naturals-1 ::int set (\mathbb{N}_1)
maplet ::['a, 'b] => ('a * 'b) ((2- ↦/ -) [60,61] 60)
disjoint ::('a <=> ('b set)) => bool (disjoint)
partition ::(['a <=> ('b set)), ('b set)] => bool (- partition - [60,61]60)

consts *Fin* :: 'a set => 'a set set (%F)

syntax (*xsymbols*)

Fin :: 'a set => 'a set set (\mathbb{F})

inductive *Fin*(*A*)

intros

emptyI : {} : *Fin*(*A*)
insertI: [a: *A*; b: *Fin*(*A*)] ==> *insert a b* : *Fin*(*A*)

defs

idZ-def: $idZ\ X \quad == \{p. \ ?\ x:X . p = (x,x)\}$
rel-def: $A \langle\langle\langle\ \rangle\ \rangle\ \rangle B \quad == Pow\ (A\ \%x\ B)$
pfun-def: $S \dashv\ \dashv\ \dashv\ R \quad == \{f. f:(S \langle\langle\langle\ \rangle\ \rangle\ \rangle R) \ \& \ (!\ x\ y1\ y2. (x,y1):f \ \& \ (x,y2):f \ \dashv\ \dashv\ \dashv\ (y1=y2))\}$

forw-comp-def: $r \% ;\ s \quad == \{(x,z). \ ?\ y. (x,y):r \ \& \ (y,z):s \}$
dom-restr-def: $s <:\ r \quad == \{(x,y). (x,y) : r \ \& \ x : s \}$
ran-restr-def: $r >:\ s \quad == \{(x,y). (x,y) : r \ \& \ y : s \}$
dom-substr-def: $s <-\ r \quad == \{(x,y). (x,y) : r \ \& \ x \sim : s \}$
ran-substr-def: $r >-\ s \quad == \{(x,y). (x,y) : r \ \& \ y \sim : s \}$

defs

ref-trans-clos-def: $r \% * \quad == lfp\ (\% s. idZ\ (dom\ r\ Un\ ran\ r)\ Un\ (r\ \% o\ s))$
trans-clos-def: $r \% + \quad == r\ \% o\ r \% *$

Appendix B. Isabelle Theories

```

tfun-def:    S ----> R  == {s. s : S -|-> R & dom s = S}
partial-inj-def: S >-|-> R == {s. s : S -|-> R &
                               (ALL x1 x2 y. (x1,y) : s & (x2,y) : s
                               --> x1 = x2)}
total-inj-def:  S >--> R  == (S >-|-> R) Int (S ----> R)
partial-surj-def: S -|->> R == {s. s : S -|-> R & ran s = R}
total-surj-def:  S -->> R  == (S -|->> R) Int (S ----> R)
biject-def:     S >-->> R == (S -->> R) Int (S >--> R)
override-def:   S (+) R   == (dom R <-: S) Un R

String-def:     String    == UNIV
Integers-def:   Integers  == {a::int. True}
Naturals-def:   Naturals  == {a::int. 0 <= a}
Naturals-1-def: Naturals-1 == {a::int. 1 <= a}

iff-def:        A <-> B   == A = B

zsize-def:      # S       == int (card S)

iter-def:       iter R n  == if (n:Naturals)
                               then (nat-rec (idZ(dom R))
                                       (%n it. R %o it)
                                       ($i n))
                               else (nat-rec (idZ(dom (R %~)))
                                       (%n it. (R %~) %o it))
                                       ($i ( n)))

numb-range-def: (a .. b)  == { k. a <= k & k <= b}
fin-part-func-def: S -||-> R == {s. s : S -|-> R & dom(s) : Fin(S)}
fin-part-inj-def:  S >-||-> R == (S >-|-> R) Int (S -||-> R)
rel-apply-def:    R %^ x   == ( • y. (x,y) : R)
maplet-def:       maplet a b == (a,b)
disjoint-def:     disjoint f == (! p:f. ! q:f. p ~ = q ---->
                               (snd p) Int (snd q) = {})
partition-def:    f partition a == (disjoint f & gen-un(ran f) = a)

```

constdefs

B.1. Core of the Mathematical Toolkit

$\text{Lambda} \quad :: [\text{'a set}, \text{'a} \Rightarrow \text{'b}] \Rightarrow \text{'a} \Leftrightarrow \text{'b}$
 $\text{Lambda } A f \quad == \{(x,y). x : A \ \& \ y = f x\}$
 $\text{Mu} \quad :: [\text{'a set}, \text{'a} \Rightarrow \text{bool}] \Rightarrow \text{'a}$
 $\text{Mu } A P \quad == \bullet x. x : A \ \& \ P x$

syntax

$*\text{Lambda} \quad :: [\text{pttrn}, \text{'a set}, \text{'a} \Rightarrow \text{'b}] \Rightarrow \text{'a} \Leftrightarrow \text{'b} \ ((\text{slambda } (-):(-). / -) [0,0,10]10)$
 $*\text{Mu} \quad :: [\text{pttrn}, \text{'a set}, \text{'a} \Rightarrow \text{bool}] \Rightarrow \text{'a} \quad ((\text{smu } (-):(-). / -) [0,0,10]10)$

translations

$\text{lambda } x:A. E \quad == \text{Lambda } A \ (\% x. E)$
 $\text{mu } x:A. P \quad == \text{Mu } A \ (\% x. P)$

syntax

- $\text{powset} \quad :: (\text{'a set}) \Rightarrow (\text{'a set})\text{set} \quad (\%P)$
- $\text{emptyset} \quad :: (\text{'a set}) \Rightarrow (\text{'a set}) \quad (\text{emptyset- } 10)$

syntax (*xsymbols*)

- $\text{powset} \quad :: (\text{'a set}) \Rightarrow (\text{'a set})\text{set} \quad (\mathbb{P})$
- $\text{emptyset} \quad :: (\text{'a set}) \Rightarrow (\text{'a set}) \quad (\emptyset- 10)$

translations

$\%P A \quad == \text{Pow } A$
 $\text{emptyset } a \Rightarrow \{\}$

Lambda Lemmas

lemma *Lambda-total*: $\text{Lambda } A f : (A \dashrightarrow (\text{range } f))$
<proof>

lemma *Lambda-dom*: $\text{dom}(\text{Lambda } A f) = A$
<proof>

lemma *Lambda-ran*: $\text{ran}(\text{Lambda } A f) \leq \text{range } f$

Appendix B. Isabelle Theories

$\langle proof \rangle$

lemma *Lambda-beta*: $!! A. a : A ==> ((\text{Lambda } A f) \% ^ a) = f a$
 $\langle proof \rangle$

lemma *Mu-equality*:

$[[a : A ; P a; ! x:A. P x --> x = a]] ==> (\text{Mu } A P) = a$
 $\langle proof \rangle$

declare *Lambda-total*[simp] *Lambda-beta*[simp] *Lambda-ran*[simp]
Mu-equality[simp] *Lambda-dom*[simp]

Set Lemmas

lemma *Int-empty*[simp]: $A \text{ Int } B = \{\} ==> A - B = A$
 $\langle proof \rangle$

lemma *subset-union-mono*[simp]: $A <= B ==> A <= B \text{ Un } C$
 $\langle proof \rangle$

lemma *union-subset-distr*[simp]: $(A \text{ Un } B <= C) = (B <= C \ \& \ A <= C)$
 $\langle proof \rangle$

Simpler Definitions

lemma *idZ-simp*: $\text{idZ } X = \{(x,x) \mid x. x:X\}$
 $\langle proof \rangle$

lemma *inverse-simp*: $R \% ^\sim = \{(y,x) \mid x y. (x,y):R\}$
 $\langle proof \rangle$

lemma *dom-simp*: $\text{dom } R = \{x. ? y. (x,y):R\}$
 $\langle proof \rangle$

lemma *ran-simp*: $\text{ran } R = \{y. ? x. (x,y):R\}$
 $\langle proof \rangle$

lemma *idZ-eqI*: $!! A. [[a=b; a:A]] ==> (a,b) : \text{idZ}(A)$
 $\langle proof \rangle$

lemmas *idZI = refl* [THEN *idZ-eqI*]

B.1. Core of the Mathematical Toolkit

lemma *idZE*:
assumes *p1*: $c : \text{idZ}(A)$
assumes *p2*: $\llbracket x:A; c = (x,x) \rrbracket \implies P$
shows P
<proof>

declare *idZI* [*intro!*]
declare *idZE* [*elim!*]

lemma *idZ-iff*: $((x,y) : \text{idZ } A) = (x=y \ \& \ x : A)$
<proof>

lemma *idZ-subset-Times*: $\text{idZ}(A) \leq A \lt * \gt A$
<proof>

Domain and Range Properties

lemma *Dom-In*: $x:\text{dom } R = (? y. y:\text{ran}(R) \ \& \ (x,y):R)$
<proof>

lemma *Ran-In*: $y:\text{ran } R = (? x. x:\text{dom } R \ \& \ (x,y):R)$
<proof>

lemma *Dom-Union[simp]*: $\text{dom}(Q \text{ Un } R) = \text{dom}(Q) \text{ Un } \text{dom}(R)$
<proof>

lemma *Ran-Union[simp]*: $\text{ran}(Q \text{ Un } R) = \text{ran}(Q) \text{ Un } \text{ran}(R)$
<proof>

lemma *Dom-Inter[simp]*: $\text{dom}(Q \text{ Int } R) \leq \text{dom}(Q) \text{ Int } \text{dom}(R)$
<proof>

lemma *Ran-Inter[simp]*: $\text{ran}(Q \text{ Int } R) \leq \text{ran}(Q) \text{ Int } \text{ran}(R)$
<proof>

lemma *Dom-Empty[simp]*: $\text{dom } \{\} = \{\}$
<proof>

lemma *Ran-Empty [simp]*: $\text{ran } \{\} = \{\}$
<proof>

lemma *Dom-Insert0*: $(\text{dom } (\text{insert } (x,y) S)) = \text{insert } x (\text{dom } S)$

Appendix B. Isabelle Theories

<proof>

lemma *Dom-Insert* [simp]: $\text{dom } (\text{insert } p \ A) = (\text{insert } (\text{fst } p) \ (\text{dom } A))$
<proof>

lemma *Ran-Insert0*: $(\text{ran } (\text{insert } (x,y) \ S)) = \text{insert } y \ (\text{ran } S)$
<proof>

lemma *Ran-Insert*[simp]: $(\text{ran } (\text{insert } p \ S)) = \text{insert } (\text{snd } p) \ (\text{ran } S)$
<proof>

lemma *pair-rel-dom* [simp]: $(a,b):r \implies (a : (\text{dom } r))$
<proof>

lemma *pair-rel-dom-fst* [simp]: $a:r \implies ((\text{fst } a) : (\text{dom } r))$
<proof>

lemma *pair-rel-ran* [simp]: $(a,b):r \implies b : \text{ran } r$
<proof>

lemma *pair-rel-dom-snd* [simp]: $a:r \implies \text{snd } a : \text{ran } r$
<proof>

lemma *dom-implies-ran* : $a : \text{dom } r \implies (\text{EX } b. (a,b) : r)$
<proof>

lemma *Dom-Rel-Empty* [simp]: $(\text{dom } f = \{\}) = (f = \{\})$
<proof>

lemma *Ran-Rel-Empty* [simp]: $(\text{ran } f = \{\}) = (f = \{\})$
<proof>

Domain and Range Restriction

lemma *dom-restrI* [rule-format]: $p : r \longrightarrow \text{fst } p : s \longrightarrow p : (s <: r)$
<proof>

declare *dom-restrI* [intro!]

B.1. Core of the Mathematical Toolkit

lemma *dom-restrD1*: $p : (s <: r) \implies p : r$
<proof>

lemma *dom-restrD2*: $p : (s <: r) \implies \text{fst } p : s$
<proof>

lemma *dom-restrE2*:
assumes *p1*: $p : (s <: r)$
assumes *p2*: $[[p : r; \text{fst } p : s]] \implies P$
shows P
<proof>

lemma *dom-restrE* :
assumes *p1*: $p : (s <: r)$
assumes *p2*: $!! a b. [[p = (a, b); (a, b) : r; a : s]] \implies P$
shows P
<proof>
declare *dom-restrE* [*elim!*]

lemma *ran-restrI* [*rule-format*]: $p : r \dashrightarrow \text{snd } p : s \dashrightarrow p : (r >: s)$
<proof>
declare *ran-restrI* [*intro!*]

lemma *ran-restrD1*: $p : (r >: s) \implies p : r$
<proof>

lemma *ran-restrD2*: $!! r. p : (r >: s) \implies \text{snd } p : s$
<proof>

lemma *ran-restrE2*:
assumes *p1*: $p : (r >: s)$
assumes *p2*: $[[p : r; \text{snd } p : s]] \implies P$
shows P
<proof>

lemma *ran-restrE*:
assumes *p1*: $p : (r >: s)$
assumes *p2*: $!! a b. [[p = (a, b); (a, b) : r; b : s]] \implies P$
shows P
<proof>
declare *ran-restrE* [*elim!*]

Appendix B. Isabelle Theories

lemma *Dom-Restrict* [simp]: $\text{dom } (S <: R) = S \text{ Int } (\text{dom } R)$
⟨proof⟩

lemma *Ran-Restrict* [simp]: $\text{ran } (S > R) = \text{ran}(S) \text{ Int } R$
⟨proof⟩

lemma *Dom-Rest-Subset* [simp]: $(S <: R) <= R$
⟨proof⟩

lemma *Ran-Rest-Subset* [simp]: $(S > R) <= S$
⟨proof⟩

lemma *Dom-Restr-mt*[simp]: $X <: \{\} = \{\}$
⟨proof⟩

lemma *Dom-Ran-Restr-Comp* : $((S <: R) > T) = (S <: (R > T))$
⟨proof⟩

lemma *Dom-Dom-Restr-Comp* [simp]: $(S <: (R <: T)) = ((S \text{ Int } R) <: T)$
⟨proof⟩

lemma *Ran-Ran-Restr-Comp* [simp]: $((S > R) > T) = (S > (R \text{ Int } T))$
⟨proof⟩

lemma *No-Dom-Restr* [simp]: $(\text{dom } R <= S) = (S <: R = R)$

⟨proof⟩

lemma *Empty-Dom-Restr* [simp]: $\{\} <: R = \{\}$
⟨proof⟩

lemma *No-Ran-Restr* [simp]: $(\text{ran } R <= S) = (R > S = R)$

⟨proof⟩

lemma *Empty-Ran-Restr* [simp]: $R > \{\} = \{\}$
⟨proof⟩

lemma *UNIV-Ran-Restr* [simp]: $x > \text{UNIV} = x$

B.1. Core of the Mathematical Toolkit

<proof>

lemma *dom-restr-single* [*simp*]: $!!R. \text{fst } p : R \implies p : R \text{ <: } f = (p:f)$
<proof>

Domain and Range Anti-Restriction

lemma *dom-substrI* [*rule-format (no-asm)*]: $p : r \dashrightarrow \text{fst } p \sim : s \dashrightarrow p : (s \text{ <-: } r)$
<proof>
declare *dom-substrI* [*intro!*]

lemma *dom-substrD1*: $!!r. p : (s \text{ <-: } r) \implies p : r$
<proof>

lemma *dom-substrD2*:
 $!!r. p : (s \text{ <-: } r) \implies \text{fst } p \sim : s$
<proof>

lemma *dom-substrE2*:
assumes *p1*: $p : (s \text{ <-: } r)$
assumes *p2*: $[[p : r; \text{fst } p \sim : s]] \implies P$
shows *P*
<proof>

lemma *dom-substrE*:
assumes *p1*: $p : (s \text{ <-: } r)$
assumes *p2*: $!! a b. [[p = (a, b); (a,b) : r; a \sim : s]] \implies P$
shows *P*
<proof>
declare *dom-substrE* [*elim!*]

lemma *ran-substrI* [*rule-format (no-asm)*]: $p : r \dashrightarrow \text{snd } p \sim : s \dashrightarrow p : (r \text{ :-> } s)$
<proof>
declare *ran-substrI* [*intro!*]

lemma *ran-substrD1*: $!!r. p : (r \text{ :-> } s) \implies p : r$
<proof>

Appendix B. Isabelle Theories

lemma *ran-substrD2*:!!*r*. $p : (r \text{ :-> } s) \implies \text{snd } p \sim :s$
 <proof>

lemma *ran-substrE2*:
assumes *p1*: $p : (r \text{ :-> } s)$
assumes *p2*: $[[p : r; \text{snd } p \sim :s]] \implies P$
shows *P*
 <proof>

lemma *ran-substrE*:
assumes *p1*: $p : (r \text{ :-> } s)$
assumes *p2*: !! *a b*. $[[p = (a, b); (a, b) : r; b \sim :s]] \implies P$
shows *P*
 <proof>
declare *ran-substrE* [*elim!*]

lemma *Dom-Anti-Restr* : $(S <-: R) = (\text{dom}(R) - S) <: R$
 <proof>

lemma *dom-substr-Un-distribL* [*simp*]: $(S <-: (A \text{ Un } B)) = ((S <-: A) \text{ Un } (S <-: B))$
 <proof>

lemma *dom-substr-Un-asso* [*simp*]: $(A <-: (B <-: C)) = ((A \text{ Un } B) <-: C)$
 <proof>

lemma *Ran-Anti-Restr* [*simp*]: $(R \text{ :-> } T) = R \text{ :> } (\text{ran}(R) - T)$
 <proof>

lemma *DomRestr-Un-DomSubstr* [*simp*]: $(S <: R) \text{ Un } (S <-: R) = R$
 <proof>

lemma *RanRestr-Un-RanSubstr* [*simp*]: $(S \text{ :> } R) \text{ Un } (S \text{ :-> } R) = S$
 <proof>

lemma *dom-anti-restr-single* [*simp*]: !! *R*. $\text{fst } p \sim : R \implies p : R <-: f = (p : f)$
 <proof>

lemma *dom-substr-empty* [*simp*]: $(\text{dom } r <-: r) = \{\}$
 <proof>

B.1. Core of the Mathematical Toolkit

lemma *equals0E*:

assumes *p1*: $\text{Collect } P = \{\}$

assumes *p2*: $\sim P x \implies Q$

shows Q

<proof>

lemma *dom-substr-subset* : $((m <-: r) = \{\}) = ((\text{dom } r) \leq m)$

<proof>

lemma *dom-substr-subset-imp* [*simp*]: $!! r. ((\text{dom } r) \leq m) \implies ((m <-: r) = \{\})$

<proof>

Relational Distributivity

lemma *Rel-Union-Distr* [*simp*]: $(r : (X <--> Y) \ \& \ s : (X <--> Y)) = (r \ \text{Un} \ s : (X <--> Y))$

<proof>

Relational Composition

lemma *forw-comp-simp* [*simp*]: $(r \% ; s) = (s \% o r)$

<proof>

lemma *Compose-Rel* [*simp*]: $(x,z):(R \% ; Q) = (\exists y. (x,y):R \ \& \ (y,z):Q)$

<proof>

lemma *Compose-Transitive* : $(P \% ; (Q \% ; R)) = ((P \% ; Q) \% ; R)$

<proof>

lemma *Compose-Id-Left* : $((\text{idZ } X) \% ; P) = (X <: P)$

<proof>

lemma *Compose-Id-Right* [*simp*]: $(P \% ; \text{idZ } X) = (P :> X)$

<proof>

lemma *Compose-Refl-Trans* [*simp*]: $!! R. (R \% ; R) \leq R \implies ((R \ \text{Un} \ \text{idZ}(S)) \% ; (R \ \text{Un} \ \text{idZ}(S))) \leq (R \ \text{Un} \ \text{idZ}(S))$

Appendix B. Isabelle Theories

$\langle proof \rangle$

Relational Inversion

lemma *Inversion* [simp]: $(y,x):(R \% \sim) = ((x,y):R)$
 $\langle proof \rangle$

lemma *Inverse-Comp* [simp]: $((Q \% ; R) \% \sim) = ((R \% \sim) \% ; (Q \% \sim))$
 $\langle proof \rangle$

lemma *Inverse-Idem* [simp]: $((R \% \sim) \% \sim) = R$
 $\langle proof \rangle$

lemma *Inverse-Id* [simp]: $((idZ X) \% \sim) = idZ X$
 $\langle proof \rangle$

lemma *Inverse-Dom* [simp]: $dom(R \% \sim) = ran (R)$
 $\langle proof \rangle$

lemma *Inverse-Ran* [simp]: $ran(R \% \sim) = dom (R)$
 $\langle proof \rangle$

lemma *Dom-Anti-Restr-Int* [simp]: $((dom g) <-: f) Int g = \{\}$
 $\langle proof \rangle$

lemma *Dom-Dom-Anti-Restr-Int* [simp]: $((dom ((dom g) <-: f)) Int (dom g)) = \{\}$
 $\langle proof \rangle$

lemma *Rel-Dom-Restr* [simp]: $!!f. [[f:(A <--> B); s: (Pow A)]] ==> ((s <: f) : (A <--> B))$
 $\langle proof \rangle$

Relational Image

lemma *Image-Rel* : $y:(R(|S|)) = (? x. \{x\} <= S \ \& \ (x,y):R)$
 $\langle proof \rangle$

lemma *Image-ran* : $R(|S|) = ran (S <: R)$
 $\langle proof \rangle$

lemma *Image-dom-comp* : $dom(Q \% ; R) = (Q \% \sim) (|dom (R)|)$
 $\langle proof \rangle$

B.1. Core of the Mathematical Toolkit

lemma *Image-ran-comp* : $\text{ran}(Q \% ; R) = R(|\text{ran}(Q)|)$
 $\langle \text{proof} \rangle$

lemma *Image-union* [simp]: $R(|(S \text{ Un } T)|) = (R(|S|)) \text{ Un } (R(|T|))$
 $\langle \text{proof} \rangle$
declare *Image-union* [simp]

lemma *Image-inter* [simp]: $R(|(S \text{ Int } T)|) \leq (R(|S|)) \text{ Int } (R(|T|))$
 $\langle \text{proof} \rangle$

lemma *Image-dom* [simp]: $R(|\text{dom}(R)|) = \text{ran}(R)$
 $\langle \text{proof} \rangle$

Overriding

lemma *overrideI1*: $!!p. p : \text{dom } s <-: r \implies p : (r (+) s)$
 $\langle \text{proof} \rangle$

lemma *overrideI2*: $!!p. p : s \implies p : (r (+) s)$
 $\langle \text{proof} \rangle$

lemma *overrideCI*[intro!]:
assumes *prem*: $(p \sim: s \implies p : (\text{dom } s) <-: r)$
shows $p : (r (+) s)$
 $\langle \text{proof} \rangle$

lemma *overrideE*[elim!]: $\llbracket p:(r (+) s); p : (\text{dom } s) <-: r \implies Q; p: s \implies Q \rrbracket \implies Q$
 $\langle \text{proof} \rangle$

lemma *override-mt-left* [simp]: $(\{\}) (+) R = R$
 $\langle \text{proof} \rangle$

lemma *override-mt-right* [simp]: $R (+) \{\} = R$
 $\langle \text{proof} \rangle$

lemma *override-idem* [simp]: $(R(+))R = R$
 $\langle \text{proof} \rangle$

lemma *override-Domain* [simp]: $\text{dom}(R(+))Q = (\text{dom}(Q)) \text{ Un } (\text{dom}(R))$

Appendix B. Isabelle Theories

\langle proof \rangle

lemma *override-assoc* : $(P(+)(Q(+)R)) = ((P(+)Q)(+)R)$

\langle proof \rangle

lemma *override-left-idem* [simp]: $(A (+) (A (+) B)) = (A (+) B)$

\langle proof \rangle

lemma *override-inter* [simp]: $((\text{dom}(Q) \text{ Int } \text{dom}(R)) = \{\}) \implies ((Q(+)R) = (Q \text{ Un } R))$

\langle proof \rangle

lemma *override-res-left* [simp]: $(V <: (Q(+)R)) = ((V <: Q) (+) (V <: R))$

\langle proof \rangle

lemma *override-res-right* [simp]: $((Q(+)R) :> W) <= ((Q :> W) (+) (R :> W))$

\langle proof \rangle

lemma *override-single* [simp]: $\{(x,y)\} (+) \{(x,z)\} = \{(x,z)\}$

\langle proof \rangle

Closure

lemmas *zrtrancl-def* = *ref-trans-clos-def*

lemma *zrtrancl-fun-mono* : $\text{mono}(\%s. \text{idZ}(\text{dom } r \text{ Un } \text{ran } r) \text{ Un } (r \% o s))$

\langle proof \rangle

lemmas *zrtrancl-unfold* = *zrtrancl-fun-mono* [THEN *zrtrancl-def* [THEN *def-lfp-unfold*]]

lemma *zrtrancl-refl0* : $!!x. [\![x : \text{dom } r \text{ Un } \text{ran } r]\!] \implies (x,x) : (r \%*)$

\langle proof \rangle

lemmas *zrtrancl-refl* = *UnCI* [THEN *zrtrancl-refl0*]

declare *zrtrancl-refl* [simp]

declare *zrtrancl-refl* [intro!]

lemma *zrtrancl-into-zrtrancl* : $!!r. [\![(a,b) : r\%*; (b,c) : r]\!] \implies (a,c) : r\%*$

\langle proof \rangle

B.1. Core of the Mathematical Toolkit

lemma *r-into-zrtrancl* [*intro*]: $!!p. p : r \implies p : r^{\%*}$
<proof>

lemma *zrtrancl-mono*: $!!r. r \leq s \implies r^{\%*} \leq s^{\%*}$
<proof>

lemma *zrtrancl-full-induct*:
 $[[(a,b) : r^{\%*}; !!x. x : \text{dom } r \implies P(x,x); !!y. y : \text{ran } r \implies P(y,y); !!x y z. [P(x,y); (x,y) : r^{\%*}; (y,z) : r] \implies P(x,z)] \implies P(a,b)$
<proof>

lemma *zrtrancl-full-induct2*:
assumes *p1*: $(a,b) : r^{\%*}$
assumes *p2*: $!!x. x : \text{dom } r \implies P x x$
assumes *p3*: $!!y. y : \text{ran } r \implies P y y$
assumes *p4*: $!!x y z. [P x y; (x,y) : r^{\%*}; (y,z) : r] \implies P x z$
shows $P a b$
<proof>

lemma *zrtrancl-induct*:
 $[[(a::'a,b) : r^{\%*}; a : \text{dom } r \implies P(a); a : \text{ran } r \implies P(a); !!y z. [(a,y) : r^{\%*}; (y,z) : r; P(y)] \implies P(z)] \implies P(b)$
<proof>

lemma *trans-zrtrancl*: $\text{trans}(r^{\%*})$
<proof>

Appendix B. Isabelle Theories

lemmas *zrtrancl-trans* = *trans-zrtrancl* [THEN *transD*, *standard*]

lemma *zrtrancl-dom1* [*simp*]: $!!r. [(a,b) : (r\%*); a\sim : \text{ran } r] \implies a : \text{dom } r$
 <proof>

lemma *zrtrancl-ran0* : $!!r. (a,b) : (r\%*) \implies b : \text{dom } r \cup \text{ran } r$
 <proof>

lemma *zrtrancl-ran1* [*simp*]: $!!r. [(a,b) : (r\%*); b\sim : \text{ran } r] \implies b : \text{dom } r$
 <proof>

lemma *zrtranclE0*: $[(a,b) : r\%*;$
 $[(a,b) : (\text{idZ } (\text{dom } r \cup \text{ran } r))]] \implies P;$
 $!!y. [(a,y) : r\%*; (y,b) : r] \implies P$
 $] \implies P$
 <proof>

lemma *zrtranclE*:
assumes *p1*: $(a,b) : r\%*$
assumes *p2*: $[a : \text{dom } r; a = b] \implies P$
assumes *p3*: $[a : \text{ran } r; a = b] \implies P$
assumes *p4*: $!!y. [(a,y) : r\%*; (y,b) : r] \implies P$
shows *P*
 <proof>

lemmas *zrtrancl-into-zrtrancl2* = *r-into-zrtrancl* [THEN *zrtrancl-trans*, *standard*]

lemma *reach-induct*:
assumes *p1*: $q : (r\%*) (| Q |)$
assumes *p2*: $!!x. x : Q \implies P x$
assumes *p3*: $!!y z. [(y : (r\%*) (| Q |)); (y, z) : r; P y] \implies P z$
shows $P q$
 <proof>

lemma *dom-zrtrancl* [*simp*]: $\text{dom } (r\%*) = (\text{dom } r \cup \text{ran } r)$
 <proof>

B.1. Core of the Mathematical Toolkit

lemma *zrtrancl-ran-eq-dom* [simp]: $!!r . \text{ran}(r\%_*) = \text{dom}(r\%_*)$
 ⟨proof⟩

lemma *dom-zrtrancl* : $\text{ran}(r\%_*) = (\text{dom } r \text{ Un } \text{ran } r)$
 ⟨proof⟩

lemma *zrtrancl-idemp* [simp]: $(r\%_*)\%_* = r\%_*$
 ⟨proof⟩

lemma *zrtrancl-idemp-self-comp* [simp]: $R\%_* \circ R\%_* = R\%_*$
 ⟨proof⟩

lemma *zrtrancl-subset-zrtrancl* : $!!r . r \leq s\%_* \implies r\%_* \leq s\%_*$
 ⟨proof⟩

lemma *zrtrancl-subset* : $!!R . [R \leq S; S \leq R\%_*] \implies S\%_* = R\%_*$
 ⟨proof⟩

lemma *zrtrancl-Un-zrtrancl* : $(R\%_* \text{ Un } S\%_*)\%_* = (R \text{ Un } S)\%_*$
 ⟨proof⟩

lemma *zrtrancl-converseD* : $!!r.(x,y) : (r\%_{\sim})\%_* \implies (y,x) : r\%_*$
 ⟨proof⟩

lemma *zrtrancl-converseI* : $!!r.(y,x) : r\%_* \implies (x,y) : (r^{-1})\%_*$
 ⟨proof⟩

lemma *zrtrancl-converse* : $((r\%_{\sim})\%_*) = ((r\%_*)\%_{\sim})$

⟨proof⟩

lemma *converse-zrtrancl-induct*:

assumes *major*: $(a,b) : r\%_*$

assumes *prem1*: $b : \text{dom } r \implies P(b)$

assumes *prem2*: $b : \text{ran } r \implies P(b)$

assumes *prem3*: $!!y z. [(y,z) : r; (z,b) : r\%_*; P(z)] \implies P(y)$

shows $P(a)$

⟨proof⟩

lemmas *converse-zrtrancl-induct2* = *converse-zrtrancl-induct* [where $a = (ax, ay)$
 and $b = (bx, by)$]

Appendix B. Isabelle Theories

lemma *converse-zrtranclE0*: $\llbracket (a,b) : r\%_*; (a,b) : (\text{idZ } (\text{dom } r \text{ Un } \text{ran } r)) \implies P; \exists y. \llbracket (a,y) : r; (y,b) : r\%_* \rrbracket \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *converse-zrtranclE*:

assumes *p1*: $(x,z) : r\%_*$

assumes *p2*: $\llbracket z : (\text{dom } r); z = x \rrbracket \implies P$

assumes *p3*: $\llbracket z : (\text{ran } r); z = x \rrbracket \implies P$

assumes *p4*: $\exists y. \llbracket (x,y) : r; (y,z) : r\%_* \rrbracket \implies P$

shows *P*

$\langle \text{proof} \rangle$

lemmas *converse-zrtranclE2* = *converse-zrtranclE* [**where** $x = (x_a, x_b)$ **and** $z = (z_a, z_b)$]

lemma *r-comp-zrtrancl-eq* : $r \circ r\%_* = r\%_* \circ r$

$\langle \text{proof} \rangle$

+

lemmas *ztrancl-def* = *trans-clos-def*

lemma *ztrancl-mono*: $\exists r. \llbracket p : r\%_+; r \leq s \rrbracket \implies p : s\%_+$

$\langle \text{proof} \rangle$

lemma *ztrancl-into-zrtrancl*: $\exists p. p : r\%_+ \implies p : r\%_*$

$\langle \text{proof} \rangle$

lemma *r-into-ztrancl*[*intro*]: $\exists p. p : r \implies p : r\%_+$

$\langle \text{proof} \rangle$

lemma *zrtrancl-into-ztrancl1*: $\exists r. \llbracket (a,b) : r\%_*; (b,c) : r \rrbracket \implies (a,c) : r\%_+$

$\langle \text{proof} \rangle$

lemma *zrtrancl-into-ztrancl2* : $\exists r. \llbracket (a,b) : r; (b,c) : r\%_* \rrbracket \implies (a,c) : r\%_+$

$\langle \text{proof} \rangle$

Appendix B. Isabelle Theories

lemma *ztrancl-converseI* : !! r. (x,y) : (r%+)⁻¹ ==> (x,y) : (r⁻¹)%+
 <proof>

lemma *ztrancl-converseD* : !!r. (x,y) : (r⁻¹)%+ ==> (x,y) : (r%+)⁻¹
 <proof>

lemma *converse-ztrancl-induct*:

assumes *major*: (a,b) : r%+

assumes *prem1*: !!y. (y,b) : r ==> P(y)

assumes *prem2*: !!y z. [(y,z) : r; (z,b) : r%+; P(z)] ==> P(y)

shows P(a)

<proof>

lemma *ztranclD* : !!R. (x,y):R%+ ==> ? z. (x,z):R & (z,y):R%*
 <proof>

lemma *lemma1*: !!r. [(a,b) : r%*; r <= A %x A] ==> a=b | a:A
 <proof>

lemma *ztrancl-subset-Sigma*: !!r. r <= A %x A ==> r%+ <= A %x A
 <proof>

lemma *Closure-Trans-Inclusion* [*simp*]: R <= (R%+)
 <proof>

lemma *Closure-Trans-Compose* [*simp*]:
 ((R%+) %o (R%+)) <= (R%+)
 <proof>

lemma *Closure-Transitive* :

!!R. [(R<=Q & Q %o Q <= Q)] ==> R%+ <= Q
 <proof>

lemma *Closure-Ref-Inclusion* [*simp*]: idZ (dom R) <= R%*
 <proof>

lemma *Closure-RefTrans-Inclus* [*simp*]: R <= R%*
 <proof>

B.2. Integers in the Mathematical Toolkit

lemma *Closure-Trans-RefTrans-Inclus* [simp]: $R\%+ \leq R\%*$
(proof)

lemma *Closure-RefTransitive* :
!!R. [| (idZ (dom R Un ran R) <= Q) ; (R<=Q) ; ((Q %o Q) <= Q) |] ==>
(R%*) <= Q
(proof)

lemma *Closure-RefTrans-Id* : $(R\%*) = ((R\%+) \text{ Un } (\text{idZ } (\text{dom } R \text{ Un } \text{ran } R)))$
(proof)

lemma *Closure-RefTrans-Id-Union* : $R\%* = (R \text{ Un } \text{idZ } (\text{dom } R \text{ Un } \text{ran } R))\%+$
(proof)

lemma *isString* [simp]: $x:\text{String}$
(proof)

end

B.2. Integers in the Mathematical Toolkit

theory *ZInteg* imports *ZMathTool* begin

constdefs

zpred :: $\text{int} \Rightarrow \text{int}$
 $\text{zpred}(Z) == Z - 1$

zsuc :: $\text{int} \Rightarrow \text{int}$
 $\text{zsuc}(Z) == Z + 1$

Appendix B. Isabelle Theories

znat and zint

lemma *znat-bin-0*: $(\text{int } 0) = 0$
<proof>

lemma *zint-inverse[simp]*: $\$i(\text{int } i) = i$
<proof>

lemma *zless-def-Suc*: $(n < m) = (? x. m = n + \text{int } (\text{Suc } x))$
<proof>

lemma *zle-imp-zless-or-eq*: $z \leq w \implies z < w \mid z = (w :: \text{int})$
<proof>

lemma *znat-inverse*: $(b \in \%N) \implies (b = \text{int } (\$i b))$
<proof>

lemma *iter-0[simp]*: $(\text{iter } R (\text{int } 0)) = (\text{idZ } (\text{dom } R))$
<proof>

lemma *znat-pred[simp]*: $!!n. n \sim 0 \implies (\text{int } (n - 1)) = \text{zpred } (\text{int } n)$
<proof>

lemma *znat-Suc-not-zint-conc-Zero[simp]*: $\text{Suc } n \sim \$i 0$
<proof>

lemma *inj-zint[simp]*: $[[n: \%N; m: \%N]] \implies ((\$i n) = (\$i m)) = (n = m)$
<proof>

lemma *in-naturals*: $(0 \leq n) = (n : \%N)$
<proof>

lemma *zero-is-natural*: $0 : \%N$
<proof>

lemma *one-is-natural*: $1 : \%N$
<proof>

lemma *two-is-natural*: $2 : \%N$
<proof>

B.2. Integers in the Mathematical Toolkit

declare *zero-is-natural* [simp] *one-is-natural* [simp] *two-is-natural* [simp]

lemma *in-naturals2*[simp]: $!!n. 0 < n \implies (n : \%N)$
<proof>

lemma *zpred-zint*: $!!n. 0 < n \implies ((\$i\ n) - 1) = (\$i\ (zpred\ n))$
<proof>

inequalities

lemma *zless-not-refl3*: $((z::int) < w) \implies z \not\sim w$
<proof>

lemma *less-zsuc-eq-le* [simp]: $(a < zsuc\ m) = (a \leq m)$
<proof>

lemma *less-zpred-eq-le* [simp]: $(a \leq zpred\ m) = (a < m)$
<proof>

Number range

lemma *num-range-empty*: $(\{\} = (a .. b)) = (b < a)$
<proof>

lemma *num-range-empty2*: $((a .. b) = \{\}) = (b < a)$
<proof>

declare *num-range-empty* [simp] *num-range-empty2* [simp]

lemma *num-range-empty-imp*[simp]: $!!a. b < a \implies (a .. b) = \{\}$
<proof>

lemma *zadd-numb-range-collect*[simp]: $\{d. \exists n. (q..m). d = ((n::int)+s)\} = ((q+s)..(m+s))$
<proof>

lemma *num-range-zless-mem*[simp]: $!!a::int. n < a \implies (n \sim: (a..b))$
<proof>

lemma *num-range-zless-mem2*[simp]: $!!b::int. b < n \implies (n \sim: (a..b))$
<proof>

Appendix B. Isabelle Theories

lemma *num-range-Naturals-mem*:
!!a. [| a:%N; n : (a..b) |] ==> n: %N
<proof>

lemma *numb-range-single[simp]*: (a .. a) = {a}
<proof>

lemma *numb-range-insert-first*: !!a. a <= b ==> ((zpred a) .. b) = insert (zpred a) (a .. b)
<proof>

lemma *numb-range-insert-last*: !!a. a <= b ==> (a .. (zsuc b)) = insert (zsuc b) (a .. b)
<proof>

lemma *zadd-to-zdiff-zle*: ((a::int) <= b + c) = (a - c <= b)
<proof>

lemma *numb-range-mem-zadd[simp]*: ((a + b) : (m .. n)) = (a : (m-b .. n-b))
<proof>

lemma *numb-range-mem-neq-zsuc*: !!a. a ~ = m ==> (a : (zsuc m .. n)) = (a : (m .. n))
<proof>

declare *numb-range-mem-neq-zsuc* [symmetric, simp]

lemma *numb-range-mem-neq-zpred*:
!!a. a ~ = m ==> (a : (n .. m)) = (a : (n .. zpred m))
<proof>

declare *numb-range-mem-neq-zsuc* [symmetric, simp] *numb-range-mem-neq-zpred* [simp]

lemma *numb-range-subset[simp]*: !!k. k <= m ==> (m .. n) <= (k .. n)
<proof>

B.2. Integers in the Mathematical Toolkit

lemma *numb-range-mem-subset*[simp]: !!k. [| k < m ; a : (m .. n) |] ==> (a : (k .. n))
<proof>

lemma *numb-range-mem-subset2*[simp]:
!!n. [| n <= k ; a : (m .. n) |] ==> a : (m .. k)
<proof>

lemma *numb-range-mem-zsuc-zpred*[simp]: !!a. [| a : (m .. zpred n) |] ==> (zsuc a : (m .. n))
<proof>

lemma *empty-numb-range*[simp]: !!b. b < c ==> (((a .. b) Int (c .. d))) = {}
<proof>

lemma *numb-range-Un* [simp]: !!a. [| a <= (b + 1) ; b <= c |] ==>
((a .. b) Un (b + 1 .. c)) = (a .. c)
<proof>

Integers and Naturals

lemma *Integers-Subset*[simp]: {(n::int). (P n)} <= %Z
<proof>

lemma *Naturals-1-Mem*[simp]: ((n::int):%Z)
<proof>

lemma *Naturals-Mem*[simp]: ~ neg n --> n:%N
<proof>

lemma *Nat-zless-zadd*[simp]: !!b. [| b:%N ; a < c |] ==> (a < c + b)
<proof>

lemma *Nat-zle-zadd*[simp]: !!b. [| b:%N ; a <= c |] ==> a <= c + b
<proof>

lemma *Nat-zsuc*[simp]: !!x. x:%N ==> zsuc x:%N

Appendix B. Isabelle Theories

<proof>

lemma *Nat-zadd [simp]*: $!!n. [| n:\%N ; m:\%N |] ==> n+m:\%N$

<proof>

lemma *zsuc-simp*: $(zsuc\ x) = (x + 1)$

<proof>

lemma *Naturals-1-zadd-zle [simp]*: $!!n. (1::nat) \leq n ==> \sim (c + n \leq c)$

<proof>

lemma *Naturals-1-Mem [simp]*: $!!a. [| a:\%N ; a\sim=0 |] ==> (a : Naturals-1)$

<proof>

lemma *zless-eq-zadd-Suc*: $w < z ==> ?n. z = w + int\ (Suc\ n)$

<proof>

lemma *zless-eq-zadd*: $x \leq y = (?z::nat. y = x + int\ z)$

<proof>

lemma *znatprop1*: $!!x. x = int\ xnat ==> P(xnat::nat) = P(\$i\ x)$

<proof>

lemma *znatprop2*:

$!!x. [| x = int\ xnat ; y = int\ ynat |] ==> P(xnat::nat)(ynat::nat) = P(\$i\ x)(\$i\ y)$

<proof>

lemma *znat-Suc [simp]*: $int\ (Suc\ n) = zsuc\ (int\ n)$

<proof>

lemma *naturalsI*: $!!x. x = int\ xnat ==> (x : \%N)$

<proof>

lemma *nat2naturals*: $(?xnat. x = int\ xnat) = (x : \%N)$

<proof>

lemma *naturals-induct*:

assumes *prem1*: $x : \%N$

B.3. Functions in the Mathematical Toolkit

```
assumes prem2: !! x. P(x) ==> P(x + int 1)
assumes prem3: P(int 0)
shows P(x)
<proof>
```

```
lemma naturals-induct:
assumes prem1: x : %N
assumes prem2: !! x.[! x:%N; P(x) ] ==> P(x + int 1)
assumes prem3: P(int 0)
shows P(x)
<proof>
```

```
lemma z-add-induct:
assumes prem1: P(k::int)
assumes prem2: k <= x
assumes prem3: !! x. P(x) ==> P(x + int 1)
shows P(x)
<proof>
```

```
lemma naturals-shift:
[[n ∈ %N ; n ≠ 0 ] ==> ∃ m. (m ∈ %N ∧ n = m + 1)
<proof>
```

```
lemma naturals-exhaust :
assumes typing : n ∈ %N
assumes H1 : [[n ∈ %N ; n = 0 ] ==> P 0
assumes H2 : [[n ∈ %N ; ∃ m. (m ∈ %N ∧ n = m + 1) ] ==> P n
shows P n
<proof>
```

end

B.3. Functions in the Mathematical Toolkit

```
theory ZFun imports ZMathTool ZInteg begin
```

```
declare dom-implies-ran [simp]
```

B.3.1. Relation Apply

lemma *apply-singleton-rel*[simp]: $(\{(x,y)\}\% \hat{x}) = y$
 $\langle proof \rangle$

lemma *mem-apply*[rule-format]: $x:dom R \ \& \ (R\% \hat{x}) = y \ \longrightarrow \ (x,y):R$
 $\langle proof \rangle$

lemma *override-apply* [simp]:
 $(R (+) \{(x,y)\}) \% \hat{x} = y$
 $\langle proof \rangle$

lemma *override-by-pair-apply2* [simp]:
 $!!x. x \sim z \implies (R (+) \{(z,y)\}) \% \hat{x} = R \% \hat{x}$
 $\langle proof \rangle$

lemma *override-by-pair-apply1*[simp]:
 $x = z \implies (R (+) \{(z, y)\}) \% \hat{x} = y$
 $\langle proof \rangle$

lemma *apply-single*: $(\{(x,y). y = F x\}\% \hat{z}) = (F z)$
 $\langle proof \rangle$

lemma *apply-single-tuple*:
 $(\{((x1,x2),y). (y = (F x1 x2))\}\% \hat{(z1,z2)}) = (F z1 z2)$
 $\langle proof \rangle$

declare *dom-implies-ran* [simp del]

lemma *dom-un-apply*[rule-format]: $(z \sim: (dom Q)) \longrightarrow (((R Un Q)\% \hat{z}) = (R\% \hat{z}))$
 $\langle proof \rangle$

declare *dom-un-apply* [simp]

declare *dom-implies-ran* [simp]

lemma *insert-apply*: $!!a. z \sim fst a \implies (insert a R) \% \hat{z} = R \% \hat{z}$
 $\langle proof \rangle$

lemma *insert-apply2*: $!!a. z \sim: dom R \implies (insert a R) \% \hat{z} = \{a\} \% \hat{z}$
 $\langle proof \rangle$

declare *insert-apply* [simp] *insert-apply2* [simp]

B.3. Functions in the Mathematical Toolkit

lemma *dom-substr-apply* [simp]: $z \sim: S \implies (S \leftarrow -: R) \% \hat{z} = R \% \hat{z}$
<proof>

lemma *override-apply2* [simp]:
 $(z \sim: (\text{dom } S)) \implies (((R (+) S) \% \hat{z}) = (R \% \hat{z}))$
<proof>

lemma *override-apply1* :
 $z : \text{dom } S \implies (R (+) S) \% \hat{z} = S \% \hat{z}$
<proof>
declare *override-apply override-apply2*[simp]

B.3.2. Relations

lemma *Rel-Dom-subset*: $f : (A \leftarrow -> B) \implies \text{dom } f \leq A$
<proof>

lemma *Rel-Dom-Elem*: $[[f : (A \leftarrow -> B); x : \text{dom } f]] \implies x : A$
<proof>

lemma *empty-rel* [simp]: $\{\} : (A \leftarrow -> B)$
<proof>

lemma *select-rel* [rule-format,simp]: $R \sim = \{\} \dashrightarrow \{(x,y). x : S \ \& \ y = (\bullet y. y : R)\}$
 $: (S \leftarrow -> R)$
<proof>

lemma *rel-dom-member*: $[[f : (X \leftarrow -> Y); (x,y) : f]] \implies x : X$
<proof>

lemma *rel-pair-dom*[rule-format]:
 $r : (X \leftarrow -> Y) \dashrightarrow (a,b) : r \dashrightarrow a : X$
<proof>

lemma *rel-pair-ran*[rule-format]:
 $r : (X \leftarrow -> Y) \dashrightarrow (a,b) : r \dashrightarrow b : Y$
<proof>

lemma *rel-pair-dom-fst*[rule-format]:

Appendix B. Isabelle Theories

$r : (X \langle \dashrightarrow Y \rangle) \dashrightarrow (x:r) \dashrightarrow \text{fst } x : X$
 $\langle \text{proof} \rangle$

lemma *Dom-Collect-Rel-fst* [simp]:
 $\text{dom } \{p. \text{EX } x:A. p = (f \ x)\} = \{d. \text{EX } x:A. d = \text{fst } (f \ x)\}$
 $\langle \text{proof} \rangle$

lemma *Rel-Dom-Restr-Ident* [rule-format]:
 $x : (X \langle \dashrightarrow Y \rangle) \dashrightarrow (X \langle : x = x \rangle)$
 $\langle \text{proof} \rangle$

lemma *Rel-Dom-Pow* [rule-format]: $x : (X \langle \dashrightarrow Y \rangle) \dashrightarrow \text{dom } x : (\text{Pow } X)$
 $\langle \text{proof} \rangle$

declare *Rel-Dom-Pow* [simp]

lemma *x-in-S-implies-S-nonempty*: $x : S \implies S \sim = \{\}$

$\langle \text{proof} \rangle$

lemma *nonempty-Pow* [simp]: $\text{Pow } S \sim = \{\}$
 $\langle \text{proof} \rangle$

lemma *nonempty-Rel* [rule-format,simp]: $R \sim = \{\} \dashrightarrow (D \langle \dashrightarrow R \rangle) \sim = \{\}$
 $\langle \text{proof} \rangle$

Partial Functions

lemma *pfunI*:
assumes $p1 : f : A \langle \dashrightarrow B \rangle$
assumes $p2 : \forall x \ y1 \ y2. [\![(x,y1) : f; (x,y2) : f]\!] \implies y1 = y2$
shows $f : A \dashrightarrow B$
 $\langle \text{proof} \rangle$

lemma *pfunE*:
 $[\![f : A \dashrightarrow B; \forall x \ y1 \ y2. (x,y1) : f \ \& \ (x,y2) : f \implies y1 = y2]\!] \implies R$
 $\langle \text{proof} \rangle$

lemma *Pfun-Single*[simp]: $\{p\} : (A \dashrightarrow B) = (\text{fst } p : A \ \& \ \text{snd } p : B)$
 $\langle \text{proof} \rangle$

lemma *Partial-Union-Distr*[rule-format]:

B.3. Functions in the Mathematical Toolkit

$(f : (X \dashrightarrow Y) \ \& \ g : (X \dashrightarrow Y) \ \& \ (\text{dom } f \text{ Int } \text{dom } g) = \{\})$
 $\dashrightarrow (f \text{ Un } g : (X \dashrightarrow Y))$
 $\langle \text{proof} \rangle$

lemma *Partial-Union-Distr-rule* [simp]:
 $[[f : (X \dashrightarrow Y); \ g : (X \dashrightarrow Y); \ (\text{dom } f \text{ Int } \text{dom } g) = \{\}]]$
 $\implies (f \text{ Un } g : (X \dashrightarrow Y))$
 $\langle \text{proof} \rangle$

lemma *Partial-Dom-Restr* [simp]:
 $[[f : (A \dashrightarrow B); \ s : (\text{Pow } A)]] \implies ((s <: f) : (A \dashrightarrow B))$
 $\langle \text{proof} \rangle$

lemma *override-pfunI* [simp]:
 $[[f : (A \dashrightarrow B); \ g : (A \dashrightarrow B)]] \implies ((f (+) g) : (A \dashrightarrow B))$
 $\langle \text{proof} \rangle$

lemma *pfun-subset* [elim!]: $[[f : (a \dashrightarrow c); \ a \leq b]] \implies (f : (b \dashrightarrow c))$
 $\langle \text{proof} \rangle$

lemma *tfun-implies-pfun*: $f : A \dashrightarrow B \implies f : A \dashrightarrow B$
 $\langle \text{proof} \rangle$

lemma *Pfun-Rel*: $f : (A \dashrightarrow B) \implies f : (A \dashrightarrow B)$
 $\langle \text{proof} \rangle$
declare *tfun-implies-pfun* [simp] *Pfun-Rel* [simp]

lemma *empty-is-pfun* [simp]: $\{\} : (S \dashrightarrow R)$
 $\langle \text{proof} \rangle$

lemma *ex-elim4* [simp]: $(EX \ x. \ x : S) = (S \sim = \{\})$
 $\langle \text{proof} \rangle$

lemma *select-partial* [rule-format, simp]:
 $R \sim = \{\} \dashrightarrow \{(x, y). \ x : S \ \& \ y = (\text{SOME } y. \ y : R)\} : (S \dashrightarrow R)$

Appendix B. Isabelle Theories

$\langle proof \rangle$

lemma *beta-apply-pfun*[*rule-format,simp*]:

$r : (X \dashv\vdash Y) \dashv\vdash (a,b) : r \dashv\vdash (r \% ^ a) = b$
 $\langle proof \rangle$

lemmas *beta-apply-tfun = tfun-implies-pfun* [*THEN beta-apply-pfun*]

lemma *rel-apply-in-rel*[*rule-format,simp*]:

$r : (X \dashv\vdash Y) \dashv\vdash (a : \text{dom } r) \dashv\vdash (a, r \% ^ a) : r$
 $\langle proof \rangle$

lemma *collect-apply-rel*[*rule-format*]:

$r : (X \dashv\vdash Y) \dashv\vdash \{(x,y). x : \text{dom } r \ \& \ (r \% ^ x) = y\} = r$
 $\langle proof \rangle$

lemma *Insert-Partial*[*rule-format*]:

$((\text{insert } x \ F) : (X \dashv\vdash Y)) \dashv\vdash (F : (X \dashv\vdash Y))$
 $\langle proof \rangle$

lemma *Set-Diff-Partial* [*rule-format,simp*]:

$(F : (X \dashv\vdash Y)) \dashv\vdash ((F - A) : (X \dashv\vdash Y))$
 $\langle proof \rangle$

declare *dom-implies-ran* [*simp del*]

lemma *Set-Diff-Dom-Partial* [*rule-format,simp*]:

$(F : (X \dashv\vdash Y)) \dashv\vdash (G : (X \dashv\vdash Y)) \dashv\vdash (G \leq F) \dashv\vdash$
 $((\text{dom } (F - G)) = ((\text{dom } F) - (\text{dom } G)))$
 $\langle proof \rangle$

lemma *Rel-Apply-in-Partial-Ran*[*rule-format*]:

$(F : (X \dashv\vdash Y)) \dashv\vdash (x : \text{dom } F) \dashv\vdash (x, F \% ^ x) : F \dashv\vdash ((F \% ^ x) : Y)$
 $\langle proof \rangle$

declare *Rel-Apply-in-Partial-Ran* [*simp*]

lemma *pfun-apply*[*simp*]:

$[| F : (X \dashv\vdash Y); x : \text{dom } F |] \implies (F \% ^ x) : Y$
 $\langle proof \rangle$

B.3. Functions in the Mathematical Toolkit

lemma *Partial-Dom-Insert*[*rule-format*]:

$t : (X \dashrightarrow Y) \dashrightarrow a \sim : \text{dom } t \dashrightarrow a : X \dashrightarrow b : Y \dashrightarrow \text{insert } (a,b) t : (X \dashrightarrow Y)$

<proof>

lemma *Dom-Nat-Part-Func-Add*[*rule-format,simp*]:

$t : (\text{Naturals} \dashrightarrow X) \dashrightarrow m : \text{Naturals} \dashrightarrow \{p. \text{EX } n : \text{dom } t. p = (n + m, t\% ^n)\} : (\text{Naturals} \dashrightarrow X)$

<proof>

lemma *apply-ident*: $f : (X \dashrightarrow Y) \implies (f = \{p. \text{EX } n : \text{dom } f. p = (n, f\% ^n)\})$

<proof>

lemma *Partial-Func-Ran-Subset*[*rule-format*]:

$Y \leq Z \dashrightarrow f : (X \dashrightarrow Y) \dashrightarrow f : (X \dashrightarrow Z)$

<proof>

declare *dom-implies-ran* [*simp*]

lemma *part-func-select*[*rule-format,simp*]:

$R : (X \dashrightarrow Y) \dashrightarrow (x, y) : R \dashrightarrow y = (\bullet z. (x, z) : R)$

<proof>

lemma *nonempty-PFun* [*rule-format,simp*]: $R \sim = \{\} \dashrightarrow (D \dashrightarrow R) \sim = \{\}$

<proof>

Total Functions

lemma *empty-fun*: $\{\} : (\{\} \dashrightarrow R)$

<proof>

declare *empty-fun* [*simp*]

lemma *select-total* [*rule-format,simp*]:

$R \sim = \{\} \dashrightarrow \{(x,y). x : S \ \& \ y = (\bullet y. y : R)\} : (S \dashrightarrow R)$

<proof>

lemma *empty-is-pfun*[*simp*]: $\{\} \in A \dashrightarrow B$

<proof>

lemma *empty-biject*[*simp*]: $\{\} : (\{\} \dashrightarrow \{\})$

Appendix B. Isabelle Theories

$\langle proof \rangle$

lemma *total-func-dom*[*rule-format*]: $r : (X \dashrightarrow Y) \dashrightarrow x : X \dashrightarrow x : \text{dom } r$

$\langle proof \rangle$

declare *total-func-dom* [*simp*]

lemma *total-is-partial*[*rule-format,simp*]: $f : (X \dashrightarrow Y) \dashrightarrow f : (X \dashv\!\dashrightarrow Y)$

$\langle proof \rangle$

lemma *nonempty-Fun*[*rule-format,simp*]: $R \sim = \{\} \dashrightarrow (D \dashrightarrow R) \sim = \{\}$

$\langle proof \rangle$

Partial Injections

lemma *empty-pinj-fun*[*simp,intro!*]: $\{\} : (S \dashv\!\dashrightarrow R)$

$\langle proof \rangle$

declare *empty-pinj-fun* [*simp*]

Total Injections

lemma *empty-total-inj*: $\{\} : (\{\} \dashrightarrow R)$

$\langle proof \rangle$

declare *empty-total-inj* [*simp*]

Partial Surjections

lemma *empty-psurj-fun*[*simp*]: $\{\} : (S \dashv\!\dashrightarrow \{\})$

$\langle proof \rangle$

Total Surjections

lemma *empty-total-surj*[*simp*]: $\{\} : (\{\} \dashrightarrow \{\})$

$\langle proof \rangle$

declare *empty-total-surj* [*simp*]

declare *dom-implies-ran* [*simp del*]

Simplified Definitions

declare *Pfun-Rel* [*simp del*]

lemma *partial-func-simp* [*simp*]:

$(f : (A \dashv\!\dashrightarrow B)) = (f : (A \dashrightarrow B) \ \& \ f : (A \dashleftarrow B))$

B.3. Functions in the Mathematical Toolkit

$\langle \text{proof} \rangle$
 $(! x y1 y2. (x,y1):f \ \& \ (x,y2):f \ \dashrightarrow \ y1=y2))$

lemma *total-func-simp* [*simp*]:
 $(f : (A \dashrightarrow B)) = (f : (A \dashv\rightarrow B) \ \& \ \text{dom } f = A)$
 $\langle \text{proof} \rangle$

lemma *partial-inj-simp* [*simp*]:
 $(f : (A \dashv\rightarrow B)) = (f : (A \dashv\rightarrow B) \ \& \ (! x1 x2. (? y. (x1,y):f \ \& \ (x2,y):f \ \dashrightarrow \ x1=x2)))$
 $\langle \text{proof} \rangle$

lemma *total-inj-simp* [*simp*]:
 $(f : (A \dashrightarrow B)) = (f : (A \dashv\rightarrow B) \ \& \ \text{dom } f = A)$
 $\langle \text{proof} \rangle$

lemma *partial-surj-simp* [*simp*]:
 $(f : (A \dashrightarrow B)) = (f : (A \dashv\rightarrow B) \ \& \ \text{ran } f = B)$
 $\langle \text{proof} \rangle$

lemma *total-surj-simp* [*simp*]:
 $(f : (A \dashrightarrow B)) = (f : (A \dashv\rightarrow B) \ \& \ \text{dom } f = A \ \& \ \text{ran } f = B)$
 $\langle \text{proof} \rangle$

declare *partial-func-simp* [*simp del*]
declare *total-func-simp* [*simp del*]
declare *partial-inj-simp* [*simp del*]
declare *total-inj-simp* [*simp del*]
declare *partial-surj-simp* [*simp del*]
declare *total-surj-simp* [*simp del*]
declare *Pfun-Rel* [*simp*]

lemma *fin-part-funcI*:
 $!! f. [! f : A \dashv\rightarrow B; \ \text{dom } f : \text{Fin } A] \implies f : A \dashv\|\rightarrow B$
 $\langle \text{proof} \rangle$
declare *fin-part-funcI* [*intro!*]

lemma *fin-part-funcE*:
 $[! f : A \dashv\|\rightarrow B; \ [! f : A \dashv\rightarrow B; \ \text{dom } f : \%F A]] \implies P \implies P$

Appendix B. Isabelle Theories

$\langle proof \rangle$
declare *fin-part-funcE* [elim!]

lemma *insert-is-pfun[simp]*:
[[$x:A; y:B; x \notin \text{dom } S; S : A \dashv\rightarrow B$]] ==> (*insert* (x,y) S) : $A \dashv\rightarrow B$
 $\langle proof \rangle$

lemma *pair-pfunI[simp]*:
[[$x:A; y:B$]] ==> $\{(x,y)\} : A \dashv\rightarrow B$
 $\langle proof \rangle$

lemma *tfun-apply[simp]*:
[[$F : X \dashrightarrow Y; x : X$]] ==> $F \% ^ x : Y$
 $\langle proof \rangle$

lemma *pair-exhaust*:
[[$x : B \% x C$]] ==> $\exists y z. x = (y,z)$
 $\langle proof \rangle$

lemma *tfun-apply-fst*:
[[$f : A \dashrightarrow B \% x C; x : A$]] ==> $\text{fst}(f \% ^ x) : B$
 $\langle proof \rangle$

lemma *tfun-apply-snd*:
[[$f : A \dashrightarrow B \% x C; x : A$]] ==> $\text{snd}(f \% ^ x) : C$
 $\langle proof \rangle$

lemma *pfun-apply-fst*:
[[$f : A \dashv\rightarrow B \% x C; x : \text{dom } f$]] ==> $\text{fst}(f \% ^ x) : B$
 $\langle proof \rangle$

lemma *pfun-apply-snd*:
[[$f : A \dashv\rightarrow B \% x C; x : \text{dom } f$]] ==> $\text{snd}(f \% ^ x) : C$
 $\langle proof \rangle$

B.3. Functions in the Mathematical Toolkit

lemma *Pow-right-subset*[simp,elim!]:
[[$f : \%P (a \%x b)$; $b \leq c$]] ==> $f : \%P (a \%x c)$
<proof>

lemma *Pow-left-subset*[simp,elim!]:
[[$f : \%P (a \%x b)$; $a \leq c$]] ==> $f : \%P (c \%x b)$
<proof>

lemma *rel-ran-subset*[elim!]:
[[$f : a \<--> b$; $b \leq c$]] ==> $f : a \<--> c$
<proof>

lemma *rel-dom-subset*[elim!]:
[[$f : a \<--> c$; $a \leq b$]] ==> $f : b \<--> c$
<proof>

lemma *pfun-ran-subset*[elim!]:
[[$f : a -|-> b$; $b \leq c$]] ==> $f : a -|-> c$
<proof>

lemma *dom-override-I*:
[[$x : \text{dom } X$; $x : \text{dom } Y$]] ==> $x : \text{dom } (X (+) Y)$
<proof>

lemma *dom-Un-I*:
[[$x : \text{dom } X$; $x : \text{dom } Y$]] ==> $x : \text{dom } (X \text{ Un } Y)$
<proof>

lemma *dom-insert-I1*:
[[$x = \text{fst } a$]] ==> $x : \text{dom } (\text{insert } a Y)$
<proof>

lemma *dom-insert-I2*:
[[$x \neq \text{fst } a$; $x : \text{dom } Y$]] ==> $x : \text{dom } (\text{insert } a Y)$
<proof>

lemma *dom-dres-I*:
[[$x : X$; $x : \text{dom } Y$]] ==> $x : \text{dom } (X <: Y)$

Appendix B. Isabelle Theories

<proof>

lemma *dom-LambdaI*:

$\llbracket x:A \rrbracket \implies x : \text{dom } (\text{Lambda } A \ f)$

<proof>

lemma *pfun-implies-dom-bounded*: $f : A \dashrightarrow B \implies \text{dom } f \subseteq A$

<proof>

lemma *override-tfunI1[simp]*:

$\llbracket f : A \dashrightarrow B; g : A \dashrightarrow B \rrbracket \implies f(+)\ g : A \dashrightarrow B$

<proof>

lemma *override-tfunI2[simp]*:

$\llbracket f : A \dashrightarrow B; g : A \dashrightarrow B \rrbracket \implies f(+)\ g : A \dashrightarrow B$

<proof>

lemma *pfun-ext*:

assumes *f-pfun*: $f : A \dashrightarrow B$

assumes *g-pfun*: $g : A \dashrightarrow B$

assumes *dom-eq*: $\text{dom } f = \text{dom } g$

assumes *cong*: $\forall i. \llbracket i : \text{dom } f; \text{dom } f = \text{dom } g \rrbracket \implies f \% ^ i = g \% ^ i$

shows $f = g$

<proof>

Due to internal translation restrictions, certain expressions were represented by *rel_applf' (j .. k)*. The following property removes this artifact:

lemma *rel-appl-norm* :

$\text{rel_appl } f' (j \ .. \ k) = \{a. \text{EX } i:j..k. f \% ^ i = a\}$

<proof>

end

B.4. Finite Sets for the Mathematical Toolkit

theory *ZFinite* **imports** *ZFun* **begin**

B.4. Finite Sets for the Mathematical Toolkit

axioms

Fin-subset: $[[A \leq B; B : \%F M]] \implies A : \%F M$

lemma *Fin-mono*: $!!A B. A \leq B \implies \%F A \leq \%F B$
<proof>

lemma *Fin-mono2*: $[[X \leq Y; a : \%F X]] \implies a : \%F Y$
<proof>

lemma *Fin-subset-Pow*: $\%F A \leq \%P A$
<proof>

lemmas *FinD* = *Fin-subset-Pow* [*THEN subsetD* [*THEN PowD*]]

lemma *Fin-induct*:

assumes *major1*: $F : \%F A$

assumes *major2*: $P (\{\})$

assumes *prems*: $!!F x. [[x : A; F : \%F A; x \sim F; P(F)]] \implies P(\text{insert } x F)$

shows $P(F)$

<proof>

declare *Fin.insertI* [*simp*]

declare *Fin.emptyI* [*simp*]

declare *Fin.intros* [*simp*]

lemma *Fin-UnI*:

assumes *major*: $F : \%F A$

assumes *prems*: $G : \%F A$

shows $F \text{ Un } G : \%F A$

<proof>

lemma *subset-Fin* [*simp*]: $(F \text{ Un } G : \%F A) = (F : \%F A \ \& \ G : \%F A)$

<proof>

Appendix B. Isabelle Theories

lemma *insert-Fin* [*simp*]: $(\text{insert } a \ A : \%F \ M) = (a : M \ \& \ A : \%F \ M)$
<proof>

lemma *Fin-imageI*: $F : \%F \ A \implies h'F : \%F \ (h'A)$
<proof>

lemma *aux*:

assumes *major*: $c : \%F \ A$

assumes *prem1*: $b : \%F \ A$

assumes *prem2*: $P(b)$

assumes *prem3*: $\llbracket x : A; y : \%F \ A; x : y; P(y) \rrbracket \implies P(y - \{x\})$

shows $c \leq b \implies P(b - c)$

<proof>

lemma *Fin-empty-induct*: $\llbracket b : \%F \ A; P(b); \forall x y. \llbracket x : A; y : \%F \ A; x : y; P(y) \rrbracket \implies P(y - \{x\}) \rrbracket \implies P(\{\})$

<proof>

B.4.1. Finite Sets

lemma *Fin-Diff-Set*: $\llbracket (A - B) : \%F \ X; B : \%F \ X \rrbracket \implies A : \%F \ X$
<proof>

lemma *size-empty*: $(\# \ \{\}) = \text{int } 0$
<proof>

declare *size-empty* [*simp*]

lemma *card-empty-set*: $(t = \{\}) \implies (\text{card } t = 0)$
<proof>

lemma *Fin-finite*: $A : \%F \ X \implies \text{finite } A$
<proof>

lemma *finite-Fin*: $\llbracket \text{finite } A; A \leq X \rrbracket \implies A : \%F \ X$
<proof>

lemma *finite-vs-Fin*: $(\text{finite } A \ \& \ A \leq X) = (A : \%F \ X)$
<proof>

lemma *Collect-Dom-Insert*: $\{d. EX \ n : \text{insert } x \ F. d = xa \ n\} = \{d. EX \ n : F. d =$

B.4. Finite Sets for the Mathematical Toolkit

$xa\ n\} \cup n\ \{xa\ x\}$
 $\langle proof \rangle$

lemma *Collect-Fin-Func*:
assumes $prem1: A : \%F\ X$
assumes $prem2: \{d. EX\ x:X. d = f\ x\} \leq X$
shows $\{d. EX\ n : A. d = (f\ n)\} : \%F\ X$
 $\langle proof \rangle$

lemma *Collect-Fin-Func-Subset*: $[[A : \%F\ X ; \{d. EX\ x:X. d = f\ x\} \leq X]] \implies$
 $\{d. EX\ n : A. d = (f\ n)\} : \%F\ X$
 $\langle proof \rangle$

lemma *insert-card-eq*:
 $[[x \sim: F; y \sim: G; finite\ F; finite\ G; card\ F = card\ G]] \implies$
 $card\ (insert\ x\ F) = card\ (insert\ y\ G)$
 $\langle proof \rangle$

lemma *Nat-Fin-Add-Collect*: $[[A : \%F\ \%N ; m : \%N]] \implies \{d. EX\ n : A. d = (n$
 $+ m)\} : \%F\ \%N$
 $\langle proof \rangle$

lemma *Collect-Dom-Insert*:
 $\{d. EX\ n : insert\ x\ F. d = xa\ n\} = insert\ (xa\ x)\ \{d. EX\ n : F. d = xa\ n\}$
 $\langle proof \rangle$

lemma *size-image*:
 $!!A. [[A : \%F\ X ; inj-on\ f\ A]] \implies \#(f\ 'A) = \#A$
 $\langle proof \rangle$

lemma *fin-finite*: $!!A. A : \%F\ X \implies finite\ A$
 $\langle proof \rangle$

lemma *size-insert-disjoint* [simp]: $!!A. [[A : \%F\ X ; x \sim: A]] \implies \#(insert\ x\ A)$
 $=\ zsuc\ (\#A)$
 $\langle proof \rangle$

Appendix B. Isabelle Theories

lemma *card-size-eq*: $(\text{card } m) = (\$i (\# m))$
<proof>

lemma *Size-Naturals* [*simp*]: $\#m : \%N$
<proof>

lemma *size-empty2*: $!!x. m : \%F X \implies (\# m = 0) = (m = \{\})$
<proof>

lemma *card-empty2*: $m : \%F X \implies ((\text{card } m = 0) = (m = \{\}))$
<proof>

lemma *size-set-notempty* [*rule-format*]: $!!m. m : \%F X \implies m \sim = \{\} \dashrightarrow 0 < \#m$
<proof>

lemma *size-set-empty*: $\# \{\} = 0$
<proof>

lemma *card-notempty*: $!!x. [|x \sim = \{\}; x : \%F X|] \implies \text{card } x \sim = 0$
<proof>

B.4.2. Finite Functions

lemma *empty-fin-pfun* [*simp*, *intro!*]: $\{\} : (S \dashrightarrow R)$
<proof>

lemma *empty-fin-pinj* [*simp*, *intro!*]: $\{\} : (S \dashrightarrow R)$
<proof>

lemma *override-fpfun* [*simp*]: $[|f:(A \dashrightarrow B); g:(A \dashrightarrow B)|] \implies ((f (+) g):(A \dashrightarrow B))$
<proof>

lemma *fin-part-fun-single* [*simp*]: $[|fst p : A; snd p : B|] \implies \{p\} : (A \dashrightarrow B)$
<proof>

lemma *Fin-Dom-Partial*: $!!t. [|t : \%F (X \%x Y); t : (X \dashrightarrow Y)|] \implies (\text{dom } t:$

B.4. Finite Sets for the Mathematical Toolkit

$\%F X$
 $\langle proof \rangle$

lemma *Dom-Partial-Fin-lemma* [rule-format (no-asm)]:
 $!! dt. dt : \%F X ==> ALL t . t:(X -|-> Y) --> dom t = dt --> t : Fin (X$
 $\%x Y)$
 $\langle proof \rangle$

lemma *Dom-Partial-Fin*: $!! t. [|t : (X -|-> Y); dom t : \%F X|] ==> (t : Fin (X$
 $\%x Y))$
 $\langle proof \rangle$

lemma *Fin-Partial-Func*[simp]: $!!t. t : (X -||-> Y) ==> t : \%F (X \%x Y)$
 $\langle proof \rangle$

lemma *Fin-Partial-Dom-Func* [simp]: $!!t. t : (X -||-> Y) ==> dom t : \%F X$
 $\langle proof \rangle$

lemma *Fin-Partial-Partial-Func* [simp]: $t : (X -||-> Y) ==> t : (X -|-> Y)$
 $\langle proof \rangle$

lemma *Dom-Int-Fin-Func-Add*:
 $[|dom t : \%F \%Z ; t : (\%Z -|-> X); m : \%Z|] ==>$
 $dom \{p. EX n:dom t. p = (n + m, t\% ^n)\} : \%F \%Z$
 $\langle proof \rangle$

lemma *Dom-Nat-Fin-Func-Add*:
 $[|dom t : \%F \%N ; t : (\%N -|-> X); m : \%N|] ==>$
 $dom \{p. EX n:dom t. p = (n + m, t\% ^n)\} : \%F \%N$
 $\langle proof \rangle$

lemma *Dom-Nat-Fin-Part-Func-Add* [simp]: $[|t : (\%N -||-> X); m : \%N|] ==>$
 $\{p. EX n:dom t. p = (n + m, t\% ^n)\} : (\%N -||-> X)$
 $\langle proof \rangle$

lemma *Fin-Part-Func-Dom-Size-aux*:
assumes *prem1*: $(dom t) = dt$
assumes *prem2*: $t:(X -|-> Y)$
assumes *prem3*: $dt : \%F X$
shows $\# t = \# (dom t)$

Appendix B. Isabelle Theories

<proof>

lemma *Fin-Part-Func-Dom-Size*: $t : (X \dashv\vdash Y) \implies \# t = \# (\text{dom } t)$
<proof>

lemma *Fin-part-func-union* [simp]:
 $[[f : (X \dashv\vdash Y) ; g : (X \dashv\vdash Y) ; ((\text{dom } f) \text{ Int } (\text{dom } g)) = \{\}] \implies$
 $f \text{ Un } g : (X \dashv\vdash Y)$
<proof>

B.4.3. Finite Number Range

lemma *fin-x-upto-x*: $(x .. x) : \%F \%Z$
<proof>

lemma *numb-range-Fin-add*: $a:\%N \implies (a .. (a + \text{int } b)) : \%F \%N$
<proof>

lemma *numb-range-Fin-zadd-aux*: $[[a:\%N ; b:\%N]] \implies (a .. a+b) : \%F \%N$
<proof>

lemma *numb-range-Size-add*: $a:\%N \implies \#(a .. (a + \text{int } b)) = \text{zsuc } (\text{int } b)$
<proof>

lemma *numb-range-Size-zadd1*: $!!b. [[b:\%N ; a:\%N]] \implies \#(a .. (a + b)) = \text{zsuc } b$
<proof>

lemma *numb-range-Size-zadd* [simp]: $!!a. [[a : \%N ; a \leq (b + 1)]] \implies \#(a .. b) = \text{zsuc } (b - a)$
<proof>

lemma *aux*: $!!z. \sim z \leq w \implies w < (z::\text{int})$
<proof>

B.5. Sequences in the Mathematical Toolkit

lemma *numb-range-Fin-zadd*: !!*a*. [|*a* : %*N*|] ==> (*a* .. *b*) : %*F* %*N*

<*proof*>

declare *numb-range-Fin-zadd* [*simp*]

lemma *zadd-numb-range-Fin-Nat* [*simp*]: !!*a*. [| *a* : %*N* ; *c* : %*N*|] ==> (*a*+*c* .. *b*+*c*) : %*F* %*N*

<*proof*>

lemma *numb-range-set-notempty* [*simp*]: !!*x*. [|*x*: %*F* *X* ; *x* ~ = {} |] ==> (1: (1..#*x*)

<*proof*>

lemma *numb-range-mem* [*simp*]: [|*a* <= *n* ; *n* <= *b*|] ==> *n* : (*a* .. *b*)

<*proof*>

lemma *upto-size-empty*: *y* < *x* ==> #(*x* .. *y*) = int 0

<*proof*>

lemma *size-x-upto-x*: card (*x* .. *x*) = 1

<*proof*>

end

B.5. Sequences in the Mathematical Toolkit

theory *ZSeq* imports *ZFinite Wellfounded-Relations* begin

types

'*a* seq = int <=> '*a*

constdefs

seq :: '*a* set => ('*a* seq) set

Appendix B. Isabelle Theories

```

seq S      == { f. (f: (Naturals -||-> S)) & (dom f = ( 1 .. (# f)))}
seq1      :: 'a set => ('a seq) set
seq1 S    == {f. f: (seq S) & ((int (0::nat)) < (# f))}
iseq      :: 'a set => ('a seq) set
iseq S    == (seq S) Int (Naturals >-||-> S)

emptyseq  :: 'a seq                                     (%<%>)
emptyseq  == {}
insertseq :: ['a, 'a seq] => 'a seq
insertseq x s == insert (1,x) {p. EX n: dom s. p = (zsuc n,s% ^ n)}

insertseq-rel :: 'a set => ('a seq * 'a seq) set
insertseq-rel A == {(s,s'). s : seq A & (? a. s' = insertseq a s)}

seq-case :: ['a set, 'b, ('a => 'a seq => 'b), 'a seq] => 'b
seq-case A b f s == • z. (s = %<%> --> z = b) & (! a:A. ! t:seq A. s =
insertseq a t --> z = f a t)

seq-rec :: ['a set, 'b, ['a, 'a seq, 'b] => 'b, 'a seq] => 'b
seq-rec A b d == wfrec (insertseq-rel A) (%f. seq-case A b (%a t. d a t (f t)))

mhead :: 'a set => 'a seq <=> 'a
mhead A == lambda s: seq A. s % ^ 1

mtail :: 'a set => 'a seq <=> 'a seq
mtail A == lambda s : seq A. lambda n : (1 .. ((# s) - (1))) . s % ^ (n+1)

```

syntax

• Sequence :: args => 'a seq (%<(-)%>)

translations

%< x, xs %> == insertseq x %<xs%>
 %< x %> == insertseq x %<%>

axioms

lem1: !!x s. [| s : seq A; x : s |] ==> x = (1, mhead A % ^ s) | (EX n : 1 .. # s - 1. x = (zsuc n, mtail A % ^ s % ^ n))
 insert-mhead-mtail: !!s. [| s : seq A; s ~ = %<%> |] ==> s = insertseq (mhead A % ^ s) (mtail A % ^ s)
 seq-mem-dom-neq-first: [|s: seq X ; a:s ; a ~ = (1,s% ^ 1)|] ==> ((fst a) : (2..#s))

B.5. Sequences in the Mathematical Toolkit

Dom-insertseq: $[[y:X; t:seq X]] \implies dom (insertseq y t) = (1..#(insertseq y t))$
insertseq-eq: $[[s:seq X; t:seq Y; insertseq a s = insertseq b t]] \implies (a=b)$

$\langle ML \rangle$

lemma *seqI*: $!!s. [[s : \%N \ -||-\> A ; dom s = (1 .. \# s)]] \implies s : seq A$
 $\langle proof \rangle$

lemma *seqE*: $[[s:seq A; [[s : \%N \ -||-\> A ; dom s = (1 .. \# s)]] \implies R]] \implies R$
 $\langle proof \rangle$

lemma *emptyseqI* [*simp*]: $!!s. x:s \implies s \sim = \%<\%>$
 $\langle proof \rangle$

lemma *mhead-eq*[*simp*]: $!!s. [[s : seq A; s \sim = \%<\%>]] \implies mhead A \%^ s = s\%^1$
 $\langle proof \rangle$

lemma *mtail-eq* [*simp*]: $!!s. [[s : seq A; s \sim = \%<\%>]] \implies mtail A \%^ s = (\lambda n : (1 .. ((\# s) - (1))) . s \%^{(n + 1)})$
 $\langle proof \rangle$

lemma *aux*: $zpred 2 = 1$
 $\langle proof \rangle$

lemma *aux1*: $zsuc (x - int 1) = x$
 $\langle proof \rangle$

sequences-general

lemma *seq-ran-subset* [*simp*]: $(s : seq X) \implies (ran s <= X)$
 $\langle proof \rangle$

lemma *size-emptyseq* [*simp*]: $\# \%<\%> = 0$
 $\langle proof \rangle$

lemma *ran-emptyseq* [*simp*]: $ran (\%<\%>) = \{\}$
 $\langle proof \rangle$

Appendix B. Isabelle Theories

lemma *dom-emptyseq* [simp]: $\text{dom } (\%<\%>) = \{\}$
<proof>

lemma *seq-imp-Fin-func*: $t : \text{seq } X \dashrightarrow t : (\%N \dashrightarrow X)$
<proof>

lemma *seq-imp-numb-range*: $t : \text{seq } X \implies (\text{dom } t) = (1 ..\#t)$
<proof>

lemma *empty-seq* [simp]: $\%<\%> : \text{seq } UNIV$
<proof>

lemma *empty-seq2* [simp]: $\%<\%> : \text{seq } G$
<proof>

lemma *Seq-Subset* [simp]: $[X \leq Y ; s : \text{seq } X] \implies s : \text{seq } Y$
<proof>

lemma *first-elem-notemptyseq* [simp]: $[x : \text{seq } X ; x \sim \%<\%>] \implies (1, x\% \wedge 1)$
 $: x$
<proof>

lemma *zsuc-zpred*: $\text{zsuc } (\text{zpred } z) = z$
<proof>

declare *seq-mem-dom-neq-first* [simp]

lemma *seq-Collect-zpred-zsuc-Fin-Func*: $x : \text{seq } X \implies$
 $\{p. \text{EX } n : 1..\text{zpred } (\#x). p = (n, x\% \wedge (\text{zsuc } n))\} : (\%N \dashrightarrow X)$
<proof>

lemma *unknown*:
 $!!x. [x : \text{seq } X ; x \sim \%<\%>] \implies 0 < \#x$
<proof>

B.5. Sequences in the Mathematical Toolkit

lemma *size-ge-zero* [simp]: $0 \leq \# s$
 ⟨proof⟩

lemma *size-is-Nat* [simp]: $\text{int } x : \mathbb{N}$
 ⟨proof⟩

lemma *numb-range-size-front* [simp]: $\#(1 .. \#s) = \#s$
 ⟨proof⟩

insertseq

lemma *Insertseq-Partial-Func* [simp]:
 $[[y : X ; t : (\mathbb{N} \dashv\rightarrow X) ; ((\text{dom } t) = (1.. \#t))]]$
 $\implies \text{insertseq } y \ t : (\mathbb{N} \dashv\rightarrow X)$
 ⟨proof⟩

lemma *Insertseq-Dom-Fin-Nat* [simp]: $!!t. \text{dom } t : \text{Fin } \mathbb{N} \implies \text{dom } (\text{insertseq } y \ t) : \mathbb{F} \ \mathbb{N}$
 ⟨proof⟩

lemma *Insertseq-Fin-Part-Func-Nat*: $[[y : X ; t : (\mathbb{N} \dashv\rightarrow X) ; ((\text{dom } t) = (1.. \#t))]] \implies$
 $\text{insertseq } y \ t : (\mathbb{N} \dashv\rightarrow X)$
 ⟨proof⟩

declare *size-insert-disjoint* [simp del]
declare *size-ge-zero* [simp del] *numb-range-size-front* [simp del]

lemma *Dom-size-insert-seq* [simp]: $t : \text{seq } X \implies \#(\text{dom } (\text{insertseq } y \ t)) = 1 + \#(\text{dom } t)$
 ⟨proof⟩

lemma *size-insert-seq*:
 $!!t. [[t : \text{seq } X ; y : X]] \implies \#(\text{insertseq } y \ t) = \text{zsuc } (\# t)$
 ⟨proof⟩

Appendix B. Isabelle Theories

declare *size-insert-seq* [*simp*]

declare *size-ge-zero* [*simp del*] *numb-range-size-front* [*simp del*]

declare *Dom-insertseq* [*simp*]

declare *size-ge-zero* [*simp*] *numb-range-size-front* [*simp*]

lemma *insertseq-seq-pred* [*simp*]: $[[t:(seq\ X) ; y:X]] \implies (insertseq\ y\ t):(seq\ X)$
<proof>

declare *insertseq-seq-pred* [*simp*]

lemma *insertseq-not-empty* [*simp*]: $insertseq\ a\ b \sim = \%<\%>$
<proof>

lemma *insertseq-not-empty2* [*simp*]: $\%<\%> \sim = insertseq\ a\ b$
<proof>

lemma *insert-not-absorb2* [*simp*]: $\sim(b \leq c) \dashrightarrow insert\ a\ b \sim = c$
<proof>

lemma *insertseq-not-absorb* [*simp*]: $b: seq\ X \implies insertseq\ a\ b \sim = b$
<proof>

axioms *insertseq-eq-lemma*: $!!s. [[s: seq\ X ; t: seq\ Y] \implies$
 $\{p. EX\ x: dom\ s . p=(x+(n::int),s\ \% \hat{x})\} =$
 $\{p. EX\ x: dom\ t . p=(x+n,t\ \% \hat{x})\} \dashrightarrow$
 $\{p. EX\ x: dom\ s . p=(x,s\ \% \hat{x})\} = \{p. EX\ x: dom\ t . p=(x,t\ \% \hat{x})\}$

B.5. Sequences in the Mathematical Toolkit

lemma *insert-eq*: $insert\ a\ m = insert\ a\ n \dashrightarrow a \sim : m \dashrightarrow a \sim : n \dashrightarrow m = n$
 <proof>

lemma *insertseq-eq-seq-help*:
assumes *prem1*: $s : seq\ X$
assumes *prem2*: $t : seq\ Y$
assumes *prem3*: $insertseq\ a\ s = insertseq\ a\ t$
shows $(s=t)$
 <proof>

lemma *insertseq-eq-seq*:
 $[[\ s : seq\ X; t : seq\ Y; insertseq\ a\ s = insertseq\ b\ t]] \implies (s=t)$
 <proof>

lemma *insertseq-relI*:
 $!!s. [[\ s : seq\ A; s' = insertseq\ a\ s]] \implies (s,s') : insertseq-rel\ A$
 <proof>

lemma *insertseq-relE*: $[[\ (s,s') : insertseq-rel\ A; !!a. [[\ s : seq\ A; s' = insertseq\ a\ s]] \implies R\]]] \implies R$
 <proof>

declare *insertseq-relI* [*intro!*]
declare *insertseq-relE* [*elim!*]

lemma *insertseq-wf-lem*: $insertseq-rel\ A \leq measure\ (\% x. zint(zsize(dom\ x)))$
 <proof>

lemmas *insertseq-wf* = *insertseq-wf-lem* [*THEN wf-measure* [*THEN wf-subset*]]

lemma *mhead-closed*: $!!s. [[\ s : seq\ A; s \sim = \%<\%>]]] \implies mhead\ A\ \% \hat{\ } s : A$
 <proof>

lemma *empty2-lem*: $\{(x,y). False\} = \{\}$
 <proof>

lemma *insert-lem*:
 $\{(xa,y). xa : insert\ x\ F \ \& \ y = f\ xa\} =$

Appendix B. Isabelle Theories

insert $(x, f x) \{(xa, y). xa : F \ \& \ y = f xa\}$
<proof>

lemma *finite-lem*: $!! f. \text{finite } A \implies \text{finite } \{(x,y). x:A \ \& \ y = f x\}$
<proof>

lemma *zsize-lambda*: $!!f. A : \%F X \implies (\# (Lambda A f)) = (\# A)$
<proof>

lemma *numb-range-empty*: $!! x. y < x \implies x .. y = \{\}$
<proof>

<ML>

lemma *numb-range-lemma*: $((1::int) .. \# (1 .. n)) = (1 .. n)$
<proof>

lemma *lambda-total1*:
 $(!! x. x:A \implies f x : B) \implies (\lambda x : A. f x) : A \dashrightarrow B$
<proof>

lemma *lem2*: $!!n. 1 \leq n \implies \text{int } 0 \leq n$
<proof>

lemma *mtail-closed*: $!!s. [\ s : \text{seq } A; \ s \sim \%<\%> \] \implies \text{mtail } A \ \%^{\wedge} s : \text{seq } A$
<proof>

lemma *seqE-lem*:
 $!! s. s : \text{seq } A$
 $\implies (\exists x s' : \text{seq } A. \exists a:A. s = \text{insertseq } a s') \mid s = \%<\%>$
<proof>

lemma *seq-cases*:
assumes *p1*: $s : \text{seq } A$
assumes *p2*: $s = \%<\%> \implies P s$
assumes *p3*: $!!a s'. [\ s' : \text{seq } A; \ a:A; \ s = (\text{insertseq } a s') \] \implies P s$
shows $P s$
<proof>

lemma *seq-case-emptyseq*: $\text{seq-case } A \ a \ f \ \%<\%> = a$
<proof>

B.5. Sequences in the Mathematical Toolkit

lemma *seq-case-insertseq*:

!!s. [| s: seq A; x : A |] ==> seq-case A a f (insertseq x s) = f x s

<proof>

declare *seq-case-emptyseq* [simp] *seq-case-insertseq* [simp]

lemma *seq-rec-unfold-help*:

(%s. seq-rec A c d s) =
wfrec (insertseq-rel A) (%f. seq-case A c (%a t. d a t (f t)))

<proof>

lemmas *seq-rec-unfold* = insertseq-wf [THEN *seq-rec-unfold-help* [THEN *eq-reflection* [THEN *def-wfrec*]]]

lemma *seq-rec-emptyseq*: seq-rec A c h %<%> = c

<proof>

lemma *seq-rec-insertseq*: !!s. [| s: seq A; a:A |] ==>

seq-rec A c h (insertseq a s) = h a s (seq-rec A c h s)

<proof>

lemma *def-seq-rec-emptyseq*:

assumes *rew*: !!s. f s == seq-rec A c h s

shows f(%<%>) = c

<proof>

lemma *def-seq-rec-insertseq*:

assumes *rew*: !!s. f s == seq-rec A c h s

assumes *p1*: s : seq A

assumes *p2*: a : A

shows f(insertseq a s) = h a s (f s)

<proof>

lemma *seq-induct*:

assumes *p1*: s:seq A

assumes *p2*: P %<%>

assumes *p3*: !! a s. [| s : seq A; a:A; P s |] ==> P(insertseq a s)

shows P s

<proof>

Appendix B. Isabelle Theories

lemma *seq-imp-fin*: !! X . $list : seq\ X \implies list : \%F\ (\%N\ \%x\ X)$
<proof>

lemma *card-list-zero-imp-list-empty*:
!! X . $list : seq\ X \implies (\#\ list = 0) = (list = \%<\%>)$
<proof>

lemma *card-list-zero-imp-list-empty2*: !! X . $list : seq\ X \implies (\#\ list \leq 0) = (list = \%<\%>)$
<proof>

lemma *card-list-zero-imp-list-empty3*: !! X . $list : seq\ X \implies (\#\ list < 1) = (list = \%<\%>)$
<proof>

declare *card-list-zero-imp-list-empty* [*simp*] *card-list-zero-imp-list-empty2* [*simp*]
card-list-zero-imp-list-empty3 [*simp*]

declare *card-list-zero-imp-list-empty* [*simp del*] *card-list-zero-imp-list-empty2* [*simp del*]
card-list-zero-imp-list-empty3 [*simp del*]

lemma *zsize-image*: !! f . [*inj* f ; $x : \%N$] $\implies \#(f\ ' (x..y)) = \#(x..y)$
<proof>

end

B.6. Support for a rudimentary lifter

theory *FunAbs* **imports** *ZMathTool* *ZFun* **begin**

types

$(\ 'a, \ 'b)$ *typeabs* = $(\ 'b\ set) * (\ 'a \implies \ 'b) * (\ 'b \implies \ 'a)$

B.6. Support for a rudimentary lifter

constdefs

```

TASet :: ('a, 'b) typeabs => ('b set)
TASet == fst

TARep :: ('a, 'b) typeabs => ('a => 'b)
TARep == fst o snd

TAAbs :: ('a, 'b) typeabs => ('b => 'a)
TAAbs == snd o snd

Typeabs :: ('a, 'b) typeabs => bool
Typeabs TA == (let T = TASet TA;
                R = TARep TA;
                A = TAAbs TA
                in T ~ = {}
                & (ALL (x::'a). (R x) : (T::'b set))
                & (ALL (x::'a). (A (R x)) = x)
                & (ALL (y::'b). y : T --> (R (A y)) = y))

FRep :: [('a, 'b) typeabs, 'a => 'c] => ('b <=> 'c)
FRep TA f == {z. EX x. z = ((TARep TA x), (f x))}

FAbs :: [('a, 'b) typeabs, ('b <=> 'c), 'a] => 'c
FAbs TA g == (% x. g % ^ (TARep TA x))

PRep :: ['a => 'b] => ('a <=> 'b)
PRep f == {p. EX x. p = (x, f x)}

PAbs :: [('a <=> 'b), 'a] => 'b
PAbs R == (% x. R % ^ x)

```

lemma *TASet*: $TASet (T,R,A) = T$
<proof>

lemma *TARep*: $TARep (T,R,A) = R$
<proof>

Appendix B. Isabelle Theories

lemma *TAAbs*: $TAAbs (T,R,A) = A$
 $\langle proof \rangle$

declare *TASet* [*simp*] *TARep* [*simp*] *TAAbs* [*simp*]

lemma *TASet-nonempty*[*rule-format*] : $Typeabs TA \dashrightarrow (TASet TA) \sim = \{\}$
 $\langle proof \rangle$

lemma *TARep-in-TASet*[*rule-format*] : $Typeabs TA \dashrightarrow (TARep TA x) : (TASet TA)$
 $\langle proof \rangle$

lemma *TAAbs-inverse*[*rule-format*] : $Typeabs TA \dashrightarrow (TAAbs TA (TARep TA x)) = x$
 $\langle proof \rangle$

lemma *TARep-inverse*[*rule-format*] : $Typeabs TA \dashrightarrow y : (TASet TA) \dashrightarrow (TARep TA (TAAbs TA y)) = y$
 $\langle proof \rangle$

declare *TASet-nonempty* [*simp*] *TARep-in-TASet* [*simp*] *TAAbs-inverse* [*simp*] *TARep-inverse* [*simp*]

lemma *TARep-inj*[*rule-format*] : $Typeabs TA \dashrightarrow (inj (TARep TA))$
 $\langle proof \rangle$

lemma *TARep-iff-inj*[*rule-format*] : $Typeabs TA \dashrightarrow ((TARep TA x) = (TARep TA y)) = (x = y)$
 $\langle proof \rangle$

declare *TARep-iff-inj* [*simp*]

lemma *FRep-apply*[*rule-format*] : $Typeabs TA \dashrightarrow ((FRep TA f) \% ^ (TARep TA x)) = (f x)$
 $\langle proof \rangle$

lemma *FRep-rel*[*rule-format*] : $Typeabs TA \dashrightarrow (FRep TA f) : ((TASet TA) <--> UNIV)$
 $\langle proof \rangle$

declare *FRep-rel* [*simp*]

lemma *FRep-pfun*[*rule-format*] : $Typeabs TA \dashrightarrow (FRep TA f) : ((TASet TA)$

B.6. Support for a rudimentary lifter

$-|-> UNIV$
 $\langle proof \rangle$

declare $FRep\text{-}pfun$ [simp]

lemma $FRep\text{-}dom$ [rule-format] : $Typeabs\ TA \dashrightarrow dom\ (FRep\ TA\ f) = TASET\ TA$
 $\langle proof \rangle$

declare $FRep\text{-}dom$ [simp]

lemma $FRep\text{-}total\text{-}fun$ [rule-format] : $Typeabs\ TA \dashrightarrow (FRep\ TA\ f) : ((TASET\ TA) \dashrightarrow UNIV)$
 $\langle proof \rangle$

declare $FRep\text{-}total\text{-}fun$ [simp]

lemma $FAbs\text{-}inverse$ [rule-format] : $Typeabs\ TA \dashrightarrow (FAbs\ TA\ (FRep\ TA\ f)) = f$
 $\langle proof \rangle$

declare $FAbs\text{-}inverse$ [simp]

lemma $FRep\text{-}inverse$ [simp]: $Typeabs\ TA \dashrightarrow r : ((TASET\ TA) \dashrightarrow UNIV) \dashrightarrow (FRep\ TA\ (FAbs\ TA\ r)) = r$
 $\langle proof \rangle$

lemma $Typeabs\text{-}Fun$:

assumes $prems$: $Typeabs\ TA$

shows $Typeabs\ (TASET\ TA \dashrightarrow UNIV, FRep\ TA, FAbs\ TA)$
 $\langle proof \rangle$

lemma $apply\text{-}FAB$ [rule-format] :

$Typeabs\ TA \dashrightarrow x : TASET\ TA \dashrightarrow (g \% ^ x) = (FAbs\ TA\ g)\ (TAAbs\ TA\ x)$
 $\langle proof \rangle$

lemma $TAAbs\text{-}inj\text{-}onto$ [rule-format]:

$Typeabs\ TA \dashrightarrow inj\text{-}on\ (TAAbs\ TA)\ (TASET\ TA)$
 $\langle proof \rangle$

lemma $TAAbs\text{-}iff\text{-}inj\text{-}onto$ [rule-format]:

$Typeabs\ TA \dashrightarrow x : TASET\ TA \dashrightarrow y : TASET\ TA \dashrightarrow$
 $((TAAbs\ TA\ x) = (TAAbs\ TA\ y)) = (x = y)$
 $\langle proof \rangle$

lemma $PRep\text{-}inverse$ [rule-format]: $R : (UNIV \dashrightarrow UNIV) \dashrightarrow ((PRep\ (PAbs$

Appendix B. Isabelle Theories

$R)) = R)$
 $\langle proof \rangle$

lemma *PAbs-inverse*: $(PAbs (PRep f)) = f$
 $\langle proof \rangle$

lemma *PRep-apply*: $(f x) = ((PRep f) \% ^ x)$
 $\langle proof \rangle$

lemma *PAbs-apply*: $(R \% ^ x) = ((PAbs R) x)$
 $\langle proof \rangle$
end

B.7. A rudimentary Theory-Morphism from List to Sequences

theory *ZSeqtoList* **imports** *ZSeq FunAbs* **begin**

consts

Rep-seqtype :: $'a \text{ list} \Rightarrow ('a \text{ seq})$

primrec

seqOfList-Nil: $Rep-seqtype [] = \% < \% >$

seqOfList-Cons: $Rep-seqtype (x \# xs) = insertseq x (Rep-seqtype xs)$

constdefs

seqtype :: $('a \text{ seq}) \text{ set}$

seqtype == $seq \text{ UNIV}$

Abs-seqtype :: $('a \text{ seq}) \Rightarrow 'a \text{ list}$

Abs-seqtype == $(\% y. \bullet x. Rep-seqtype(x)=y)$

ListSeqAbs :: $('a \text{ list}, 'a \text{ seq}) \text{ typeabs}$

ListSeqAbs == $(seqtype, Rep-seqtype, Abs-seqtype)$

concatseq :: $('a \text{ seq} * 'a \text{ seq}) \Leftrightarrow 'a \text{ seq}$

concatseq == $\{(s,t),y). y = TAREp \text{ ListSeqAbs} \\ ((TAAbs \text{ ListSeqAbs } s) \bullet (TAAbs \text{ ListSeqAbs } t))\}$

Appendix B. Isabelle Theories

```

prefixseq == {(s,t). ? v:seq UNIV. (concatseq%^(s,v)) = t}
suffixseq :: 'a seq <=> 'a seq
suffixseq == {(s,t). ? u:seq UNIV. (concatseq%^(u,s)) = t}
inseq     :: 'a seq <=> 'a seq
inseq     == {(s,t). ? v:seq UNIV. (concatseq%^(v,s)) = t}

```

syntax

```

*concatseq ::['a seq, 'a seq] => 'a seq    ((2 - %&x / -) [70,71] 70)
*headseq   ::('a seq) => 'a                ((1 head -) 70)
*lastseq   ::('a seq) => 'a                ((1 last -) 70)
*frontseq  ::('a seq) => ('a seq)          ((1 front -) 70)
*tailseq   ::('a seq) => ('a seq)          ((1 tail -) 70)
*filtering ::['a seq, 'a set] => 'a seq    ((2 - %| ' / -) [70,71] 70)
*revseqseq ::('a seq) => ('a seq)          ((1 revseq -) 70)
*prefix    ::['a seq, 'a set] => 'a seq    ((2 - prefix / -) [70,71] 70)
*suffix    ::['a seq, 'a set] => 'a seq    ((2 - suffix / -) [70,71] 70)
*in        ::['a seq, 'a set] => 'a seq    ((2 - in / -) [70,71] 70)

```

translations

```

s %& ^ t == concatseq%^(s,t)
head s   == headseq% ^s
last s   == lastseq% ^s
front s  == frontseq% ^s
tail s   == tailseq% ^s
revseq s == revseqseq% ^s
s %| ' m == filtering% ^ (s,m)
s prefix t == (s,t):prefixseq
s suffix t == (s,t):suffixseq
s in t    == (s,t):inseq

```

axioms

```

snoc-neq-empty: t %& ^ %< aa %> ~ = %< %>
snoc-inj1:    [| t:seq A; t':seq A; aa: A; aa':A |] ==>
              t %& ^ %< aa %> = t' %& ^ %< aa' %> ==> t = t'
snoc-inj2:    [| t:seq A; t':seq A; aa: A; aa':A |] ==>
              t %& ^ %< aa %> = t' %& ^ %< aa' %> ==> aa = aa'

```

B.7. A rudimentary Theory-Morphism from List to Sequences

rev-insert-eq-snoc:

$$\begin{aligned} & \llbracket s:\text{seq } A; a:A \rrbracket \implies \\ & \text{revseq } (\text{insertseq } a \ s) = s \ \% \& \wedge \ \% \langle a \ \% \rangle \end{aligned}$$

$$\text{seq-card-dom}: s:\text{seq } A \implies \#(\text{dom } s) = \# \ s$$

$$\text{mem-concatseq}: s : \text{seq } X \dashrightarrow t : \text{seq } Y \dashrightarrow x : (\text{ran}(s \ \% \& \wedge \ t)) = (x : (\text{ran } s) \mid x : (\text{ran } t))$$

$$\text{mem-filtering}: s : \text{seq } X \dashrightarrow x : (\text{ran}(s \ \% \mid \wedge \ V)) = (x : (\text{ran } s) \ \& \ (x : V))$$

$$\text{snoc-wf-lem}: \text{snoc-rel } A \leq \text{measure } (\% \ x. \ \text{zint}(\text{zsize}(\text{dom } x)))$$

lemma *Rep-seqtype [simp]: Rep-seqtype(x): seqtype*
<proof>

lemma *insertseq-not-absorb-seqtype [simp]: s : seqtype ==> (insertseq a s ~ = s)*
<proof>

lemma *Rep-seqtype-inj [simp]: inj Rep-seqtype*
<proof>

lemma *Inv-f-f: inj(f) ==> (• xa. f xa = f x) = x*
<proof>

lemma *Rep-seqtype-inverse [simp]: Abs-seqtype(Rep-seqtype(x)) = x*
<proof>

lemma *List-of-seq-emptyseq: ((Abs-seqtype \%<%>) = [])*
<proof>

lemma *seq-Collect-Fin-Func:*

$$\begin{aligned} & !!x. \llbracket x : (\text{Naturals} \dashrightarrow \text{UNIV}) ; \\ & \quad (ka + 1) = (\text{card } x) ; ((\text{dom } x) = (1.. \ \text{zsuc } (\text{int } ka))) \rrbracket \implies \\ & \quad \{p. \ \text{EX } n: 1..(\text{int } ka). \ p = (n, \ x \% \wedge \ n + 1)\} : (\text{Naturals} \dashrightarrow \text{UNIV}) \end{aligned}$$

<proof>

Appendix B. Isabelle Theories

lemma *seq-Collect-Size*: $[[x : (\text{Naturals } -||-\> \text{UNIV}) ;$
 $((\text{Suc } ka) = (\text{card } x)) ; ((\text{dom } x) = (1.. \text{zsuc } (\text{int } ka)))] \implies$
 $(ka = \text{card } \{p. \text{EX } n: 1..\text{int } ka. p = (n, x \% ^n + 1::\text{int})\})$
 $\langle \text{proof} \rangle$

lemma *inj-Abs-seqtype2*:
 $!!x. x : \text{seqtype} \implies (\text{EX } y. x = \text{Rep-seqtype } y)$
 $\langle \text{proof} \rangle$

lemma *f-Inv-f*: $y : \text{range}(f) \implies f (\bullet x. f x = y) = y$
 $\langle \text{proof} \rangle$

lemma *Abs-seqtype-inverse* [simp]: $y : \text{seqtype} \implies (\text{Rep-seqtype } (\text{Abs-seqtype } y)) =$
 y
 $\langle \text{proof} \rangle$

lemma *seqtype-notempty* [simp]: $\text{seqtype} \sim = \{\}$
 $\langle \text{proof} \rangle$

lemma *ListSeqAbsIsTypeabs* [simp]: $\text{Typeabs } \text{ListSeqAbs}$
 $\langle \text{proof} \rangle$

lemma *Rep-seqtype-lemma*: $\text{Rep-seqtype} = \text{TARep } \text{ListSeqAbs}$
 $\langle \text{proof} \rangle$

lemma *Abs-seqtype-lemma*: $\text{Abs-seqtype} = \text{TAAbs } \text{ListSeqAbs}$
 $\langle \text{proof} \rangle$

lemma *seqtype-lemma*: $\text{seqtype} = \text{TASet } \text{ListSeqAbs}$
 $\langle \text{proof} \rangle$

lemma *TARep-LSAbs-Nil*: $\text{TARep } \text{ListSeqAbs } [] = \%<\%>$
 $\langle \text{proof} \rangle$

lemma *TARep-LSAbs-Cons*: $\text{TARep } \text{ListSeqAbs } (x\#xs) = \text{insertseq } x (\text{TARep } \text{ListSeqAbs } xs)$
 $\langle \text{proof} \rangle$

B.7. A rudimentary Theory-Morphism from List to Sequences

lemma *TAAbs-LSAbs-emptyseq*: $(TAAbs ListSeqAbs \%<\%>) = []$
 ⟨proof⟩

lemma *seqtype-emptyseq [simp]*: $\%<\%> : TASET ListSeqAbs$
 ⟨proof⟩

lemma *seq-seqtype*: $s: seq X ==> s: seqtype$
 ⟨proof⟩

lemma *dom-insertseq*: $(dom (insertseq a s)) =$
 $(insert 1 \{d . EX x: dom s. d = (x+1)\})$
 ⟨proof⟩

⟨ML⟩

lemma *insert-ran [simp]*: $ran (insert (a, b) R) = insert b (ran R)$
 ⟨proof⟩

lemma *ran-lem*:
 !!s. s: seq A ==>
 Range {p. ? n:Domain s. p = (zsuc n, s % ^ n)} = Range s
 ⟨proof⟩

lemma *ran-insertseq [simp]*:
 !!s. s: seq X ==> ran (insertseq a s) = (insert a (ran s))
 ⟨proof⟩

lemma *ran-insertseq-TAAbs [simp]*: $s: TASET ListSeqAbs ==> ran (insertseq a s)$
 $= (insert a (ran s))$
 ⟨proof⟩

lemma *concatseq-TypeAbs*: $(concatseq \% ^ (s,t)) =$
 $TAREP ListSeqAbs ((TAAbs ListSeqAbs s) \bullet (TAAbs ListSeqAbs t))$
 ⟨proof⟩

lemma *revseqseq-TypeAbs*: $(revseq s) = (TAREP ListSeqAbs (rev (TAAbs ListSeqAbs s)))$

Appendix B. Isabelle Theories

s)))
<proof>

lemma *last-TypeAbs*: (last s) = (List.last (TAAbs ListSeqAbs s))
<proof>

lemma *head-TypeAbs*: (head s) = (hd(TAAbs ListSeqAbs s))
<proof>

lemma *front-TypeAbs*: (front s) = (TARep ListSeqAbs (butlast(TAAbs ListSeqAbs s)))
<proof>

lemma *tail-TypeAbs*: (tail s) = (TARep ListSeqAbs (tl(TAAbs ListSeqAbs s)))
<proof>

lemma *filtering-TypeAbs*: (filtering% ^ (s, V)) = TARep ListSeqAbs (filter (% a. a : V) (TAAbs ListSeqAbs s))
<proof>

lemma *sizeseq-TypeAbs-seq*: s:seq X ==> (# s) = int (length (TAAbs ListSeqAbs s))
<proof>

lemma *sizeseq-TypeAbs*: !!s. s: TASET ListSeqAbs ==> (# s) = int (length (TAAbs ListSeqAbs s))
<proof>

lemma *insertseq-TypeAbs*: !!s. t:TASET ListSeqAbs ==> insertseq x t = TARep ListSeqAbs (x # (TAAbs ListSeqAbs t))
<proof>

lemma *ran-mem-TypeAbs-seq*: s:seq X ==> a : (ran s) = (a mem (TAAbs ListSeqAbs s))
<proof>

lemma *ran-mem-TypeAbs*:s: TASET ListSeqAbs ==> a : (ran s) = (a mem (TAAbs ListSeqAbs s))
<proof>

B.7. A rudimentary Theory-Morphism from List to Sequences

$\langle ML \rangle$

lemma *concatseq-assoc*: $s \% \& \wedge t \% \& \wedge u = s \% \& \wedge (t \% \& \wedge u)$
 $\langle proof \rangle$

lemma *concatseq-emptyseq*[*rule-format*]: $s : seq\ X \dashrightarrow \% \langle \% \rangle \% \& \wedge s = s$
 $\langle proof \rangle$

lemma *concatseq-emptyseq2*[*rule-format*]: $s : seq\ X \dashrightarrow s \% \& \wedge \% \langle \% \rangle = s$
 $\langle proof \rangle$

lemma *concatseq-is-empty*[*rule-format*]:
 $s : seq\ X \dashrightarrow t : seq\ Y \dashrightarrow$
 $(s \% \& \wedge t = \% \langle \% \rangle) = (s = \% \langle \% \rangle \ \& \ t = \% \langle \% \rangle)$
 $\langle proof \rangle$

lemma *same-concatseq-eq*[*rule-format*]:
 $s : seq\ X \dashrightarrow t : seq\ Y \dashrightarrow u : seq\ Z \dashrightarrow (s \% \& \wedge t = s \% \& \wedge u) = (t = u)$
 $\langle proof \rangle$

declare *concatseq-emptyseq* [*simp*] *concatseq-emptyseq2* [*simp*] *concatseq-is-empty* [*simp*]

lemma *revseq-revseq-ident*[*rule-format*]: $s : seq\ X \dashrightarrow (revseq\ (revseq\ s)) = s$
 $\langle proof \rangle$

lemma *revseq-concatseq*[*rule-format*]: $s : seq\ X \dashrightarrow t : seq\ Y \dashrightarrow$

Appendix B. Isabelle Theories

$(\text{revseq } (s \ \& \ ^\wedge \ t)) = ((\text{revseq } t) \ \& \ ^\wedge \ (\text{revseq } s))$
 $\langle \text{proof} \rangle$

lemma *size-revseq*[*rule-format*]: $s : \text{seq } X \longrightarrow \#(\text{revseq } s) = \#s$
 $\langle \text{proof} \rangle$

lemma *last-cons*[*rule-format*] :
 $t : \text{seq } X \longrightarrow$
 $(\text{last } (\text{insertseq } x \ (\text{insertseq } y \ t))) = (\text{last } (\text{insertseq } y \ t))$
 $\langle \text{proof} \rangle$

lemma *last-concatseq*[*rule-format*] :
 $t : \text{seq } X \longrightarrow s : \text{seq } Y \longrightarrow$
 $(\text{last } (s \ \& \ ^\wedge \ (\text{insertseq } y \ t))) = (\text{last } (\text{insertseq } y \ t))$
 $\langle \text{proof} \rangle$

lemma *last-single*: $(\text{last } (\text{insertseq } x \ \%<\%>)) = x$
 $\langle \text{proof} \rangle$

lemma *front-nt*: $\text{front}(\%<\%>) = \%<\%>$
 $\langle \text{proof} \rangle$

lemma *aux*:
 $(x : \text{seq } Z \ \& \ y : Z) \longrightarrow \text{front}(x \ \%& \ ^\wedge \ \%< y \%>) = x$
 $\langle \text{proof} \rangle$

B.7. A rudimentary Theory-Morphism from List to Sequences

lemma *front-snoc*:

!!x . [| x : seq Z; y : Z |] ==> front(x %& ^ %< y %>) = x
<proof>

declare *front-mt* [simp] *front-snoc* [simp]

lemma *TAAbs-LSA*: $TAAbs\ ListSeqAbs = (\%u. \bullet x. (TARep\ ListSeqAbs)\ x = u)$
<proof>

lemma *TASet-LSA*: $TASet\ ListSeqAbs = seq\ UNIV$
<proof>

lemma *TARep-in-seqUNIV*: $TARep\ ListSeqAbs\ l : seq\ UNIV$
<proof>

lemma *TAAbs-LSA-cons*: $(x\#\!xs) = TAAbs\ ListSeqAbs\ (insertseq\ x\ (TARep\ ListSeqAbs\ xs))$
<proof>

lemma *insertseq-concat*: $a : X \dashrightarrow s : seq\ X \dashrightarrow t : seq\ X \dashrightarrow$
 $(insertseq\ a\ s)\ \%&^{\wedge}\ t = insertseq\ a\ (s\ \%&^{\wedge}\ t)$
<proof>

declare *insertseq-concat* [simp]

Appendix B. Isabelle Theories

lemma *concatseq-closed*:

!!a. [[a : seq G; b : seq G]] ==> (a %& ^ b) : seq G

<proof>

declare *concatseq-closed* [simp]

lemma *snoc-closed*:

!!a. [[s : seq G; b : G]] ==> (s %& ^ %<b%>) : seq G

<proof>

declare *snoc-closed* [simp]

declare *seq-card-dom* [simp]

lemma *aux*: (int m < int n) = (m < n)

<proof>

lemmas *snoc-wf* = *snoc-wf-lem* [THEN *wf-measure* [THEN *wf-subset*]]

lemma *snocE-lem*:

!! s. s : seq A

==> (? t : seq A. ? a:A. s = t %& ^ %<a%>) | s = %<%>

<proof>

lemma *seqsnoc-cases*:

assumes *p1*: s:seq A

assumes *p2*: s = %<%> ==> P s

assumes *p3*: !!a s'. [[s' : seq A; a:A; s = s' %& ^ %<a %>] ==> P s

shows P s

<proof>

lemma *seq-snocinduct*:

assumes *p1*: s:seq A

assumes *p2*: P %<%>

B.8. Bags from the Mathematical Toolkit

```
assumes p3: !! a s. [| s : seq A; a:A; P s|] ==> P(s %& ^ %< a %>)  
shows P s  
⟨proof⟩
```

```
declare snoc-neq-empty [simp] snoc-neq-empty [THEN not-sym, simp]
```

```
lemma snoc-case-mt: snoc-case A a f %<%> = a  
⟨proof⟩
```

```
lemma snoc-case-conc:  
!!s. [| s : seq A; x : A |] ==> snoc-case A a f (s %& ^ %<x %>) = f x s  
⟨proof⟩
```

```
declare snoc-case-mt [simp] snoc-case-conc [simp]
```

```
lemma front-closed:  
!!a. [| a : seq G|] ==> front(a) : seq G  
⟨proof⟩  
declare front-closed [simp]
```

end

B.8. Bags from the Mathematical Toolkit

```
theory ZBag imports ZSeqtoList begin
```

```
types
```

```
('a) bag = 'a <=> int
```

```
constdefs
```

```
bag :: 'a set => ('a bag) set  
bag S == S -|-> Naturals-1
```

```
emptybag :: 'a bag                                (%[%])  
%[%] == {}
```

```
insertbag :: ['a, 'a bag] => 'a bag
```

Appendix B. Isabelle Theories

```

insertbag x B == (if x : dom B
                  then (B (+) {(x, 1 + (B % ^ x))})
                  else B Un {(x,1)})

count :: 'a bag <=> ('a <=> int)
count == {(B,c). c = ((lambda x:UNIV . 0) (+) B)}

infixcount :: ['a bag, 'a] => int          ((- [#]/ -) [70,71]70)
B [#] x == count % ^ B % ^ x

bagelem :: 'a <=> ('a bag)
bagelem == {(x,B). x : dom B}

bagelem-fun :: ['a, 'a bag] => bool       ((- [:/ -) [60,61]60)
x [: B == (x,B) : bagelem

bagunion :: ('a bag * 'a bag) <=> 'a bag
bagunion == {(B,C,D). dom D = dom B Un dom C &
              (ALL x. D [#] x = B [#] x + C [#] x)}

bagunion-fun
:: ['a bag, 'a bag] => 'a bag          ((- BUn/ -) [70,71]70)
B BUn C == bagunion % ^ (B,C)

itemsbag :: ('a seq) <=> ('a bag)
itemsbag == {(S,B). ALL x. (B : (bag UNIV)) &
                           (B [#] x = # {i. (i: dom S) &
                           ((S % ^ i) = x)})}

items :: ('a seq) => ('a bag)
items S == (itemsbag % ^ S)

syntax
• Bag :: args => 'a bag                (%[ (-) %])

translations
%[ x, xs %] == insertbag x %[xs%]
%[ x %] == insertbag x %[%]

axioms
dom-ran-item: t.seq X --> ((dom (items t)) = (ran t))

```

B.8. Bags from the Mathematical Toolkit

mem-bagunion: $(A : (\text{bag } X)) \dashrightarrow (B : (\text{bag } Y)) \dashrightarrow$
 $(x [: (A \text{ BUn } B)) = ((x [: A] \mid (x [: B]))$
bagunion-not-emptybag: $(B \sim = \%[\%]) \dashrightarrow ((A \text{ BUn } B) \sim = \%[\%])$
bagunion-is-bag: $(A : (\text{bag } X)) \dashrightarrow (B : (\text{bag } X)) \dashrightarrow ((A \text{ BUn } B) : (\text{bag } X))$
items-concatseq: $(S : (\text{seq } X)) \dashrightarrow (T : (\text{seq } X)) \dashrightarrow$
 $((\text{items } (S \% \& \wedge T)) = ((\text{items } S) \text{ BUn } (\text{items } T)))$
bagunion-emptybag: $(A : (\text{bag } X)) \dashrightarrow ((\%[\%] \text{ BUn } A) = A)$
bagunion-emptybag2: $(A : (\text{bag } X)) \dashrightarrow ((A \text{ BUn } \%[\%]) = A)$
count-bagunion: $(A : (\text{bag } X)) \dashrightarrow (B : (\text{bag } X)) \dashrightarrow$
 $((\text{count } (A \text{ BUn } B) [\#] x) = (\text{count } A [\#] x + \text{count } B [\#] x))$
items-emptybag: $\text{items } \% \langle \% \rangle = \%[\%]$
items-singletonbag: $\text{items } \% \langle x \% \rangle = \% [x \%]$
items-mem-ran: $(s : (\text{seq } X)) \dashrightarrow ((x [: (\text{items } s)) = (x : (\text{ran } s)))$
dom-bagunion: $A : \text{bag } X \dashrightarrow B : \text{bag } X \dashrightarrow$
 $(\text{dom } (A \text{ BUn } B)) = ((\text{dom } A) \text{ Un } (\text{dom } B))$
domsubstr-bagunion: $A : \text{bag } X \dashrightarrow B : \text{bag } X \dashrightarrow \text{dom } B \leq M \dashrightarrow$
 $((M \leftarrow (A \text{ BUn } B)) = (M \leftarrow A))$

declare *bagunion-not-emptybag* [simp] *bagunion-is-bag* [simp]
items-singletonbag [simp]
items-concatseq *bagunion-emptybag* *bagunion-emptybag2*
dom-bagunion *domsubstr-bagunion*

B.8.1. Basic Theorems

lemma *BagDomPow* [simp]: $x : \text{bag } X \dashrightarrow \text{dom } x : (\text{Pow } X)$
 <proof>

lemma *BagDomMem* [simp]: $(a [: B) = (a : (\text{dom } B))$
 <proof>

lemma *BagSingleton*: $\% [x \%] = \{(x, 1)\}$
 <proof>

lemma *BagEmpty*[simp]: $\%[\%] : (\text{bag } X)$
 <proof>

lemma *SizeBagEmpty*[simp]: $\# \%[\%] = 0$
 <proof>

declare *SizeBagEmpty* [simp]

lemma *SizeBagSingleton*[simp]: $\# \% [x \%] = 1$

Appendix B. Isabelle Theories

$\langle proof \rangle$

lemma *NotBagEmptyEqualSingleton* [simp]: $\%[x\ \%] \sim = \%[\%]$

$\langle proof \rangle$

lemma *NotBagEmptyEqualSingleton2* [simp]: $\%[\%] \sim = (\%[x\ \%])$

$\langle proof \rangle$

lemma *BagSingletonEqual* [simp]: $(\%[x\ \%] = \%[y\ \%]) = (x = y)$

$\langle proof \rangle$

lemma *DomBagEmpty* [simp]: $dom\ \%[\%] = \{\}$

$\langle proof \rangle$

lemma *DomBagSingleton* [simp]: $dom\ \%[x\ \%] = \{x\}$

$\langle proof \rangle$

lemma *NotMemBagEmpty* [simp]: $\sim x\ [: \%[\%]$

$\langle proof \rangle$

lemma *MemBagSingleton* [simp]: $(x\ [: \%[y\ \%]) = (x = y)$

$\langle proof \rangle$

lemma *CountBagEmpty* [simp]: $\%[\%] [\#] x = 0$

$\langle proof \rangle$

lemma *count-bagsingleton*:

$\%[x\ \%] [\#] y = (if\ x=y\ then\ 1\ else\ 0)$

$\langle proof \rangle$

lemma *single-is-bag* [rule-format,simp]:

$((fst\ a) : X) \dashrightarrow ((snd\ a) : Naturals-1) \dashrightarrow (\{a\} : (bag\ X))$

$\langle proof \rangle$

lemma *single-is-bag2* [rule-format,simp]:

$(a : G) \dashrightarrow (\%[a\ \%] : (bag\ G))$

$\langle proof \rangle$

lemma *single-bagset*[rule-format,simp]:

$(a = (fst\ b)) \dashrightarrow ((snd\ b) = 1) \dashrightarrow (\%[a\ \%] = \{b\})$

B.8. Bags from the Mathematical Toolkit

<proof>

lemma *not-mem-emptybag* [simp]: $a \sim: \%[\%]$

<proof>

B.8.2. Bag Union

lemma *bagunion-emptybag-singlebag* [simp]: $(\%[\%] \text{ BUn } \%[a \%]) = \%[a \%]$

<proof>

lemma *bagunion-emptybag-singlebag2* [simp]: $(\%[a \%] \text{ BUn } \%[\%]) = \%[a \%]$

<proof>

lemma *domsubstr-bagunion-singlebag* [rule-format,simp]:

$(i : G) \dashrightarrow A : \text{bag } G \dashrightarrow (\{i\} <-: (A \text{ BUn } \%[i \%])) = (\{i\} <-: A)$

<proof>

lemma *dom-BagSingleton-I*:

$[|x = a|] \implies x : \text{dom } (\%[a \%])$

<proof>

end

theory *ProdContrib*

imports *Main*

begin

lemma *split-paired-Lam*: $P = (\lambda (a,b). P (a,b))$

<proof>

Appendix B. Isabelle Theories

lemma *split-paired-split*: $(\% x. \text{split } f \ x) = (\% (a,b). \text{split } f \ (a,b))$
<proof>

lemma *split-paired-CollectL*: $\{(a,b). P \ a \ b\} = \{((a,c),b). P \ (a,c) \ b\}$
<proof>

lemma *split-pattern*: **assumes** $a: z=(x,y)$ **shows** $(\% (x,y). F \ x \ y) \ z = F \ x \ y$
<proof>

lemma *split-paired-Ex*: $(EX \ x. P \ x) = (EX \ a \ b. P \ (a,b))$
<proof>

<ML>

declare *split-paired-Ex* [*simp*]

<ML>

lemma *METACOND* $x ==> ((x::\text{unit}),y) = ((),y)$
<proof>

lemma *METACOND-pair-snd-unit*: $METACOND \ y ==> (x,(y::\text{unit})) = (x,())$
<proof>

lemma *unit-eq-true*: $((x::\text{unit}) = y) = \text{True}$
<proof>

declare *unit-eq-true* [*simp*]

end

B.9. Definitions of Syntax and Core-Combinators of Z

```
theory ZPure
imports contrib/ProdContrib
uses ZPrelude.sml
     ZAbsy.sml
     ZEnv.sml
     ZTheory.sml
     ZEncoder.sml
     ZConversion.sml
```

```
begin
```

B.9.1. Initialize Z-Environment, provide Access

<ML>

B.9.2. Syntax

```
types Zschema = bool
```

```
syntax
```

```
SSTROKE :: idt => idt          (-'')
-IN      :: idt => idt          (-?)
-OUT     :: idt => idt          (-!)
SSTROKE :: idt => logic        (-'')
-IN      :: idt => logic        (-?)
-OUT     :: idt => logic        (-!)

%Delta  :: idt => Zschema      ((%Delta -) [100]5)
%Xi     :: idt => Zschema      ((%Xi -) [100]5)
%theta  :: 'a => Zschema       ((%theta -) [100]5)
```

Appendix B. Isabelle Theories

syntax (*xsymbols*)

```
%Delta :: idt => Zschema (( $\Delta$  -) [100]5)
%Xi    :: idt => Zschema (( $\Xi$  -) [100]5)
%theta :: 'a  => Zschema (( $\vartheta$  -) [100]5)
```

nonterminals

```
rnms
rnm
```

syntax

```
-rnm    :: [idt , idt] => rnm          ((- '/' -) )
         :: rnm => rnms                (-)
-rnms   :: [rnm, rnms] => rnms        (-, / -)
%not    :: Zschema => Zschema         (%not (-)// [40]40)
\ '/'   :: [Zschema, Zschema] => Zschema (infixl 30)
'\      :: [Zschema, Zschema] => Zschema (infixl 35)
==+==>  :: [Zschema, Zschema] => Zschema (infixl 25)
<==+==> :: [Zschema, Zschema] => Zschema (infixr 25)
pre     :: Zschema => Zschema
%E     :: [Zschema, Zschema] => Zschema (%E (2-) • (-) [30,31]30)
%E1    :: [Zschema, Zschema] => Zschema (%E1 (3-) • (-) [30,31]30)
%A     :: [Zschema, Zschema] => Zschema (%A (2-) • (-) [30,31]30)
-hiding :: [Zschema, args] => Zschema ((-)\[(-)] [80,81]80)
-geninst:: [idt, args] => Zschema ((-)[(-)] [80,81]80)
-rename :: [idt, rnms] => Zschema ((-)\[(-)] [80,81]80)
-schset :: [idt, 'a] => 'b set ((1\{(-) • (-).})
-select :: ['a, idt] => 'c (-, - [200,201]200)
|\      :: [Zschema, Zschema] => Zschema (infixl 80)
%%;    :: [Zschema, Zschema] => Zschema (infixl 80)
>>    :: [Zschema, Zschema] => Zschema (infixl 80)
```

syntax (*xsymbols*)

```
%E     :: [Zschema, Zschema] => Zschema ( $\exists$  (2-) • (-) [30,31]30)
```

B.9. Definitions of Syntax and Core-Combinators of Z

$\%E1$:: [*Zschema*, *Zschema*] => *Zschema* ($\exists!$ (3-) • (-) [30,31]30)
 $\%A$:: [*Zschema*, *Zschema*] => *Zschema* (\forall (2-) • (-) [30,31]30)

translations

$\%not\ P$ => $\sim P$
 $A \setminus B$ => $A \mid B$
 $A \wedge B$ => $A \& B$
 $A =+=> B$ => $A \dashrightarrow B$
 $A <+=> B$ => $A = B$

types *Predicate* = *bool*
DeclPart = *bool*

nonterminals

DeclParts
SchemaName

syntax

$\%DeclPart$:: *DeclPart* => *DeclParts* (-)
 $\%combdecls$:: [*DeclPart*, *DeclParts*] => *DeclParts* (-; / -)

consts

$\%SCHEMA$:: [*bool*, *bool*] => *Zschema*

translations

$\%combdecls\ D\ D1$ => $D \& D1$

syntax

$\%AXIOM0$:: *Zschema*
 (+.. // -..)
 $\%AXIOM1$:: *DeclParts* => *Zschema*
 (+.. // (2 -) // -..)
 $\%AXIOM2$:: *Predicate* => *Zschema*

Appendix B. Isabelle Theories

```
(+.. //|-- //(2 -)//-..)
-AXIOM3 :: [DeclParts, Predicate] => Zschema
(+..//(2 -)//|--//(2 -)//-..)
```

translations

```
-AXIOM0 == SCHEMA True True
-AXIOM1 D == SCHEMA D True
-AXIOM2 P == SCHEMA True P
-AXIOM3 D P == SCHEMA D P
```

syntax

```
-GEN0 :: args => Zschema
      (+== [(-) ]===-//-==)
-GEN1 :: [args, DeclParts]=> Zschema
      (+== [(-) ]===-//(2 -) //=-==)
-GEN2 :: [args, Predicate]=> Zschema
      (+== [(-) ]===-//|--//(2 -)//=-==)
-GEN3 :: [args, DeclParts, Predicate] => Zschema
      (+== [(-) ]===-//(2 -)//|--//(2 -)//=-==)
```

translations

```
-GEN0 M == %M. (-AXIOM0)
-GEN1 M D == %M. (-AXIOM1 D)
-GEN2 M P == %M. (-AXIOM2 P)
-GEN3 M D P == %M. (-AXIOM3 D P)
```

syntax

```
-SCHEME0:: SchemaName => Zschema      (+--- - ---- //---- )
-SCHEME1::[SchemaName, args] => Zschema  (+--- - [-] ----//----)
-SCHEME2::[SchemaName, DeclParts] => Zschema      (+--- - ----//(2
-)//----)
-SCHEME3::[SchemaName, Predicate] => Zschema      (+--- - ----//|--//(2
-)//----)
-SCHEME4::[SchemaName, args, DeclPart] => Zschema  (+--- - [-] ----//(2
-)//----)
-SCHEME5::[SchemaName, args, Predicate]=> Zschema (+--- - [-] ----//|--//(2
-)//----)
-SCHEME6::[SchemaName, DeclParts,Predicate]=>Zschema(+--- - ----//(2 -)//|--//(2
-)//----)
-SCHEME7::[SchemaName, args, DeclParts, Predicate]=>Zschema (+--- - [-]
```

B.9. Definitions of Syntax and Core-Combinators of Z

----//(2 -)//|----//(2 -)//----

translations

-SCHEME0 N == (prop) N == -AXIOM0
 -SCHEME1 N M == (prop) N == (-GEN0 M)
 -SCHEME2 N D == (prop) N == (-AXIOM1 D)
 -SCHEME3 N P == (prop) N == (-AXIOM2 P)
 -SCHEME4 N M D == (prop) N == (-GEN1 M D)
 -SCHEME5 N M P == (prop) N == (-GEN2 M P)
 -SCHEME6 N D P == (prop) N == (-AXIOM3 D P)
 -SCHEME7 N M D P == (prop) N == (-GEN3 M D P)

B.9.3. Semantic Representation

consts

SNAME0 :: 'a => 'a
 SNAME :: ['a=>'b,'a] => 'b
 SBinder0 :: ['a, 'b => 'd] => ('b => 'd)
 SBinder :: ['a,['b,'c] => 'd] => (('b*'c)=>'d)

asSet :: ['a => bool] => 'a set
 asPred :: ['a set] => 'a => bool
 turnstyle :: ('a => bool) => bool (|- (3-))
 theta :: [string,'a] => 'a
 DELTA :: [bool,bool] => bool
 XI :: [bool,'a,'a] => bool
 PROJ :: ['a,'a=>'b,'c]=>'b
 DECL :: [bool,bool] => bool ((0-|-----/-) [15,14]14)
 PRE :: ['a => bool] => bool
 SBall :: ['a set, 'a => bool] => bool
 SBex :: ['a set, 'a => bool] => bool
 SBex1 :: ['a set, 'a => bool] => bool
 SSet :: ['a set, 'a => 'b] => 'b set

defs

SNAME0-def: SNAME0 x == x
 SNAME-def: SNAME f args == f args
 SBinder0-def: SBinder0 An P == P
 SBinder-def: SBinder An P == (% (x,y). (P x y))
 asSet-def: asSet == Collect

Appendix B. Isabelle Theories

```

asPred-def:  asPred      == (%X. %x. x : X)
turnstyle-def: |- P      == All P
PROJ-def:    PROJ X F ide == F X
DELTA-def:   DELTA s t   == s & t
XI-def:      XI s t t'   == s & t=t'
theta-def:   theta s t   == t
DECL-def:    DECL P Q    == P & Q
PRE-def:     PRE          == Ex
SBall-def:   SBall       == Ball
SBex-def:    SBex        == Bex
SBex1-def:   SBex1 A P   == ?! x. x : A & P x
SSet-def:    SSet A f    == f ' A

```

nonterminals

```

sbind
sbinds

```

syntax

```

-sbind :: ['a , 'b] => sbind      ((?- ~> -) 10)
      :: sbind => sbinds         (-)
-sbinds :: [sbind, sbinds] => sbinds  (-,/ -)
-SB     :: [sbinds, 'a] => 'a       ((4SB (-). (-) 10)

```

translations

```

%Delta S => DELTA S (SSTROKE S)
%Delta S <= DELTA S T
%Xi S => XI (%Delta S) (%theta S) (%theta (SSTROKE S))
%Xi S <= XI (%Delta S) (T) (U)

```

```

-SB (-sbinds (-sbind x y) sbds) e == SBinder x (%y. -SB sbds e)
-SB (-sbind x y) e                == SBinder0 x (%y. e)

```

```

pre S <= PRE S
%E S • T <= SBex S T
%E1 S • T <= SBex1 S T
%A S • T <= SBall S T
{.S • T.} <= SSet S T

```

B.9.4. Syntax: The parse-translation setup

$\langle ML \rangle$

B.9.5. Derived Rules

Schema Binders

lemma *SB0-ext*: $(\bigwedge x. f x = g x) \implies SBinder0 a f = SBinder0 b g$
 $\langle proof \rangle$

lemma *SB-ext*: $(\bigwedge x. f x = g x) \implies SBinder a f = SBinder b g$
 $\langle proof \rangle$

lemma *conv-sname0* : $f \equiv g \implies SNAME0 f = g$
 $\langle proof \rangle$

lemma *conv-sname* : $f \equiv g \implies SNAME f args = g args$
 $\langle proof \rangle$

lemma *turnstyle-SB0*: $(\bigwedge x. f x) \implies |- SBinder0 a f$
 $\langle proof \rangle$

lemma *turnstyle-SB*: $(\bigwedge x. |- f x) \implies |- SBinder a f$
 $\langle proof \rangle$

lemma *SB0-ext-id*: $(\bigwedge x. f x = g x) \implies SBinder0 a f = SBinder0 a g$
 $\langle proof \rangle$

lemma *SB-ext-id*: $(\bigwedge x. f x = g x) \implies SBinder a f = SBinder a g$
 $\langle proof \rangle$

lemma *SB0-extL*: $(\bigwedge x. f x = g x) \implies SBinder0 a f = g$
 $\langle proof \rangle$

lemma *SB-extL*: $(\bigwedge x y. f x y = g (x, y)) \implies SBinder a f = g$
 $\langle proof \rangle$

lemmas *Zexts* = *ext SB0-ext SB-ext*

lemma *SB0-eq*: $SBinder0 a P x = P x$
 $\langle proof \rangle$

Appendix B. Isabelle Theories

lemma *SB-eq*: $SBinder\ a\ P\ z = (P\ (fst\ z)\ (snd\ z))$
<proof>

lemma *SB-pair-eq*: $SBinder\ a\ P\ (x,y) = (P\ x\ y)$
<proof>

lemma *EX-SBinder0*: $(\exists\ x.\ (SBinder0\ a\ P)\ x) = (\exists\ x.\ P\ x)$
<proof>

lemma *EX-SBinder*: $(\exists\ x.\ (SBinder\ a\ P)\ x) = (\exists\ x\ y.\ P\ x\ y)$
<proof>

declare *SB0-eq SB-eq EX-SBinder0 EX-SBinder[simp]*

DECL

lemma *DECL-I*: $\llbracket A; A \implies B \rrbracket \implies A \mid \text{-----} B$
<proof>

lemma *DECL-D1*: $A \mid \text{-----} B \implies A$
<proof>

lemma *DECL-D2*: $A \mid \text{-----} B \implies B$
<proof>

lemma *DECL-E*: $\llbracket P \mid \text{-----} Q; \llbracket P; Q \rrbracket \implies R \rrbracket \implies R$
<proof>

lemma *DECL-cong*:
 $\llbracket P = P'; P' \implies Q = Q' \rrbracket \implies (P \mid \text{-----} Q) = (P' \mid \text{-----} Q')$
<proof>

declare *DECL-I [intro!]*

declare *DECL-E [elim!]*

declare *DECL-cong [cong]*

Tests:

<ML>

lemma *SBall-I*: $(\bigwedge x.\ x \in A \implies P\ x) \implies SBall\ A\ P$
<proof>

lemma *SBall-I2*: $(\bigwedge x.\ A\ x \implies P\ x) \implies SBall\ (asSet\ A)\ P$
<proof>

B.9. Definitions of Syntax and Core-Combinators of Z

lemma bla : $(\bigwedge z y . x=(y,z) \implies f y z) \implies (SBinder a f) x$
<proof>

lemma $bla0$: $(\bigwedge y . x=y \implies f y) \implies (SBinder0 a f) x$
<proof>

lemma bla' : $(SBinder a f) x \implies (\exists y z . f y z)$
<proof>

lemma $bla0'$: $(SBinder0 a f) x \implies f x$
<proof>

lemma $blaE$: $(\exists y z . f y z \wedge x=(y,z)) \implies (SBinder a f) x$
<proof>

lemma $bla0E$: $(f x) \implies (SBinder0 a f) x$
<proof>

lemma $PRE-I$: $\llbracket P x \rrbracket \implies PRE P$
<proof>

lemma $PRE-E$: $\llbracket PRE P; \bigwedge x . P x \implies Q \rrbracket \implies Q$
<proof>

lemma $SBex-E1$: $\llbracket SBex A P; \bigwedge x . \llbracket x : A; P x \rrbracket \implies Q \rrbracket \implies Q$
<proof>

lemma $SBex-E2$: $\llbracket SBex (asSet A) P; \bigwedge x . \llbracket A x; P x \rrbracket \implies Q \rrbracket \implies Q$
<proof>

lemma $SBex-I1$: $\llbracket P x; x : A \rrbracket \implies SBex A P$
<proof>

lemma $SBex-I2$: $\llbracket P x; A x \rrbracket \implies SBex (asSet A) P$
<proof>

lemma $turnstyle-E$: $\mid- P \implies P x$
<proof>

Appendix B. Isabelle Theories

lemma *SBall-E* : $\llbracket \text{SBall } A \ P; \ x : A \rrbracket \implies P \ x$
<proof>

B.9.6. Proof Support: Z-tactics

lemmas *Z2HOL* = *SNAME0-def SNAME-def SBinder0-def SBinder-def turnstyle-def*

asSet-def asPred-def DECL-def theta-def PROJ-def
DELTA-def XI-def PRE-def SBall-def SBex-def SBex1-def

lemmas *set-simps* = *insert-subset*
insert-not-empty empty-not-insert
Int-absorb Int-empty-left Int-empty-right
Un-absorb Un-empty-left Un-empty-right Un-empty
UN-empty UN-insert image-empty
Compl-disjoint double-complement
Union-empty Union-insert empty-subsetI subset-refl
Diff-cancel empty-Diff Diff-empty Diff-disjoint

lemmas *prod-simps* = *fst-conv snd-conv split Pair-eq*

ML Code

<ML>

end

B.10. Method Package for Z Representation and Z Refinement

theory *ZMethod*
imports *ZPure*
uses *ProofObligationMgr.sml*
begin

B.10.1. Configuring the Generic PO Manager

<ML>
Initialize Environment
<ML>

B.10.2. The Z Method Package

On top of the generic PO Manager, which has now been imported and instantiated, we define a number of Method- specific PO generation methods. In our case, we implement support for classical forward data refinement (with an optimizing option for functional abstraction relations) following the lines described in the Spivey-Book also described in the Woodcock/Davis Book "Using Z".

There is a general configuration operation `set_abs` that sets the underlying abstraction relation and concepts such as abstract or concrete state.

PO generators are in particular:

1. `gen_state_cc` for checking that state schemas (representing system invariants) are in fact satisfiable.
2. `gen_op_cc` for checking that operation schemas are "implementable" or "non-blocking", i.e. there exists a function that maps inputs and state to outputs and successor state. (Note that this function is **not** necessarily computable).
3. `gen_thm_tcc` extracts from an arbitrary definition or axiom the side-conditions for type-constraint-consistency. (a representational side-condition of HOL-Z).
4. `declare_abs` allows for setting the abstraction relation (per schema definition). An additional option [`functional`] specifies the relation to be functional, which results in different (usually simpler) proof obligations.
5. `refine_init` for generating a PO assuring compatibility of Init schemas (in forward simulation)
6. `refine_op` for generating two PO's establishing refinement of operation schemas.

`<ML>`

`end`

B.11. Main Theory for the HOL-Z Environment

```
theory Z
imports ZSeqtoList
        ZBag
```

Appendix B. Isabelle Theories

ZPure
ZMethod

begin

B.11.1. Proof Support: The .holz-loader (generated by ZETA)

<ML>

B.11.2. Proof Support: Toplevel-Command zlemma

<ML>

B.11.3. Baustelle: Code von DARMA zum integrieren

lemmas *dom-infers* = *dom-override-I dom-Un-I dom-insert-I1 dom-insert-I2*
dom-dres-I dom-BagSingleton-I

lemmas *type-infers* = *SigmaI subsetI [THEN PowI]*
pair-rel-dom-fst pair-rel-dom

declare *UNIV-I [tc-simp]*
declare *dom-infers [tc-simp]*
declare *type-infers [tc-simp]*

<ML>

lemmas *oplus-def* = *override-def*
lemmas *oplus-I2* = *overrideI2*
lemmas *oplus-I1* = *overrideI1*
lemmas *oplus-CI* = *overrideCI*
lemmas *oplus-single* = *override-single*
lemmas *oplus-res-right* = *override-res-right*
lemmas *oplus-res-left* = *override-res-left*
lemmas *oplus-Inter* = *override-inter*
lemmas *oplus-idem* = *override-idem*
lemmas *oplus-mt-right* = *override-mt-right*
lemmas *oplus-mt-left* = *override-mt-left*

B.11. Main Theory for the HOL-Z Environment

lemmas *oplus-Domain* = *override-Domain*
lemmas *oplus-comp* = *override-assoc*
lemmas *oplus-fpfun* = *override-fpfun*
lemmas *oplus-pfunI* = *override-pfunI*

lemmas *oplus-apply2* = *override-apply2*
lemmas *oplus-apply* = *override-apply*

lemmas *oplus-apply1* = *override-apply1*
lemmas *oplus-non-apply* = *override-by-pair-apply2*
lemmas *oplus-by-pair-apply1* = *override-by-pair-apply1*
lemmas *oplus-by-pair-apply2* = *override-by-pair-apply2*

lemmas *Rel-Apply-in-total-range* = *tfun-apply*
lemmas *partial-func-def* = *pfun-def*
lemmas *total-func-def* = *tfun-def*

lemmas *pair-mem-apply* = *beta-apply-pfun*
lemmas *Rel-Apply-in-Partial-Ran2* = *pfun-apply*
lemmas *total-func-implies-Pfun* = *tfun-implies-pfun*

lemmas *partial-fun-ran-subset* = *pfun-ran-subset*

lemmas *partial-fun-dom-subset* = *pfun-subset*

lemmas *dom-oplus-I* = *dom-override-I*

lemmas *empty-pfun* = *empty-is-pfun*

lemmas *dom-insert-apply* = *insert-apply*
lemmas *dom-insert-apply2* = *insert-apply2*

end