```
booleans : THEORY
  BEGIN

    boolean : TYPE+

    bool : TYPE+ = boolean

    FALSE, TRUE : bool

    NOT, ¬: [bool → bool]

    AND, &, ∧, OR, ∨, IMPLIES, ⇒, ⇒, WHEN, IFF, ⇔, ⇔: [bool, bool → bool]

  END booleans
```

equalities[T: TYPE]: THEORY

BEGIN

=: [T, T → boolean]

END equalities

```
notequal[T: TYPE]: THEORY
  BEGIN

    x, y: VAR T

    ≠(x, y): boolean = NOT (x = y);

    ≠: [T, T → bool] = ≠

  END notequal
```

```
if_def[T: TYPE]: THEORY
  BEGIN

    IF: [boolean, T, T -> T]

  END if_def
```

```
boolean_props : THEORY
  BEGIN

    A, B : VAR bool

    bool_exclusive : POSTULATE NOT (FALSE = TRUE)

    bool_inclusive : POSTULATE A = FALSE OR A = TRUE

    not_def : POSTULATE (NOT A) = IF A THEN FALSE ELSE TRUE ENDIF

    and_def : POSTULATE (A AND B) = IF A THEN B ELSE FALSE ENDIF

    syand_def : POSTULATE & = AND

    or_def : POSTULATE (A OR B) = IF A THEN TRUE ELSE B ENDIF

    implies_def : POSTULATE (A IMPLIES B) = IF A THEN B ELSE TRUE ENDIF

    syimplies_def : POSTULATE => = IMPLIES

    when_def : POSTULATE (A WHEN B) = (B IMPLIES A)

    iff_def : POSTULATE (A IFF B) = ((A AND B) OR (NOT A AND NOT B))

    syiff_def : POSTULATE <=> = IFF

    excluded_middle : LEMMA A OR NOT A

  END boolean_props
```

```
xor_def : THEORY
 BEGIN

  A, B : VAR bool

  XOR(A, B) : bool = (A ≠ B)

  xor_def : LEMMA (A XOR B) = IF A THEN NOT B ELSE B ENDIF

 END xor_def
```

```
quantifier_props[t: TYPE]: THEORY
 BEGIN

  x: VAR t

  p, q: VAR [t -> bool]

  not_exists: LEMMA (∃ x: p(x)) = NOT (∀ x: NOT p(x))

  exists_not: LEMMA (∃ x: NOT p(x)) = NOT (∀ x: p(x))

  exists_or: LEMMA
    (∃ x: p(x) OR q(x)) =
     ((∃ x: p(x)) OR (∃ x: q(x)))

  exists_implies: LEMMA
    (∃ x: p(x) IMPLIES q(x)) =
     ((∃ x: NOT p(x)) OR (∃ x: q(x)))

  exists_and: LEMMA
    (∃ x: p(x) AND q(x)) IMPLIES
     ((∃ x: p(x)) AND (∃ x: q(x)))

  not_forall: LEMMA (∀ x: p(x)) = NOT (∃ x: NOT p(x))

  forall_not: LEMMA (∀ x: NOT p(x)) = NOT (∃ x: p(x))

  forall_and: LEMMA
    (∀ x: p(x) AND q(x)) =
     ((∀ x: p(x)) AND (∀ x: q(x)))

  forall_or: LEMMA
    ((∀ x: p(x)) OR (∀ x: q(x))) IMPLIES
     (∀ x: p(x) OR q(x))

 END quantifier_props
```

```
defined_types[t: TYPE]: THEORY
 BEGIN

  pred: TYPE = [t -> bool]

  PRED: TYPE = [t -> bool]

  predicate: TYPE = [t -> bool]

  PREDICATE: TYPE = [t -> bool]

  setof: TYPE = [t -> bool]

  SETOF: TYPE = [t -> bool]

 END defined_types
```

```
exists1[T: TYPE]: THEORY
 BEGIN

  x, y: VAR T

  p, q: VAR pred[T]

  unique?(p): bool = ∀ x, y: p(x) AND p(y) IMPLIES x = y

  exists1(p): bool = (∃ x: p(x)) AND unique?(p)

  unique_lem: LEMMA
    (∀ x: p(x) IMPLIES q(x)) IMPLIES (unique?(q) IMPLIES unique?(p))

  exists1_lem: LEMMA (exists1! x: p(x)) IMPLIES (∃ x: p(x))

 END exists1
```

```
equality_props[T: TYPE]: THEORY
 BEGIN

  x, y, z: VAR T

  b: VAR bool

  IF_true: POSTULATE IF TRUE THEN x ELSE y ENDIF = x

  IF_false: POSTULATE IF FALSE THEN x ELSE y ENDIF = y

  IF_same: LEMMA IF b THEN x ELSE x ENDIF = x

  reflexivity_of_equals: POSTULATE x = x

  transitivity_of_equals: POSTULATE x = y AND y = z IMPLIES x = z

  symmetry_of_equals: POSTULATE x = y IMPLIES y = x

 END equality_props
```

```
if_props[s, t: TYPE]: THEORY
 BEGIN

  a, b, c: VAR bool

  x, y: VAR s

  f: VAR [s -> t]

  lift_if1: LEMMA
    f(IF a THEN x ELSE y ENDIF) =
     IF a THEN f(x) ELSE f(y) ENDIF

  lift_if2: LEMMA
    IF (IF a THEN b ELSE c ENDIF) THEN x ELSE y ENDIF =
     IF a
       THEN (IF b THEN x ELSE y ENDIF)
     ELSE (IF c THEN x ELSE y ENDIF)
     ENDIF

 END if_props
```

```
functions[D, R: TYPE]: THEORY
 BEGIN

  f, g: VAR [D -> R]

  x, x1, x2: VAR D

  y: VAR R

  Drel: VAR PRED[[D]]

  Rrel: VAR PRED[[R]]

  extensionality_postulate: POSTULATE
    (∀ (x: D): f(x) = g(x)) IFF f = g

  extensionality: LEMMA (∀ (x: D): f(x) = g(x)) IMPLIES f = g

  congruence: POSTULATE f = g AND x1 = x2 IMPLIES f(x1) = g(x2)

  η: LEMMA (λ (x: D): f(x)) = f

  injective?(f): bool =
        (∀ x1, x2: (f(x1) = f(x2) ⇒ (x1 = x2)))

  surjective?(f): bool = (∀ y: (∃ x: f(x) = y))

  bijective?(f): bool = injective?(f) & surjective?(f)

  bij_is_inj: JUDGEMENT (bijective?) SUBTYPE_OF (injective?)

  bij_is_surj: JUDGEMENT (bijective?) SUBTYPE_OF (surjective?)

  domain(f): TYPE = D

  range(f): TYPE = R

  graph(f)(x, y): bool = (f(x) = y)

  preserves(f, Drel, Rrel): bool =
        ∀ x1, x2: Drel(x1, x2) IMPLIES Rrel(f(x1), f(x2))
```

preserves(Drel, Rrel)(f): bool = preserves(f, Drel, Rrel)

inverts(f, Drel, Rrel): bool =
  $\forall$ x_1, x_2: Drel(x_1, x_2) IMPLIES Rrel($f(x_2)$, $f(x_1)$)

inverts(Drel, Rrel)(f): bool = inverts(f, Drel, Rrel)

END functions

```
functions_alt[D, R: TYPE, Drel: PRED[[D]], Rrel: PRED[[R]]]: THEORY
 BEGIN

  f: VAR [D -> R]

  preserves: [[D -> R] -> bool] = preserves(Drel, Rrel)

  inverts: [[D -> R] -> bool] = inverts(Drel, Rrel)

 END functions_alt
```

transpose[T_1, T_2, T_3: TYPE]: THEORY
BEGIN

f: VAR $[T_1 \rightarrow [T_2 \rightarrow T_3]]$

x: VAR T_1

y: VAR T_2

transpose(f)(y)(x): T_3 = $f(x)(y)$

END transpose

```
restrict[T: TYPE, S: TYPE FROM T, R: TYPE]: THEORY
 BEGIN

  f: VAR [T -> R]

  s: VAR S

  restrict(f)(s): R = f(s)

  CONVERSION restrict


  injective_restrict: LEMMA
     injective?(f) IMPLIES injective?(restrict(f))

  restrict_of_inj_is_inj: JUDGEMENT restrict(f: (injective?[T, R])) HAS_TYPE
        (injective?[S, R])

 END restrict
```

restrict_props[T: TYPE, R: TYPE]: THEORY
BEGIN

f: VAR [$T \rightarrow R$]

restrict_full: LEMMA restrict[T, T, R](f) $=$ f

END restrict_props

extend[T: TYPE, S: TYPE FROM T, R: TYPE, d: R]: THEORY
BEGIN

f: VAR $[S \to R]$

t: VAR T

extend(f)(t): R = IF S_pred(t) THEN $f(t)$ ELSE d ENDIF

restrict_extend: LEMMA restrict[T, S, R](extend(f)) $=$ f

END extend

```
extend_bool[T: TYPE, S: TYPE FROM T]: THEORY
  BEGIN

    CONVERSION extend[T, S, bool, FALSE]


  END extend_bool
```

extend_props[T: TYPE, R: TYPE, d: R]: THEORY
BEGIN

f: VAR [$T \to R$]

extend_full: LEMMA extend[T, T, R, d](f) = f

END extend_props

extend_func_props[T: TYPE, S: TYPE FROM T, R: TYPE, d: R]: THEORY
BEGIN

surjective_extend: JUDGEMENT extend[T, S, R, d](f: (surjective?[S, R])) HAS_TYPE (surjective?[T, R])

END extend_func_props

K_conversion[T_1, T_2: TYPE]: THEORY
BEGIN

K_conversion(x: T_1)(y: T_2): T_1 = x

END K_conversion

```
K_props[T1, T2: TYPE, S: TYPE FROM T1]: THEORY
 BEGIN

  K_preserves: JUDGEMENT K_conversion[T1, T2](x: S)(y: T2) HAS_TYPE
        S

  K_preserves1: JUDGEMENT K_conversion[T1, T2](x: S) HAS_TYPE
        [T2 -> S]

 END K_props
```

identity[T: TYPE]: THEORY
BEGIN

x: VAR T

I: (bijective?[T, T]) = (λ x: x)

id: (bijective?[T, T]) = (λ x: x)

identity: (bijective?[T, T]) = (λ x: x)

END identity

```
identity_props[T: TYPE, S: TYPE FROM T]: THEORY
 BEGIN

  x: VAR S

  I_preserves: JUDGEMENT I[T](x) HAS_TYPE S

  id_preserves: JUDGEMENT id[T](x) HAS_TYPE S

  identity_preserves: JUDGEMENT identity[T](x) HAS_TYPE S

 END identity_props
```

```
relations[T: TYPE]: THEORY
 BEGIN

  R: VAR PRED[[T]]

  x, y, z: VAR T

  eq: pred[[T]] = (λ x, y: x = y)

  reflexive?(R): bool = ∀ x: R(x, x)

  irreflexive?(R): bool = ∀ x: NOT R(x, x)

  symmetric?(R): bool = ∀ x, y: R(x, y) IMPLIES R(y, x)

  antisymmetric?(R): bool =
        ∀ x, y: R(x, y) & R(y, x) ⇒ x = y

  connected?(R): bool =
        ∀ x, y: x ≠ y IMPLIES R(x, y) OR R(y, x)

  transitive?(R): bool =
        ∀ x, y, z: R(x, y) & R(y, z) ⇒ R(x, z)

  equivalence?(R): bool =
        reflexive?(R) AND symmetric?(R) AND transitive?(R)

  equivalence: TYPE = (equivalence?)

  equiv_is_reflexive: JUDGEMENT (equivalence?) SUBTYPE_OF (reflexive?)

  equiv_is_symmetric: JUDGEMENT (equivalence?) SUBTYPE_OF (symmetric?)

  equiv_is_transitive: JUDGEMENT (equivalence?) SUBTYPE_OF
        (transitive?)

  RC(R)(x, y): bool = (x = y) OR R(x, y)

  TC(R)(x, y):  INDUCTIVE bool =
          R(x, y) OR (∃ z: TC(R)(x, z) AND TC(R)(z, y))

  RTC(R)(x, y): bool = (x = y) OR TC(R)(x, y)
```

END relations

```
orders[T: TYPE]: THEORY
 BEGIN

  x, y: VAR T

  ≤, <: VAR pred[[T]]

  p: VAR pred[T]

  preorder?(≤): bool = reflexive?(≤) & transitive?(≤)

  preorder_is_reflexive: JUDGEMENT (preorder?) SUBTYPE_OF
        (reflexive?[T])

  preorder_is_transitive: JUDGEMENT (preorder?) SUBTYPE_OF
        (transitive?[T])

  equiv_is_preorder: JUDGEMENT (equivalence?[T]) SUBTYPE_OF
        (preorder?)

  partial_order?(≤): bool = preorder?(≤) & antisymmetric?(≤)

  po_is_preorder: JUDGEMENT (partial_order?) SUBTYPE_OF (preorder?)

  po_is_antisymmetric: JUDGEMENT (partial_order?) SUBTYPE_OF
        (antisymmetric?[T])

  strict_order?(<): bool = irreflexive?(<) & transitive?(<)

  strict_is_irreflexive: JUDGEMENT (strict_order?) SUBTYPE_OF
        (irreflexive?[T])

  strict_order_is_antisymmetric: JUDGEMENT (strict_order?) SUBTYPE_OF
        (antisymmetric?[T])

  strict_is_transitive: JUDGEMENT (strict_order?) SUBTYPE_OF
        (transitive?[T])

  dichotomous?(≤): bool = (∀ x, y: (x ≤ y OR y ≤ x))

  total_order?(≤): bool = partial_order?(≤) & dichotomous?(≤)
```

total_is_po: JUDGEMENT (total_order?) SUBTYPE_OF (partial_order?)

total_is_dichotomous: JUDGEMENT (total_order?) SUBTYPE_OF
(dichotomous?)

linear_order?(≤): bool = total_order?(≤)

linear_is_total: JUDGEMENT (linear_order?) SUBTYPE_OF (total_order?)

total_is_linear: JUDGEMENT (total_order?) SUBTYPE_OF (linear_order?)

trichotomous?(<): bool =
(∀ x, y: $x < y$ OR $y < x$ OR $x = y$)

strict_total_order?(<): bool =
strict_order?(<) & trichotomous?(<)

strict_total_is_strict: JUDGEMENT (strict_total_order?) SUBTYPE_OF
(strict_order?)

strict_total_is_trichotomous: JUDGEMENT (strict_total_order?) SUBTYPE_OF
(trichotomous?)

well_founded?(<): bool =
(∀ p:
(∃ y: $p(y)$) IMPLIES
(∃ (y: (p)): (∀ (x: (p)): (NOT $x < y$))))

strict_well_founded?(<): bool =
strict_order?(<) & well_founded?(<)

strict_well_founded_is_strict: JUDGEMENT (strict_well_founded?) SUBTYPE_OF
(strict_order?)

strict_well_founded_is_well_founded: JUDGEMENT (strict_well_founded?) SUBTYPE_OF
(well_founded?)

well_ordered?(<): bool =
strict_total_order?(<) & well_founded?(<)

well_ordered_is_strict_total: JUDGEMENT (well_ordered?) SUBTYPE_OF
(strict_total_order?)

```
well_ordered_is_well_founded: JUDGEMENT (well_ordered?) SUBTYPE_OF
    (well_founded?)

nonempty_pred: TYPE = {p: pred[T] | ∃ (x: T): p(x)}

pe: VAR pred[T]

upper_bound?(<)(x, pe): bool = ∀ (y: (pe)): y < x

upper_bound?(<)(pe)(x): bool = upper_bound?(<)(x, pe)

lower_bound?(<)(x, pe): bool = ∀ (y: (pe)): x < y

lower_bound?(<)(pe)(x): bool = lower_bound?(<)(x, pe)

least_upper_bound?(<)(x, pe): bool =
    upper_bound?(<)(x, pe) AND
     ∀ y: upper_bound?(<)(y, pe) IMPLIES (x < y OR x = y)

least_upper_bound?(<)(pe)(x): bool =
    least_upper_bound?(<)(x, pe)

greatest_lower_bound?(<)(x, pe): bool =
    lower_bound?(<)(x, pe) AND
     ∀ y: lower_bound?(<)(y, pe) IMPLIES (y < x OR x = y)

greatest_lower_bound?(<)(pe)(x): bool =
    greatest_lower_bound?(<)(x, pe)

END orders
```

orders_alt[T: TYPE, $<$: pred[[T]], pe: nonempty_pred[T]]: THEORY
BEGIN

x: VAR T

upper_bound?: [$T \rightarrow$ bool] = upper_bound?($<$)(pe)

least_upper_bound?: [$T \rightarrow$ bool] = least_upper_bound?($<$)(pe)

lower_bound?: [$T \rightarrow$ bool] = lower_bound?($<$)(pe)

greatest_lower_bound?: [$T \rightarrow$ bool] =
greatest_lower_bound?($<$)(pe)

least_upper_bound_is_upper_bound: JUDGEMENT (least_upper_bound?) SUBTYPE_OF
(upper_bound?)

greatest_lower_bound_is_lower_bound: JUDGEMENT (greatest_lower_bound?) SUBTYPE_OF
(lower_bound?)

END orders_alt

restrict_order_props[T: TYPE, S: TYPE FROM T]: THEORY
BEGIN

reflexive_restrict: JUDGEMENT restrict(R: (reflexive?[T])) HAS_TYPE
(reflexive?[S])

irreflexive_restrict: JUDGEMENT restrict(R: (irreflexive?[T])) HAS_TYPE
(irreflexive?[S])

symmetric_restrict: JUDGEMENT restrict(R: (symmetric?[T])) HAS_TYPE
(symmetric?[S])

antisymmetric_restrict: JUDGEMENT restrict(R: (antisymmetric?[T])) HAS_TYPE
(antisymmetric?[S])

connected_restrict: JUDGEMENT restrict(R: (connected?[T])) HAS_TYPE
(connected?[S])

transitive_restrict: JUDGEMENT restrict(R: (transitive?[T])) HAS_TYPE
(transitive?[S])

equivalence_restrict: JUDGEMENT restrict(R: (equivalence?[T])) HAS_TYPE
(equivalence?[S])

preorder_restrict: JUDGEMENT restrict(R: (preorder?[T])) HAS_TYPE
(preorder?[S])

partial_order_restrict: JUDGEMENT restrict(R: (partial_order?[T])) HAS_TYPE
(partial_order?[S])

strict_order_restrict: JUDGEMENT restrict(R: (strict_order?[T])) HAS_TYPE
(strict_order?[S])

dichotomous_restrict: JUDGEMENT restrict(R: (dichotomous?[T])) HAS_TYPE
(dichotomous?[S])

total_order_restrict: JUDGEMENT restrict(R: (total_order?[T])) HAS_TYPE
(total_order?[S])

trichotomous_restrict: JUDGEMENT restrict(R: (trichotomous?[T])) HAS_TYPE
(trichotomous?[S])

strict_total_order_restrict: JUDGEMENT restrict(R: (strict_total_order?[T])) HAS_TYPE
(strict_total_order?[S])

well_founded_restrict: JUDGEMENT restrict(R: (well_founded?[T])) HAS_TYPE
(well_founded?[S])

well_ordered_restrict: JUDGEMENT restrict(R: (well_ordered?[T])) HAS_TYPE
(well_ordered?[S])

END restrict_order_props

extend_order_props $[T$: TYPE, S: TYPE FROM $T]$: THEORY
BEGIN

irreflexive_extend: JUDGEMENT extend$[[T], [S]$, bool, FALSE$]$(R: (irreflexive?$[S]$))
HAS_TYPE (irreflexive?$[T]$)

symmetric_extend: JUDGEMENT extend$[[T], [S]$, bool, FALSE$]$(R: (symmetric?$[S]$))
HAS_TYPE (symmetric?$[T]$)

antisymmetric_extend: JUDGEMENT extend
$[[T], [S]$, bool,
FALSE$]$(R: (antisymmetric?$[S]$))
HAS_TYPE (antisymmetric?$[T]$)

transitive_extend: JUDGEMENT extend$[[T], [S]$, bool, FALSE$]$(R: (transitive?$[S]$))
HAS_TYPE (transitive?$[T]$)

strict_order_extend: JUDGEMENT extend
$[[T], [S]$, bool, FALSE$]$(R: (strict_order?$[S]$))
HAS_TYPE (strict_order?$[T]$)

END extend_order_props

```
wf_induction[T: TYPE, <: (well_founded?[T])]: THEORY
 BEGIN

  wf_induction: LEMMA
    (∀ (p: pred[T]):
        (∀ (x: T): (∀ (y: T): y < x IMPLIES p(y)) IMPLIES p(x)) IMPLIES
         (∀ (x: T): p(x)))

 END wf_induction
```

measure_induction$[T$: TYPE, M: TYPE, m: $[T \to M]$, $<$: (well_founded?$[M]$)$]$: THEORY
BEGIN

measure_induction: LEMMA
$(\forall$ $(p$: pred$[T]$):
$(\forall$ $(x$: $T)$: $(\forall$ $(y$: $T)$: $m(y) < m(x)$ IMPLIES $p(y))$ IMPLIES $p(x))$ IMPLIES
$(\forall$ $(x$: $T)$: $p(x)))$

END measure_induction

epsilons[T: TYPE+]: THEORY
BEGIN

p: VAR pred[T]

x: VAR T

$\varepsilon(p)$: T

epsilon_ax: AXIOM $(\exists\ x:\ p(x)) \Rightarrow p(\varepsilon(p))$

END epsilons

sets[T: TYPE]: THEORY
BEGIN

set: TYPE = setof[T]

x, y: VAR T

a, b, c: VAR set

p: VAR PRED[T]

$(x \in a)$: bool = $a(x)$

empty?(a): bool = ($\forall$ x: NOT $(x \in a)$)

$\emptyset$: set = $\{x$ | FALSE$\}$

nonempty?(a): bool = NOT empty?(a)

nonempty_set: TYPE = (nonempty?)

full?(a): bool = ($\forall$ x: $(x \in a)$)

fullset: set = $\{x$ | TRUE$\}$

nontrivial?(a): bool = $a \neq \emptyset$ & $a \neq$ fullset

$(a \subseteq b)$: bool = ($\forall$ x: $(x \in a) \Rightarrow (x \in b)$)

$(a \subset b)$: bool = $(a \subseteq b)$ & $a \neq b$

$(a \cup b)$: set = $\{x \mid (x \in a)$ OR $(x \in b)\}$

$(a \cap b)$: set = $\{x \mid (x \in a)$ AND $(x \in b)\}$

disjoint?(a, b): bool = empty?($(a \cap b)$)

$\overline{a}$: set = $\{x$ | NOT $(x \in a)\}$

$(a \setminus b)$: set = $\{x \mid (x \in a)$ AND NOT $(x \in b)\}$

symmetric_difference(a, b): set = $((a \setminus b) \cup (b \setminus a))$

every(p)(a): bool = ∀ (x: (a)): $p(x)$

every(p, a): bool = ∀ (x: (a)): $p(x)$

some(p)(a): bool = ∃ (x: (a)): $p(x)$

some(p, a): bool = ∃ (x: (a)): $p(x)$

singleton?(a): bool =
  (∃ (x: (a)): (∀ (y: (a)): $x = y$))

singleton(x): (singleton?) = $\{y \mid y = x\}$

$(a \cup \{x\})$: (nonempty?) = $\{y \mid x = y$ OR $(y \in a)\}$

$(a \setminus \{x\})$: set = $\{y \mid x \neq y$ AND $(y \in a)\}$

choose(p: (nonempty?)): (p)

choose_is_epsilon: AXIOM
  ∀ (p: (nonempty?)): choose(p) $=$ $\varepsilon(p)$

the(p: (singleton?)): (p)

the_lem: LEMMA ∀ (p: (singleton?)): the(p) $=$ $\varepsilon(p)$

the_prop: LEMMA ∀ (p: (singleton?)): p(the(p))

singleton_elt(a: (singleton?)): T = the! x: $(x \in a)$

CONVERSION+ singleton_elt

is_singleton: LEMMA
  ∀ a:
    (nonempty?(a) AND ∀ x, y: $a(x)$ AND $a(y)$ IMPLIES $(x = y)$) IMPLIES
      singleton?(a)

singleton_elt_lem: LEMMA
  singleton?(a) AND $a(x)$ IMPLIES singleton_elt(a) $= x$

singleton_elt_def : LEMMA
  singleton?(a) IMPLIES singleton_elt(a) $=$ choose(a)

singleton_singleton : LEMMA
  singleton?(a) IMPLIES ($\exists$ x: a $=$ singleton(x))

singleton_rew : LEMMA singleton_elt(singleton(x)) $=$ x

AUTO_REWRITE+ singleton_rew

rest(a): set =
    IF empty?(a) THEN a ELSE $(a \setminus \{$choose$(a)\})$ ENDIF

setofsets : TYPE = setof[setof[T]]

A, B: VAR setofsets

powerset(a): setofsets = $\{b \mid (b \subseteq a)\}$

$\bigcup A$: set = $\{x \mid \exists\ (a: (A)): a(x)\}$

$\bigcap A$: set = $\{x \mid \forall\ (a: (A)): a(x)\}$

nonempty_singleton : JUDGEMENT (singleton?) SUBTYPE_OF (nonempty?)

nonempty_union1 : JUDGEMENT union(a: (nonempty?), b: set) HAS_TYPE
    (nonempty?)

nonempty_union2 : JUDGEMENT union(a: set, b: (nonempty?)) HAS_TYPE
    (nonempty?)

END sets

```
sets_lemmas[T: TYPE]: THEORY
 BEGIN

  x, y: VAR T

  a, b, c: VAR set[T]

  A, B: VAR setofsets[T]

  extensionality: LEMMA
    (∀ x: (x ∈ a) IFF (x ∈ b)) IMPLIES (a = b)

  emptyset_is_empty?: LEMMA empty?(a) IFF a = ∅

  empty_no_members: LEMMA NOT (x ∈ ∅)

  emptyset_min: LEMMA (a ⊆ ∅) IMPLIES a = ∅

  nonempty_member: LEMMA nonempty?(a) IFF ∃ x: (x ∈ a)

  fullset_member: LEMMA (x ∈ fullset)

  fullset_max: LEMMA (fullset ⊆ a) IMPLIES a = fullset

  fullset_is_full?: LEMMA full?(a) IFF a = fullset

  nonempty_exists: LEMMA nonempty?(a) IFF ∃ (x: (a)): TRUE

  subset_emptyset: LEMMA (∅ ⊆ a)

  subset_fullset: LEMMA (a ⊆ fullset)

  subset_reflexive: LEMMA (a ⊆ a)

  subset_antisymmetric: LEMMA (a ⊆ b) AND (b ⊆ a) IMPLIES a = b

  subset_transitive: LEMMA
    (a ⊆ b) AND (b ⊆ c) IMPLIES (a ⊆ c)

  subset_partial_order: LEMMA partial_order?(subset?[T])

  subset_is_partial_order: JUDGEMENT subset?[T] HAS_TYPE
```

(partial_order?[set[T]])

strict_subset_irreflexive : LEMMA NOT $(a \subset a)$

strict_subset_transitive : LEMMA
$(a \subset b)$ AND $(b \subset c)$ IMPLIES $(a \subset c)$

strict_subset_strict_order : LEMMA strict_order?(strict_subset?[T])

strict_subset_is_strict_order : JUDGEMENT strict_subset?[T] HAS_TYPE
(strict_order?[set[T]])

union_idempotent : LEMMA $(a \cup a) = a$

union_commutative : LEMMA $(a \cup b) = (b \cup a)$

union_associative : LEMMA $((a \cup b) \cup c) = (a \cup (b \cup c))$

union_empty : LEMMA $(a \cup \emptyset) = a$

union_full : LEMMA $(a \cup$ fullset$) =$ fullset

union_subset1 : LEMMA $(a \subseteq (a \cup b))$

union_subset2 : LEMMA $(a \subseteq b)$ IMPLIES $(a \cup b) = b$

subset_union : LEMMA
$((a \cup b) \subseteq c) = ((a \subseteq c)$ AND $(b \subseteq c))$

union_upper_bound : LEMMA
$(a \subseteq c)$ AND $(b \subseteq c)$ IMPLIES $((a \cup b) \subseteq c)$

union_difference : LEMMA $(a \cup b) = (a \cup (b \setminus a))$

union_diff_subset : LEMMA $(a \subseteq b)$ IMPLIES $(a \cup (b \setminus a)) = b$

intersection_idempotent : LEMMA $(a \cap a) = a$

intersection_commutative : LEMMA $(a \cap b) = (b \cap a)$

intersection_associative : LEMMA
$((a \cap b) \cap c) = (a \cap (b \cap c))$

intersection_empty : LEMMA $(a \cap \emptyset) = \emptyset$

intersection_full : LEMMA $(a \cap \text{fullset}) = a$

intersection_subset1 : LEMMA $((a \cap b) \subseteq a)$

intersection_subset2 : LEMMA $(a \subseteq b)$ IMPLIES $(a \cap b) = a$

intersection_lower_bound : LEMMA
$(c \subseteq a)$ AND $(c \subseteq b)$ IMPLIES $(c \subseteq (a \cap b))$

distribute_intersection_union : LEMMA
$(a \cap (b \cup c)) = ((a \cap b) \cup (a \cap c))$

distribute_union_intersection : LEMMA
$(a \cup (b \cap c)) = ((a \cup b) \cap (a \cup c))$

complement_emptyset : LEMMA $\overline{\emptyset[T]} = \text{fullset}$

complement_fullset : LEMMA $\overline{\text{fullset}[T]} = \emptyset$

complement_complement : LEMMA $\overline{\overline{a}} = a$

complement_equal : LEMMA $\overline{a} = \overline{b}$ IFF $a = b$

subset_complement : LEMMA $(\overline{a} \subseteq \overline{b})$ IFF $(b \subseteq a)$

demorgan1 : LEMMA $\overline{(a \cup b)} = (\overline{a} \cap \overline{b})$

demorgan2 : LEMMA $\overline{(a \cap b)} = (\overline{a} \cup \overline{b})$

difference_emptyset1 : LEMMA $(a \setminus \emptyset) = a$

difference_emptyset2 : LEMMA $(\emptyset \setminus a) = \emptyset$

difference_fullset1 : LEMMA $(a \setminus \text{fullset}) = \emptyset$

difference_fullset2 : LEMMA $(\text{fullset} \setminus a) = \overline{a}$

difference_intersection : LEMMA $(a \setminus b) = (a \cap \overline{b})$

difference_difference1 : LEMMA
$((a \setminus b) \setminus c) = (a \setminus (b \cup c))$

difference_difference2 : LEMMA
$(a \setminus (b \setminus c)) = ((a \setminus b) \cup (a \cap c))$

difference_subset : LEMMA $((a \setminus b) \subseteq a)$

difference_subset2 : LEMMA $(a \subseteq b)$ IMPLIES $(a \setminus b) = \emptyset$

difference_disjoint : LEMMA disjoint?(a, $(b \setminus a)$)

difference_disjoint2 : LEMMA disjoint?(a, b) IMPLIES $(a \setminus b) = a$

diff_union_inter : LEMMA
$((a \cup b) \setminus a) = (b \setminus (a \cap b))$

nonempty_add : LEMMA NOT empty?($(a \cup \{x\})$)

member_add : LEMMA $(x \in a)$ IMPLIES $(a \cup \{x\}) = a$

member_add_reduce : LEMMA
$(x \in (a \cup \{y\})) =$ ($x = y$ OR $(x \in a)$)

member_remove : LEMMA NOT $(x \in a)$ IMPLIES $(a \setminus \{x\}) = a$

add_remove_member : LEMMA
$(x \in a)$ IMPLIES $((a \setminus \{x\}) \cup \{x\}) = a$

remove_add_member : LEMMA
NOT $(x \in a)$ IMPLIES $((a \cup \{x\}) \setminus \{x\}) = a$

subset_add : LEMMA $(a \subseteq (a \cup \{x\}))$

add_as_union : LEMMA $(a \cup \{x\}) =$ ($a \cup$ singleton(x))

singleton_as_add : LEMMA singleton(x) $= (\emptyset \cup \{x\})$

subset_remove : LEMMA $((a \setminus \{x\}) \subseteq a)$

remove_as_difference : LEMMA
$(a \setminus \{x\}) =$ ($a \setminus$ singleton(x))

remove_member_singleton : LEMMA $(\text{singleton}(x) \setminus \{x\}) = \emptyset$

choose_rest : LEMMA
NOT empty?(a) IMPLIES $(\text{rest}(a) \cup \{\text{choose}(a)\}) = a$

choose_member : LEMMA NOT empty?(a) IMPLIES $(\text{choose}(a) \in a)$

choose_not_member : LEMMA
NOT empty?(a) IMPLIES NOT $(\text{choose}(a) \in \text{rest}(a))$

rest_not_equal : LEMMA NOT empty?(a) IMPLIES $\text{rest}(a) \neq a$

rest_member : LEMMA $(x \in \text{rest}(a))$ IMPLIES $(x \in a)$

rest_subset : LEMMA $(\text{rest}(a) \subseteq a)$

choose_add : LEMMA
$\text{choose}((a \cup \{x\})) = x$ OR
$(\text{choose}((a \cup \{x\})) \in a)$

choose_rest_or : LEMMA
$(x \in a)$ IMPLIES $(x \in \text{rest}(a))$ OR $x = \text{choose}(a)$

choose_singleton : LEMMA $\text{choose}(\text{singleton}(x)) = x$

rest_singleton : LEMMA $\text{rest}(\text{singleton}(x)) = \emptyset[T]$

singleton_subset : LEMMA $(x \in a)$ IFF $(\text{singleton}(x) \subseteq a)$

rest_empty_lem : LEMMA
NOT empty?(a) AND empty?(rest(a)) IMPLIES
$a = \text{extend}[T, (a), \text{bool}, \text{FALSE}](\text{singleton}(\text{choose}(a)))$

singleton_disjoint : LEMMA
NOT $(x \in a)$ IMPLIES disjoint?(singleton(x), a)

disjoint_remove_left : LEMMA
disjoint?(a, b) IMPLIES disjoint?(($a \setminus \{x\}$), b)

disjoint_remove_right : LEMMA
disjoint?(a, b) IMPLIES disjoint?(a, ($b \setminus \{x\}$))

union_disj_remove_left : LEMMA
disjoint?(a, b) AND $a(x)$ IMPLIES
$((a \setminus \{x\}) \cup b) = ((a \cup b) \setminus \{x\})$

union_disj_remove_right : LEMMA
disjoint?(a, b) AND $b(x)$ IMPLIES
$(a \cup (b \setminus \{x\})) = ((a \cup b) \setminus \{x\})$

subset_powerset : LEMMA $(a \subseteq b)$ IFF $(a \in \text{powerset}(b))$

empty_powerset : LEMMA empty?(a) IFF singleton?(powerset(a))

powerset_emptyset : LEMMA $(\emptyset \in \text{powerset}(a))$

nonempty_powerset : JUDGEMENT powerset(a) HAS_TYPE
(nonempty?[set[T]])

powerset_union : LEMMA $\bigcup \text{powerset}(a) = a$

powerset_intersection : LEMMA empty?($\bigcap$ powerset(a))

powerset_subset : LEMMA
$(a \subseteq b)$ IFF $(\text{powerset}(a) \subseteq \text{powerset}(b))$

Union_empty : LEMMA
empty?($\bigcup A$) IFF empty?(A) OR every(empty?)(A)

Union_full : LEMMA
full?($\bigcup A$) IFF $(\forall\ x :\ \exists\ (a :\ (A)) :\ (x \in a))$

Union_member : LEMMA
$(x \in \bigcup A)$ IFF $(\exists\ (a :\ (A)) :\ (x \in a))$

Union_subset : LEMMA $\forall\ (a :\ (A)) :\ (a \subseteq \bigcup A)$

Union_surjective : JUDGEMENT Union HAS_TYPE
(surjective?[setofsets[T], set[T]])

Union_emptyset_rew : LEMMA $\bigcup \emptyset[\text{set}[T]] = \emptyset$

Union_union_rew : LEMMA

nonempty?(A) IMPLIES
$\bigcup A$ = (choose(A) $\cup \bigcup$ rest(A))

Intersection_empty : LEMMA
empty?($\bigcap A$) IFF
($\forall$ x : $\exists$ (a : (A)) : NOT ($x \in a$))

Intersection_full : LEMMA full?($\bigcap A$) IFF every(full?)(A)

Intersection_member : LEMMA
($x \in \bigcap A$) IFF ($\forall$ (a : (A)) : ($x \in a$))

Intersection_empty_full : COROLLARY full?($\bigcap \emptyset$[set[T]])

Intersection_surjective : JUDGEMENT Intersection HAS_TYPE
(surjective?[setofsets[T], set[T]])

Intersection_intersection_rew : LEMMA
nonempty?(A) IMPLIES
$\bigcap A$ = (choose(A) $\cap \bigcap$ rest(A))

Complement(A) : setofsets[T] =
{a | $\exists$ (b : (A)) : $a = \overline{b}$}

Complement_empty : LEMMA empty?(Complement(A)) IFF empty?(A)

Complement_full : LEMMA full?(Complement(A)) IFF full?(A)

Complement_Complement : LEMMA Complement(Complement(A)) = A

subset_Complement : LEMMA
(Complement(A) $\subseteq$ Complement(B)) IFF ($A \subseteq B$)

Complement_bijective : JUDGEMENT Complement HAS_TYPE
(bijective?[setofsets[T], setofsets[T]])

Demorgan1 : LEMMA $\overline{\bigcup A}$ = $\bigcap$Complement(A)

Demorgan2 : LEMMA $\overline{\bigcap A}$ = $\bigcup$Complement(A)

END sets_lemmas

```
function_inverse_def[D: TYPE, R: TYPE]: THEORY
 BEGIN

  d: VAR D

  r: VAR R

  f: VAR [D -> R]

  g: VAR [R -> D]

  left_inverse?(g, f): bool = ∀ d: g(f(d)) = d

  right_inverse?(g, f): bool = ∀ r: f(g(r)) = r

  inverse?(g, f): bool =
        ∀ r: (∃ d: f(d) = r) ⇒ f(g(r)) = r

  left_inverse?(f)(g): MACRO bool = left_inverse?(g, f)

  right_inverse?(f)(g): MACRO bool = right_inverse?(g, f)

  inverse?(f)(g): MACRO bool = inverse?(g, f)

  left_inverse_is_inverse: LEMMA
    ∀ f, (g: (λ (g): left_inverse?(g, f))):
      inverse?(g, f)

  left_inj_surj: LEMMA
    ∀ f, (g: (λ (g): left_inverse?(g, f))):
      injective?(f) AND surjective?(g)

  inj_left_alt: LEMMA
    ∀ (f: (injective?[D, R])), (g: (λ (g): inverse?(g, f))):
      left_inverse?(g, f)

  surj_inv_alt: COROLLARY
    ∀ (f: (injective?[D, R])), (g: (λ (g): inverse?(g, f))):
      surjective?(g)

  injective_inverse_alt: LEMMA
    ∀ (f: (injective?[D, R])), (g: (λ (g): inverse?(g, f))):
```

$r = f(d) \Rightarrow g(r) = d$

comp_inverse_left_inj_alt: LEMMA
$\forall$ (f: (injective?[D, R])), (g: (λ (g): inverse?(g, f))):
$g(f(d)) = d$

noninjective_inverse_exists: LEMMA
$\forall$ f: injective?(f) OR ($\exists$ g: inverse?(g, f))

right_inverse_is_inverse: LEMMA
$\forall$ f, (g: (λ (g): right_inverse?(g, f))):
inverse?(g, f)

right_surj_inj: LEMMA
$\forall$ f, (g: (λ (g): right_inverse?(g, f))):
surjective?(f) AND injective?(g)

surj_right_alt: LEMMA
$\forall$ (f: (surjective?[D, R])), (g: (λ (g): inverse?(g, f))):
right_inverse?(g, f)

inj_inv_alt: COROLLARY
$\forall$ (f: (surjective?[D, R])), (g: (λ (g): inverse?(g, f))):
injective?(g)

surjective_inverse_alt: LEMMA
$\forall$ (f: (surjective?[D, R])), (g: (λ (g): inverse?(g, f))):
$g(r) = d \Rightarrow r = f(d)$

comp_inverse_right_surj_alt: LEMMA
$\forall$ (f: (surjective?[D, R])), (g: (λ (g): inverse?(g, f))):
$f(g(r)) = r$

surjective_inverse_exists: LEMMA
$\forall$ (f: (surjective?[D, R])): $\exists$ g: inverse?(g, f)

left_right_bij: COROLLARY
$\forall$ f, g:
right_inverse?(g, f) AND left_inverse?(g, f) $\Rightarrow$
bijective?(f) AND bijective?(g)

bij_left_right: COROLLARY

$\forall$ (f: (bijective?[D, R])), (g: (λ (g): inverse?(g, f))):
right_inverse?(g, f) AND left_inverse?(g, f)

bij_inv_is_bij_alt: COROLLARY
$\forall$ (f: (bijective?[D, R])), (g: (λ (g): inverse?(g, f))):
bijective?(g)

bijective_inverse_alt: COROLLARY
$\forall$ (f: (bijective?[D, R])), (g: (λ (g): inverse?(g, f))):
$g(r) = d$ IFF $r = f(d)$

comp_inverse_right_alt: COROLLARY
$\forall$ (f: (bijective?[D, R])), (g: (λ (g): inverse?(g, f))):
$f(g(r)) = r$

comp_inverse_left_alt: COROLLARY
$\forall$ (f: (bijective?[D, R])), (g: (λ (g): inverse?(g, f))):
$g(f(d)) = d$

bijective_inverse_exists: LEMMA
$\forall$ (f: (bijective?[D, R])):
exists1(λ (g): inverse?(g, f))

exists_inv1: LEMMA
($\exists$ g: TRUE) IFF
(($\exists$ (d: D): TRUE) OR ($\forall$ (r: R): FALSE))

exists_inv2: LEMMA
($\exists$ (f: (surjective?[D, R])): TRUE) $\Rightarrow$
(($\exists$ (d: D): TRUE) OR ($\forall$ (r: R): FALSE))

exists_inv3: LEMMA
($\exists$ f: NOT injective?(f)) $\Rightarrow$
(($\exists$ (d: D): TRUE) OR ($\forall$ (r: R): FALSE))

END function_inverse_def

function_inverse$[D$: TYPE+, R: TYPE$]$: THEORY
BEGIN

x: VAR D

y: VAR R

f: VAR $[D \to R]$

g: VAR $[R \to D]$

inverse$(f)(y)$: D = $(\varepsilon!\ x: f(x) = y)$

unique_bijective_inverse: JUDGEMENT inverse(f: (bijective?$[D, R]$))(y) HAS_TYPE
$\{x: D \mid f(x) = y\}$

bijective_inverse_is_bijective: JUDGEMENT inverse(f: (bijective?$[D, R]$)) HAS_TYPE
(bijective?$[R, D]$)

surjective_inverse: LEMMA
$\forall$ (f: (surjective?$[D, R]$)):
inverse$(f)(y) = x$ IMPLIES $y = f(x)$

inverse_surjective: LEMMA
$\forall$ (f: (surjective?$[D, R]$)): f(inverse$(f)(y)) = y$

injective_inverse: LEMMA
$\forall$ (f: (injective?$[D, R]$)):
$y = f(x)$ IMPLIES inverse$(f)(y) = x$

inverse_injective: LEMMA
$\forall$ (f: (injective?$[D, R]$)): inverse$(f)(f(x)) = x$

bijective_inverse: LEMMA
$\forall$ (f: (bijective?$[D, R]$)):
inverse$(f)(y) = x$ IFF $y = f(x)$

bij_inv_is_bij: LEMMA bijective?(f) IMPLIES bijective?(inverse(f))

surj_right: LEMMA
surjective?(f) IFF right_inverse?(inverse(f), f)

inj_left: LEMMA injective?(f) IFF left_inverse?(inverse(f), f)

inj_inv: LEMMA surjective?(f) IMPLIES injective?(inverse(f))

surj_inv: LEMMA injective?(f) IMPLIES surjective?(inverse(f))

inv_inj_is_surj: JUDGEMENT inverse(f: (injective?$[D, R]$)) HAS_TYPE
(surjective?$[R, D]$)

inv_surj_is_inj: JUDGEMENT inverse(f: (surjective?$[D, R]$)) HAS_TYPE
(injective?$[R, D]$)

comp_inverse_right_surj: LEMMA
$\forall$ (f: (surjective?$[D, R]$)): f(inverse(f)(y)) $= y$

comp_inverse_left_inj: LEMMA
$\forall$ (f: (injective?$[D, R]$)): inverse(f)($f(x)$) $= x$

comp_inverse_right: LEMMA
$\forall$ (f: (bijective?$[D, R]$)): f(inverse(f)(y)) $= y$

comp_inverse_left: LEMMA
$\forall$ (f: (bijective?$[D, R]$)): inverse(f)($f(x)$) $= x$

END function_inverse

```
function_inverse_alt[D: TYPE, R: TYPE]: THEORY
 BEGIN

  ASSUMING
   inverse_types: ASSUMPTION
     (∃ (d: D): TRUE) OR (∀ (r: R): FALSE)
  ENDASSUMING

  d: VAR D

  r: VAR R

  f: VAR [D → R]

  g: VAR [R → D]

  inverses(f): TYPE+ = (λ (g: [R → D]): inverse?(g, f))

  inverse_alt(f): inverses(f) =
      choose({g: inverses(f) | TRUE})

  bijective_inverse_is_inverse_alt: COROLLARY
    ∀ (f: (bijective?[D, R])), (g: inverses(f)):
      g = inverse_alt(f)

  unique_bijective_inverse_alt: JUDGEMENT inverse_alt(f: (bijective?[D, R]))(r)
      HAS_TYPE {d | f(d) = r}

  bijective_inverse_alt_is_bijective: JUDGEMENT inverse_alt(f: (bijective?[D, R]))
      HAS_TYPE (bijective?[R, D])

  inv_inj_is_surj_alt: JUDGEMENT inverse_alt(f: (injective?[D, R])) HAS_TYPE
      (surjective?[R, D])

  inv_surj_is_inj_alt: JUDGEMENT inverse_alt(f: (surjective?[D, R])) HAS_TYPE
      (injective?[R, D])

 END function_inverse_alt
```

function_image[D, R: TYPE]: THEORY
BEGIN

f: VAR [$D \rightarrow R$]

x: VAR D

y: VAR R

X, X_1, X_2: VAR set[D]

Y, Y_1, Y_2: VAR set[R]

fun_exists: LEMMA
($\exists$ y: TRUE) OR (NOT $\exists$ x: TRUE) IMPLIES ($\exists$ f: TRUE)

image(f, X): set[R] =
{y: R | ($\exists$ (x: (X)): $y = f(x)$)}

image(f)(X): set[R] = image(f, X)

inverse_image(f, Y): set[D] = {x: D | ($f(x) \in Y$)}

inverse_image(f)(Y): set[D] = inverse_image(f, Y)

image_inverse_image: LEMMA
(image(f, inverse_image(f, Y)) $\subseteq Y$)

inverse_image_image: LEMMA
($X \subseteq$ inverse_image(f, image(f, X)))

image_subset: LEMMA
($X_1 \subseteq X_2$) IMPLIES (image(f, X_1) $\subseteq$ image(f, X_2))

inverse_image_subset: LEMMA
($Y_1 \subseteq Y_2$) IMPLIES
(inverse_image(f, Y_1) $\subseteq$ inverse_image(f, Y_2))

image_union: LEMMA
image(f, ($X_1 \cup X_2$)) $=$ (image(f, X_1) $\cup$ image(f, X_2))

image_intersection: LEMMA

$(\text{image}(f,\ (X_1 \cap X_2)) \subseteq (\text{image}(f,\ X_1) \cap \text{image}(f,\ X_2)))$

inverse_image_union: LEMMA
  inverse_image(f, $(Y_1 \cup Y_2)$) =
    (inverse_image(f, Y_1) $\cup$ inverse_image(f, Y_2))

inverse_image_intersection: LEMMA
  inverse_image(f, $(Y_1 \cap Y_2)$) =
    (inverse_image(f, Y_1) $\cap$ inverse_image(f, Y_2))

inverse_image_complement: LEMMA
  inverse_image(f, $\overline{Y}$) = $\overline{\text{inverse\_image}(f,\ Y)}$

END function_image

function_props$[T_1,\ T_2,\ T_3:$ TYPE$]$: THEORY
BEGIN

x: VAR T_1

f_1: VAR $[T_1 \to T_2]$

f_2: VAR $[T_2 \to T_3]$

X: VAR set$[T_1]$

R_1: VAR PRED$[[T_1]]$

R_2: VAR PRED$[[T_2]]$

R_3: VAR PRED$[[T_3]]$

$f_2 \circ f_1(x)$: T_3 = $f_2(f_1(x))$

composition_injective: JUDGEMENT $O(f_2$: (injective?$[T_2,\ T_3]$), f_1: (injective?$[T_1,\ T_2]$))
HAS_TYPE (injective?$[T_1,\ T_3]$)

composition_surjective: JUDGEMENT $O(f_2$: (surjective?$[T_2,\ T_3]$),
f_1: (surjective?$[T_1,\ T_2]$))
HAS_TYPE (surjective?$[T_1,\ T_3]$)

composition_bijective: JUDGEMENT $O(f_2$: (bijective?$[T_2,\ T_3]$), f_1: (bijective?$[T_1,\ T_2]$))
HAS_TYPE (bijective?$[T_1,\ T_3]$)

image_composition: LEMMA
image(f_2, image(f_1, X)) $=$ image($f_2 \circ f_1$, X)

preserves_composition: LEMMA
preserves(f_1, R_1, R_2) AND preserves(f_2, R_2, R_3) IMPLIES
preserves($f_2 \circ f_1$, R_1, R_3)

inverts_composition1: LEMMA
preserves(f_1, R_1, R_2) AND inverts(f_2, R_2, R_3) IMPLIES
inverts($f_2 \circ f_1$, R_1, R_3)

inverts_composition2: LEMMA
inverts(f_1, R_1, R_2) AND preserves(f_2, R_2, R_3) IMPLIES

inverts($f_2 \circ f_1$, R_1, R_3)

END function_props

function_props_alt[T_1, T_2, T_3: TYPE, R_1: PRED[[T_1]], R_2: PRED[[T_2]],
R_3: PRED[[T_3]]]: THEORY

BEGIN

composition_preserves: JUDGEMENT $O(f_2$: (preserves[T_2, T_3, R_2, R_3]),
f_1: (preserves[T_1, T_2, R_1, R_2]))
HAS_TYPE (preserves[T_1, T_3, R_1, R_3])

composition_inverts1: JUDGEMENT $O(f_2$: (preserves[T_2, T_3, R_2, R_3]),
f_1: (inverts[T_1, T_2, R_1, R_2]))
HAS_TYPE (inverts[T_1, T_3, R_1, R_3])

composition_inverts2: JUDGEMENT $O(f_2$: (inverts[T_2, T_3, R_2, R_3]),
f_1: (preserves[T_1, T_2, R_1, R_2]))
HAS_TYPE (inverts[T_1, T_3, R_1, R_3])

END function_props_alt

function_props2$[T_1,\ T_2,\ T_3,\ T_4:$ TYPE$]$: THEORY

BEGIN

f_1: VAR $[T_1 \rightarrow T_2]$

f_2: VAR $[T_2 \rightarrow T_3]$

f_3: VAR $[T_3 \rightarrow T_4]$

assoc: LEMMA $(f_3 \circ f_2) \circ f_1 = f_3 \circ (f_2 \circ f_1)$

END function_props2

relation_defs$[T_1, T_2:$ TYPE$]$: THEORY
BEGIN

$R, S:$ VAR pred$[[T_2]]$

$x:$ VAR T_1

$y:$ VAR T_2

$X:$ VAR set$[T_1]$

$Y:$ VAR set$[T_2]$

domain?$(R)(x: T_1)$: bool = $\exists\ (y: T_2): R(x, y)$

range?$(R)(y: T_2)$: bool = $\exists\ (x: T_1): R(x, y)$

domain(R): TYPE = $\{x: T_1 \mid \exists\ (y: T_2): R(x, y)\}$

range(R): TYPE = $\{y: T_2 \mid \exists\ (x: T_1): R(x, y)\}$

rinverse$(R)(y, x)$: bool = $R(x, y)$

rcomplement$(R)(x, y)$: bool = NOT $R(x, y)$

runion$(R, S)(x, y)$: bool = $R(x, y)$ OR $S(x, y)$

rintersection$(R, S)(x, y)$: bool = $R(x, y)$ AND $S(x, y)$

image(R, X): set$[T_2]$ =
$\{y: T_2 \mid \exists\ (x: (X)): R(x, y)\}$

image$(R)(X)$: set$[T_2]$ = image(R, X)

preimage(R, Y): set$[T_1]$ =
$\{x: T_1 \mid \exists\ (y: (Y)): R(x, y)\}$

preimage$(R)(Y)$: set$[T_1]$ = preimage(R, Y)

postcondition(R, X): set$[T_2]$ =
$\{y: T_2 \mid \exists\ (x: (X)): R(x, y)\}$

postcondition(R)(X): set[T_2] = postcondition(R, X)

precondition(R, Y): set[T_1] =
$\{x: T_1 \mid \forall\ (y: T_2 \mid R(x, y)): Y(y)\}$

precondition(R)(Y): set[T_1] = precondition(R, Y)

converse(R): pred[[T_1]] =
$(\lambda\ (y: T_2), (x: T_1): R(x, y))$

isomorphism?(R): bool =
($\exists$ (f: (bijective?[T_1, T_2])): R = graph(f))

total?(R): bool = $\forall\ (x: T_1): \exists\ (y: T_2): R(x, y)$

onto?(R): bool = $\forall\ (y: T_2): \exists\ (x: T_1): R(x, y)$

END relation_defs

relation_props$[T_1, T_2, T_3:$ TYPE$]$: THEORY
BEGIN

R_1: VAR pred$[[T_2]]$

R_2: VAR pred$[[T_3]]$

x: VAR T_1

y: VAR T_2

z: VAR T_3

$R_1 \circ R_2(x, z)$: bool = $\exists\ y: R_1(x, y)$ AND $R_2(y, z)$

total_composition: LEMMA
total?(R_1) & total?(R_2) $\Rightarrow$ total?($R_1 \circ R_2$)

onto_composition: LEMMA
onto?(R_1) & onto?(R_2) $\Rightarrow$ onto?($R_1 \circ R_2$)

composition_total: JUDGEMENT $O(R_1$: (total?$[T_1, T_2]$), R_2: (total?$[T_2, T_3]$)) HAS_TYPE
(total?$[T_1, T_3]$)

composition_onto: JUDGEMENT $O(R_1$: (onto?$[T_1, T_2]$), R_2: (onto?$[T_2, T_3]$)) HAS_TYPE
(onto?$[T_1, T_3]$)

END relation_props

relation_props2 $[T_1, T_2, T_3, T_4:$ TYPE $]:$ THEORY

BEGIN

$R_1:$ VAR pred$[[T_2]]$

$R_2:$ VAR pred$[[T_3]]$

$R_3:$ VAR pred$[[T_4]]$

assoc: LEMMA $(R_1 \circ R_2) \circ R_3 = R_1 \circ (R_2 \circ R_3)$

END relation_props2

relation_converse_props[T : TYPE] : THEORY

BEGIN

reflexive_converse : JUDGEMENT converse(R : (reflexive?[T])) HAS_TYPE (reflexive?[T])

irreflexive_converse : JUDGEMENT converse(R : (irreflexive?[T])) HAS_TYPE (irreflexive?[T])

symmetric_converse : JUDGEMENT converse(R : (symmetric?[T])) HAS_TYPE (symmetric?[T])

antisymmetric_converse : JUDGEMENT converse(R : (antisymmetric?[T])) HAS_TYPE (antisymmetric?[T])

connected_converse : JUDGEMENT converse(R : (connected?[T])) HAS_TYPE (connected?[T])

transitive_converse : JUDGEMENT converse(R : (transitive?[T])) HAS_TYPE (transitive?[T])

equivalence_converse : JUDGEMENT converse(R : (equivalence?[T])) HAS_TYPE (equivalence?[T])

preorder_converse : JUDGEMENT converse(R : (preorder?[T])) HAS_TYPE (preorder?[T])

partial_order_converse : JUDGEMENT converse(R : (partial_order?[T])) HAS_TYPE (partial_order?[T])

strict_order_converse : JUDGEMENT converse(R : (strict_order?[T])) HAS_TYPE (strict_order?[T])

dichotomous_converse : JUDGEMENT converse(R : (dichotomous?[T])) HAS_TYPE (dichotomous?[T])

total_order_converse : JUDGEMENT converse(R : (total_order?[T])) HAS_TYPE (total_order?[T])

trichotomous_converse : JUDGEMENT converse(R : (trichotomous?[T])) HAS_TYPE (trichotomous?[T])

strict_total_order_converse : JUDGEMENT converse(R : (strict_total_order?[T])) HAS_TYPE (strict_total_order?[T])

END relation_converse_props

indexed_sets[index, T: TYPE]: THEORY
BEGIN

i: VAR index

x: VAR T

A, B: VAR [index → set[T]]

S: VAR set[T]

$\bigcup A$: set[T] = $\{x \mid \exists\ i:\ A(i)(x)\}$

$\bigcap A$: set[T] = $\{x \mid \forall\ i:\ A(i)(x)\}$

IUnion_Union: LEMMA $\bigcup A = \bigcup$ image(A)(fullset[index])

IIntersection_Intersection: LEMMA
$\bigcap A = \bigcap$ image(A)(fullset[index])

IUnion_union: LEMMA
$\bigcup \lambda\ i:\ (A(i) \cup B(i)) = (\bigcup A \cup \bigcup B)$

IIntersection_intersection: LEMMA
$\bigcap \lambda\ i:\ (A(i) \cap B(i)) = (\bigcap A \cap \bigcap B)$

IUnion_intersection: LEMMA
$\bigcup \lambda\ i:\ (A(i) \cap S) = (\bigcup A \cap S)$

IIntersection_union: LEMMA
$\bigcap \lambda\ i:\ (A(i) \cup S) = (\bigcap A \cup S)$

END indexed_sets

operator_defs$[T:$ TYPE$]:$ THEORY
BEGIN

$O,\ \times:$ VAR $[T,\ T \to T]$

$-:$ VAR $[T \to T]$

$x,\ y,\ z:$ VAR T

commutative?(O): bool =
  $(\forall\ x,\ y:\ x \circ y = y \circ x)$

associative?(O): bool =
  $(\forall\ x,\ y,\ z:$
    $(x \circ y) \circ z = x \circ (y \circ z))$

left_identity?$(O)(y)$: bool = $(\forall\ x:\ y \circ x = x)$

right_identity?$(O)(y)$: bool = $(\forall\ x:\ x \circ y = x)$

identity?$(O)(y)$: bool =
  $(\forall\ x:\ x \circ y = x$ AND $y \circ x = x)$

has_identity?(O): bool = $(\exists\ y:$ identity?$(O)(y))$

zero?$(O)(y)$: bool =
  $(\forall\ x:\ x \circ y = y$ AND $y \circ x = y)$

has_zero?(O): bool = $(\exists\ y:$ zero?$(O)(y))$

inverses?$(O)(-)(y)$: bool =
  $(\forall\ x:\ x \circ -x = y$ AND $(-x) \circ x = y)$

has_inverses?(O): bool = $(\exists\ -,\ y:$ inverses?$(O)(-)(y))$

distributive?$(\times,\ O)$: bool =
  $(\forall\ x,\ y,\ z:$
    $x \times (y \circ z) =$
      $(x \times y) \circ (x \times z))$

END operator_defs

```
numbers: THEORY
  BEGIN

    number: TYPE+

  END numbers
```

number_fields: THEORY
BEGIN

number_field: TYPE+ FROM number

numfield: TYPE+ = number_field

number_field?(n: number): bool = number_field_pred(n)

nonzero_number: TYPE+ = $\{r$: number_field $\mid r \neq 0\}$ CONTAINING 1

nznum: TYPE+ = nonzero_number

$+, -, \times$: [numfield, numfield $\rightarrow$ numfield]

$/$: [numfield, nznum $\rightarrow$ numfield]

$-$: [numfield $\rightarrow$ numfield]

x, y, z: VAR numfield

n0x: VAR nznum

commutative_add: POSTULATE $x + y = y + x$

associative_add: POSTULATE
$x + (y + z) = (x + y) + z$

identity_add: POSTULATE $x + 0 = x$

inverse_add: AXIOM $x + -x = 0$

minus_add: AXIOM $x - y = x + -y$

commutative_mult: AXIOM $x \times y = y \times x$

associative_mult: AXIOM
$x \times (y \times z) = (x \times y) \times z$

identity_mult: AXIOM $1 \times x = x$

inverse_mult: AXIOM n0x $\times$ (1/n0x) $= 1$

div_def : AXIOM $y/\text{n0x} = y \times (1/\text{n0x})$

distributive : POSTULATE
$x \times (y + z) = (x \times y) + (x \times z)$

END number_fields

reals: THEORY

BEGIN

$\mathbb{R}$: TYPE+ FROM number_field

real?(n: number): bool = number_field_pred(n) AND real_pred(n)

$\mathbb{R}_{\neq 0}$: TYPE+ = $\{r\colon \mathbb{R} \mid r \neq 0\}$ CONTAINING 1

$\mathbb{R}_{\neq 0}$: TYPE+ = $\mathbb{R}_{\neq 0}$

x, y: VAR $\mathbb{R}$

n0z: VAR $\mathbb{R}_{\neq 0}$

closed_plus: AXIOM real_pred($x + y$)

closed_minus: AXIOM real_pred($x - y$)

closed_times: AXIOM real_pred($x \times y$)

closed_divides: AXIOM real_pred(x/n0z)

closed_neg: AXIOM real_pred($-x$)

real_plus_real_is_real: JUDGEMENT $+(x, y)$ HAS_TYPE $\mathbb{R}$

real_minus_real_is_real: JUDGEMENT $-(x, y)$ HAS_TYPE $\mathbb{R}$

real_times_real_is_real: JUDGEMENT $\times(x, y)$ HAS_TYPE $\mathbb{R}$

real_div_nzreal_is_real: JUDGEMENT $/(x$, n0z) HAS_TYPE $\mathbb{R}$

minus_real_is_real: JUDGEMENT $-(x)$ HAS_TYPE $\mathbb{R}$

$<(x, y)$: bool

$\leq(x, y)$: bool = $x < y$ OR $x = y$;

$>(x, y)$: bool = $y < x$;

$\geq(x, y)$: bool = $y \leq x$

reals_totally_ordered : POSTULATE strict_total_order?(<)

END reals

real_axioms : THEORY
BEGIN

x, y, z : VAR $\mathbb{R}$

n0x : VAR $\mathbb{R}_{\neq 0}$

posreal_add_closed : POSTULATE
$x > 0$ AND $y > 0$ IMPLIES $x + y > 0$

posreal_mult_closed : AXIOM $x > 0$ AND $y > 0$ IMPLIES $x \times y > 0$

posreal_neg : POSTULATE $x > 0$ IMPLIES NOT $-x > 0$

trichotomy : POSTULATE $x > 0$ OR $x = 0$ OR $0 > x$

END real_axioms

```
bounded_real_defs: THEORY
 BEGIN

  x, y: VAR ℝ

  S: VAR (nonempty?[ℝ])

  upper_bound?(x, S): bool = ∀ (s: (S)): s ≤ x

  upper_bound?(S)(x): bool = upper_bound?(x, S)

  lower_bound?(x, S): bool = ∀ (s: (S)): x ≤ s

  lower_bound?(S)(x): bool = lower_bound?(x, S)

  least_upper_bound?(x, S): bool =
        upper_bound?(x, S) AND
         ∀ y: upper_bound?(y, S) IMPLIES (x ≤ y)

  least_upper_bound?(S)(x): bool = least_upper_bound?(x, S)

  greatest_lower_bound?(x, S): bool =
        lower_bound?(x, S) AND
         ∀ y: lower_bound?(y, S) IMPLIES (y ≤ x)

  greatest_lower_bound?(S)(x): bool =
        greatest_lower_bound?(x, S)

  real_complete: AXIOM
    ∀ S:
        (∃ y: upper_bound?(y, S)) IMPLIES
         (∃ y: least_upper_bound?(y, S))

  real_lower_complete: LEMMA
    ∀ S:
        (∃ y: lower_bound?(y, S)) IMPLIES
         (∃ x: greatest_lower_bound?(x, S))

  bounded_above?(S): bool = (∃ x: upper_bound?(x, S))

  bounded_below?(S): bool = (∃ x: lower_bound?(x, S))
```

bounded?(S): bool = bounded_above?(S) AND bounded_below?(S)

bounded_set: TYPE = (bounded?)

SA: VAR (bounded_above?)

lub_exists: LEMMA ($\exists$ x: least_upper_bound?(x, SA))

lub(SA): {x | least_upper_bound?(x, SA)}

lub_lem: LEMMA lub(SA) $=$ x IFF least_upper_bound?(x, SA)

SB: VAR (bounded_below?)

glb_exists: LEMMA ($\exists$ x: greatest_lower_bound?(x, SB))

glb(SB): {x | greatest_lower_bound?(x, SB)}

glb_lem: LEMMA glb(SB) $=$ x IFF greatest_lower_bound?(x, SB)

END bounded_real_defs

```
bounded_real_defs_alt[S: (nonempty?[ℝ])]: THEORY
 BEGIN

  x: VAR ℝ

  upper_bound?: [ℝ → bool] = upper_bound?(S)

  lower_bound?: [ℝ → bool] = lower_bound?(S)

  least_upper_bound?: [ℝ → bool] = least_upper_bound?(S)

  greatest_lower_bound?: [ℝ → bool] = greatest_lower_bound?(S)

  lub_is_upper_bound: JUDGEMENT (least_upper_bound?) SUBTYPE_OF
        (upper_bound?)

  glb_is_lower_bound: JUDGEMENT (greatest_lower_bound?) SUBTYPE_OF
        (lower_bound?)

 END bounded_real_defs_alt
```

real_types: THEORY

BEGIN

x: VAR $\mathbb{R}$

$\mathbb{R}_{\geq 0}$: TYPE+ = $\{x\colon \mathbb{R} \mid x \geq 0\}$ CONTAINING 0

$\mathbb{R}_{\leq 0}$: TYPE+ = $\{x\colon \mathbb{R} \mid x \leq 0\}$ CONTAINING 0

$\mathbb{R}_{>0}$: TYPE+ = $\{x\colon \mathbb{R}_{\geq 0} \mid x > 0\}$ CONTAINING 1

$\mathbb{R}_{<0}$: TYPE+ = $\{x\colon \mathbb{R}_{\leq 0} \mid x < 0\}$ CONTAINING -1

$\mathbb{R}_{\geq 0}$: TYPE = $\mathbb{R}_{\geq 0}$

$\mathbb{R}_{\leq 0}$: TYPE = $\mathbb{R}_{\leq 0}$

posreal_is_nzreal: JUDGEMENT $\mathbb{R}_{>0}$ SUBTYPE_OF $\mathbb{R}_{\neq 0}$

negreal_is_nzreal: JUDGEMENT $\mathbb{R}_{<0}$ SUBTYPE_OF $\mathbb{R}_{\neq 0}$

nzx, nzy: VAR $\mathbb{R}_{\neq 0}$

px, py: VAR $\mathbb{R}_{>0}$

nx, ny: VAR $\mathbb{R}_{<0}$

nnx, nny: VAR $\mathbb{R}_{\geq 0}$

npx, npy: VAR $\mathbb{R}_{\leq 0}$

nonneg_real_add_closed: LEMMA $\text{nnx} + \text{nny} \geq 0$

nonpos_real_add_closed: LEMMA $\text{npx} + \text{npy} \leq 0$

negreal_add_closed: LEMMA $\text{nx} + \text{ny} < 0$

nonneg_real_mult_closed: LEMMA $\text{nnx} \times \text{nny} \geq 0$

nzreal_times_nzreal_is_nzreal: JUDGEMENT $\times$(nzx, nzy) HAS_TYPE $\mathbb{R}_{\neq 0}$

nzreal_div_nzreal_is_nzreal: JUDGEMENT /(nzx, nzy) HAS_TYPE $\mathbb{R}_{\neq 0}$

minus_nzreal_is_nzreal: JUDGEMENT $-$(nzx) HAS_TYPE $\mathbb{R}_{\neq 0}$

nnreal_plus_nnreal_is_nnreal: JUDGEMENT $+$(nnx, nny) HAS_TYPE $\mathbb{R}_{\geq 0}$

nnreal_times_nnreal_is_nnreal: JUDGEMENT $\times$(nnx, nny) HAS_TYPE $\mathbb{R}_{\geq 0}$

nnreal_div_posreal_is_nnreal: JUDGEMENT /(nnx, py) HAS_TYPE $\mathbb{R}_{\geq 0}$

nnreal_div_negreal_is_npreal: JUDGEMENT /(nnx, ny) HAS_TYPE $\mathbb{R}_{\leq 0}$

npreal_plus_npreal_is_npreal: JUDGEMENT $+$(npx, npy) HAS_TYPE $\mathbb{R}_{\leq 0}$

npreal_times_npreal_is_nnreal: JUDGEMENT $\times$(npx, npy) HAS_TYPE $\mathbb{R}_{\geq 0}$

npreal_div_posreal_is_npreal: JUDGEMENT /(npx, py) HAS_TYPE $\mathbb{R}_{\leq 0}$

npreal_div_negreal_is_nnreal: JUDGEMENT /(npx, ny) HAS_TYPE $\mathbb{R}_{\geq 0}$

posreal_plus_nnreal_is_posreal: JUDGEMENT $+$(px, nny) HAS_TYPE $\mathbb{R}_{>0}$

nnreal_plus_posreal_is_posreal: JUDGEMENT $+$(nnx, py) HAS_TYPE $\mathbb{R}_{>0}$

posreal_times_posreal_is_posreal: JUDGEMENT $\times$(px, py) HAS_TYPE $\mathbb{R}_{>0}$

posreal_div_posreal_is_posreal: JUDGEMENT /(px, py) HAS_TYPE $\mathbb{R}_{>0}$

negreal_plus_negreal_is_negreal: JUDGEMENT $+$(nx, ny) HAS_TYPE $\mathbb{R}_{<0}$

negreal_times_negreal_is_posreal: JUDGEMENT $\times$(nx, ny) HAS_TYPE $\mathbb{R}_{>0}$

negreal_div_negreal_is_posreal: JUDGEMENT /(nx, ny) HAS_TYPE $\mathbb{R}_{>0}$

END real_types

rationals : THEORY
BEGIN

$\mathbb{Q}$: TYPE+ FROM $\mathbb{R}$

$\mathbb{Q}$: TYPE+ = $\mathbb{Q}$

rational?(n: number): bool =
number_field_pred(n) AND real_pred(n) AND rational_pred(n)

$\mathbb{Q}_{\neq 0}$: TYPE+ = $\{r: \mathbb{Q} \mid r \neq 0\}$ CONTAINING 1

$\mathbb{Q}_{\neq 0}$: TYPE+ = $\mathbb{Q}_{\neq 0}$

x, y: VAR $\mathbb{Q}$

n0z: VAR $\mathbb{Q}_{\neq 0}$

closed_plus: AXIOM rational_pred($x + y$)

closed_minus: AXIOM rational_pred($x - y$)

closed_times: AXIOM rational_pred($x \times y$)

closed_divides: AXIOM rational_pred(x/n0z)

closed_neg: AXIOM rational_pred($-x$)

rat_plus_rat_is_rat: JUDGEMENT $+(x, y)$ HAS_TYPE $\mathbb{Q}$

rat_minus_rat_is_rat: JUDGEMENT $-(x, y)$ HAS_TYPE $\mathbb{Q}$

rat_times_rat_is_rat: JUDGEMENT $\times(x, y)$ HAS_TYPE $\mathbb{Q}$

rat_div_nzrat_is_rat: JUDGEMENT $/(x,$ n0z$)$ HAS_TYPE $\mathbb{Q}$

minus_rat_is_rat: JUDGEMENT $-(x)$ HAS_TYPE $\mathbb{Q}$

$\mathbb{Q}_{\geq 0}$: TYPE+ = $\{r: \mathbb{Q} \mid r \geq 0\}$ CONTAINING 0

$\mathbb{Q}_{\leq 0}$: TYPE+ = $\{r: \mathbb{Q} \mid r \leq 0\}$ CONTAINING 0

$\mathbb{Q}_{>0}$: TYPE+ = $\{r\colon \mathbb{Q}_{\geq 0} \mid r > 0\}$ CONTAINING 1

$\mathbb{Q}_{<0}$: TYPE+ = $\{r\colon \mathbb{Q}_{\leq 0} \mid r < 0\}$ CONTAINING -1

$\mathbb{Q}_{\geq 0}$: TYPE+ = $\mathbb{Q}_{\geq 0}$

$\mathbb{Q}_{\leq 0}$: TYPE+ = $\mathbb{Q}_{\leq 0}$

nnx, nny: VAR $\mathbb{Q}_{\geq 0}$

npx, npy: VAR $\mathbb{Q}_{\leq 0}$

px, py: VAR $\mathbb{Q}_{>0}$

nx, ny: VAR $\mathbb{Q}_{<0}$

n0x, n0y: VAR $\mathbb{Q}_{\neq 0}$

posrat_is_nzrat: JUDGEMENT $\mathbb{Q}_{>0}$ SUBTYPE_OF $\mathbb{Q}_{\neq 0}$

negrat_is_nzrat: JUDGEMENT $\mathbb{Q}_{<0}$ SUBTYPE_OF $\mathbb{Q}_{\neq 0}$

nzrat_times_nzrat_is_nzrat: JUDGEMENT $\times$(n0x, n0y) HAS_TYPE $\mathbb{Q}_{\neq 0}$

nzrat_div_nzrat_is_nzrat: JUDGEMENT /(n0x, n0y) HAS_TYPE $\mathbb{Q}_{\neq 0}$

minus_nzrat_is_nzrat: JUDGEMENT $-$(n0x) HAS_TYPE $\mathbb{Q}_{\neq 0}$

nnrat_plus_nnrat_is_nnrat: JUDGEMENT $+$(nnx, nny) HAS_TYPE $\mathbb{Q}_{\geq 0}$

nnrat_times_nnrat_is_nnrat: JUDGEMENT $\times$(nnx, nny) HAS_TYPE $\mathbb{Q}_{\geq 0}$

nnrat_div_posrat_is_nnrat: JUDGEMENT /(nnx, py) HAS_TYPE $\mathbb{Q}_{\geq 0}$

nnrrat_div_negrat_is_nprat: JUDGEMENT /(nnx, ny) HAS_TYPE $\mathbb{Q}_{\leq 0}$

nprat_plus_nprat_is_nprat: JUDGEMENT $+$(npx, npy) HAS_TYPE $\mathbb{Q}_{\leq 0}$

nprat_times_nprat_is_nnrat: JUDGEMENT $\times$(npx, npy) HAS_TYPE $\mathbb{Q}_{\geq 0}$

nprat_div_posrat_is_nprat: JUDGEMENT /(npx, py) HAS_TYPE $\mathbb{Q}_{\leq 0}$

nprat_div_negrat_is_nnrat: JUDGEMENT /(npx, ny) HAS_TYPE $\mathbb{Q}_{\geq 0}$

posrat_plus_nnrat_is_posrat: JUDGEMENT +(px, nny) HAS_TYPE $\mathbb{Q}_{>0}$

nnrat_plus_posrat_is_posrat: JUDGEMENT +(nnx, py) HAS_TYPE $\mathbb{Q}_{>0}$

posrat_times_posrat_is_posrat: JUDGEMENT ×(px, py) HAS_TYPE $\mathbb{Q}_{>0}$

posrat_div_posrat_is_posrat: JUDGEMENT /(px, py) HAS_TYPE $\mathbb{Q}_{>0}$

negrat_plus_negrat_is_negrat: JUDGEMENT +(nx, ny) HAS_TYPE $\mathbb{Q}_{<0}$

negrat_times_negrat_is_posrat: JUDGEMENT ×(nx, ny) HAS_TYPE $\mathbb{Q}_{>0}$

negrat_div_negrat_is_posrat: JUDGEMENT /(nx, ny) HAS_TYPE $\mathbb{Q}_{>0}$

END rationals

integers: THEORY
BEGIN

$\mathbb{Z}$: TYPE+ FROM $\mathbb{Q}$

$\mathbb{Z}$: TYPE+ = $\mathbb{Z}$

integer?(n: number): bool =
number_field_pred(n) AND
real_pred(n) AND rational_pred(n) AND integer_pred(n)

nonzero_integer: TYPE+ = $\{i: \mathbb{Z} \mid i \neq 0\}$ CONTAINING 1

$\mathbb{Z}_{\neq 0}$: TYPE+ = nonzero_integer

i, j: VAR $\mathbb{Z}$

n0i, n0j: VAR $\mathbb{Z}_{\neq 0}$

closed_plus: AXIOM integer_pred($i + j$)

closed_minus: AXIOM integer_pred($i - j$)

closed_times: AXIOM integer_pred($i \times j$)

closed_neg: AXIOM integer_pred($-i$)

upfrom(i): TYPE+ = $\{s: \mathbb{Z} \mid s \geq i\}$ CONTAINING i

above(i): TYPE+ = $\{s: \mathbb{Z} \mid s > i\}$ CONTAINING $i + 1$

int_plus_int_is_int: JUDGEMENT $+(i, j)$ HAS_TYPE $\mathbb{Z}$

int_minus_int_is_int: JUDGEMENT $-(i, j)$ HAS_TYPE $\mathbb{Z}$

int_times_int_is_int: JUDGEMENT $\times(i, j)$ HAS_TYPE $\mathbb{Z}$

minus_int_is_int: JUDGEMENT $-(i)$ HAS_TYPE $\mathbb{Z}$

minus_nzint_is_nzint: JUDGEMENT $-$(n0i) HAS_TYPE $\mathbb{Z}_{\neq 0}$

$\mathbb{N}$: TYPE+ = $\{i: \mathbb{Z} \mid i \geq 0\}$ CONTAINING 0

$\mathbb{Z}_{\neq 0}$: TYPE+ = $\{i\colon \mathbb{Z} \mid i \leq 0\}$ CONTAINING 0

$\mathbb{N}_{>0}$: TYPE+ = $\{i\colon \mathbb{N} \mid i > 0\}$ CONTAINING 1

$\mathbb{Z}_{<0}$: TYPE+ = $\{i\colon \mathbb{Z}_{\neq 0} \mid i < 0\}$ CONTAINING -1

$\mathbb{N}_{>0}$: TYPE+ = $\mathbb{N}_{>0}$

nni, nnj: VAR $\mathbb{N}$

npi, npj: VAR $\mathbb{Z}_{\neq 0}$

π, pj: VAR $\mathbb{N}_{>0}$

ni, nj: VAR $\mathbb{Z}_{<0}$

posint_is_nzint: JUDGEMENT $\mathbb{N}_{>0}$ SUBTYPE_OF $\mathbb{Z}_{\neq 0}$

negint_is_nzint: JUDGEMENT $\mathbb{Z}_{<0}$ SUBTYPE_OF $\mathbb{Z}_{\neq 0}$

nzint_times_nzint_is_nzint: JUDGEMENT $\times$(n0i, n0j) HAS_TYPE $\mathbb{Z}_{\neq 0}$

nnint_plus_nnint_is_nnint: JUDGEMENT $+$(nni, nnj) HAS_TYPE $\mathbb{N}$

nnint_times_nnint_is_nnint: JUDGEMENT $\times$(nni, nnj) HAS_TYPE $\mathbb{N}$

npint_plus_npint_is_npint: JUDGEMENT $+$(npi, npj) HAS_TYPE $\mathbb{Z}_{\neq 0}$

npint_times_npint_is_nnint: JUDGEMENT $\times$(npi, npj) HAS_TYPE $\mathbb{N}$

posint_plus_nnint_is_posint: JUDGEMENT $+$(π, nnj) HAS_TYPE $\mathbb{N}_{>0}$

nnint_plus_posint_is_posint: JUDGEMENT $+$(nni, pj) HAS_TYPE $\mathbb{N}_{>0}$

posint_times_posint_is_posint: JUDGEMENT $\times$(π, pj) HAS_TYPE $\mathbb{N}_{>0}$

negint_plus_negint_is_negint: JUDGEMENT $+$(ni, nj) HAS_TYPE $\mathbb{Z}_{<0}$

negint_times_negint_is_posint: JUDGEMENT $\times$(ni, nj) HAS_TYPE $\mathbb{N}_{>0}$

subrange(i, j): TYPE = $\{k\colon \mathbb{Z} \mid i \leq k$ AND $k \leq j\}$

even?(i): bool = $\exists\ j:\ i = j \times 2$

odd?(i): bool = $\exists\ j:\ i = j \times 2 + 1$

even_int: TYPE+ = (even?) CONTAINING 0

odd_int: TYPE+ = (odd?) CONTAINING 1

e_1, e_2: VAR even_int

o_1, o_2: VAR odd_int

odd_is_nzint: JUDGEMENT odd_int SUBTYPE_OF $\mathbb{Z}_{\neq 0}$

even_plus_even_is_even: JUDGEMENT $+(e_1,\ e_2)$ HAS_TYPE even_int

even_minus_even_is_even: JUDGEMENT $-(e_1,\ e_2)$ HAS_TYPE even_int

odd_plus_odd_is_even: JUDGEMENT $+(o_1,\ o_2)$ HAS_TYPE even_int

odd_minus_odd_is_even: JUDGEMENT $-(o_1,\ o_2)$ HAS_TYPE even_int

odd_plus_even_is_odd: JUDGEMENT $+(o_1,\ e_2)$ HAS_TYPE odd_int

odd_minus_even_is_odd: JUDGEMENT $-(o_1,\ e_2)$ HAS_TYPE odd_int

even_plus_odd_is_odd: JUDGEMENT $+(e_1,\ o_2)$ HAS_TYPE odd_int

even_minus_odd_is_odd: JUDGEMENT $-(e_1,\ o_2)$ HAS_TYPE odd_int

even_times_int_is_even: JUDGEMENT $\times(e_1,\ i)$ HAS_TYPE even_int

int_times_even_is_even: JUDGEMENT $\times(i,\ e_2)$ HAS_TYPE even_int

odd_times_odd_is_odd: JUDGEMENT $\times(o_1,\ o_2)$ HAS_TYPE odd_int

minus_even_is_even: JUDGEMENT $-(e_1)$ HAS_TYPE even_int

minus_odd_is_odd: JUDGEMENT $-(o_1)$ HAS_TYPE odd_int

END integers

naturalnumbers: THEORY
BEGIN

$\mathbb{N}$: TYPE = $\mathbb{N}$

$\mathbb{N}$: TYPE+ = $\mathbb{N}$

i, j, k: VAR $\mathbb{N}$

n: VAR $\mathbb{N}_{>0}$

upfrom_nat_is_nat: JUDGEMENT upfrom(i) SUBTYPE_OF $\mathbb{N}$

upfrom_posnat_is_posnat: JUDGEMENT upfrom(n) SUBTYPE_OF $\mathbb{N}_{>0}$

above_nat_is_posnat: JUDGEMENT above(i) SUBTYPE_OF $\mathbb{N}_{>0}$

subrange_nat_is_nat: JUDGEMENT subrange(i, j) SUBTYPE_OF $\mathbb{N}$

subrange_posnat_is_posnat: JUDGEMENT subrange(n, j) SUBTYPE_OF $\mathbb{N}_{>0}$

upto(i): TYPE+ = $\{s: \mathbb{N} \mid s \leq i\}$ CONTAINING i

below(i): TYPE = $\{s: \mathbb{N} \mid s < i\}$

succ(i): $\mathbb{N}$ = $i+1$

pred(i): $\mathbb{N}$ = IF $i > 0$ THEN $i-1$ ELSE 0 ENDIF;

~(i, j): $\mathbb{N}$ = IF $i > j$ THEN $i-j$ ELSE 0 ENDIF

wf_nat: AXIOM well_founded?(λ i, j: $i < j$)

p: VAR pred$[\mathbb{N}]$

nat_induction: LEMMA
  ($p(0)$ AND ($\forall$ j: $p(j)$ IMPLIES $p(j+1)$)) IMPLIES
    ($\forall$ i: $p(i)$)

NAT_induction: LEMMA
  ($\forall$ j: ($\forall$ k: $k < j$ IMPLIES $p(k)$) IMPLIES $p(j)$) IMPLIES
    ($\forall$ i: $p(i)$)

even_nat: TYPE+ = $\{i \mid$ even?$(i)\}$ CONTAINING 0

even_posnat: TYPE+ = $\{n \mid$ even?$(n)\}$ CONTAINING 2

odd_posnat: TYPE+ = $\{n \mid$ odd?$(n)\}$ CONTAINING 1

even_negint: TYPE+ = $\{n\colon \mathbb{Z}_{<0} \mid$ even?$(n)\}$ CONTAINING -2

odd_negint: TYPE+ = $\{n\colon \mathbb{Z}_{<0} \mid$ odd?$(n)\}$ CONTAINING -1

x: VAR $\mathbb{Z}$

even_or_odd: THEOREM even?(x) IFF NOT odd?(x)

odd_iff_not_even: LEMMA odd?(x) IFF NOT even?(x)

even_iff_not_odd: LEMMA even?(x) IFF NOT odd?(x)

odd_or_even_int: LEMMA odd?(x) OR even?(x)

odd_iff_even_succ: LEMMA odd?(x) IFF even?$(x+1)$

even_iff_odd_succ: LEMMA even?(x) IFF odd?$(x+1)$

even_plus1: LEMMA even?(x) IFF NOT even?$(x+1)$

odd_plus1: LEMMA odd?(x) IFF NOT odd?$(x+1)$

even_div2: LEMMA even?(x) IMPLIES integer_pred$(x/2)$

odd_div2: LEMMA odd?(x) IMPLIES integer_pred$((x-1)/2)$

END naturalnumbers

```
min_nat[T: TYPE FROM ℕ]: THEORY
 BEGIN

  S: VAR (nonempty?[T])

  a, x: VAR T

  min(S): {a | S(a) AND (∀ x: S(x) IMPLIES a ≤ x)}

  minimum?(a, S): bool = S(a) AND (∀ x: S(x) IMPLIES a ≤ x)

  min_def: LEMMA
    ∀ (S: (nonempty?[T])): min(S) = a IFF minimum?(a, S)

 END min_nat
```

real_defs: THEORY
BEGIN

m, n: VAR $\mathbb{R}$

sgn(m): $\mathbb{Z}$ = IF $m \geq 0$ THEN 1 ELSE -1 ENDIF

$|m|$: $\{n: \mathbb{R}_{\geq 0} \mid n \geq m$ AND $n \geq -m\}$ =
IF $m < 0$ THEN $-m$ ELSE m ENDIF

nonzero_abs_is_pos: JUDGEMENT abs(x: $\mathbb{R}_{\neq 0}$) HAS_TYPE
$\{y: \mathbb{R}_{>0} \mid y \geq x\}$

rat_abs_is_nonneg: JUDGEMENT abs(q: $\mathbb{Q}$) HAS_TYPE $\{r: \mathbb{Q}_{\geq 0} \mid r \geq q\}$

nzrat_abs_is_pos: JUDGEMENT abs(q: $\mathbb{Q}_{\neq 0}$) HAS_TYPE $\{r: \mathbb{Q}_{>0} \mid r \geq q\}$

int_abs_is_nonneg: JUDGEMENT abs(i: $\mathbb{Z}$) HAS_TYPE $\{j: \mathbb{N} \mid j \geq i\}$

nzint_abs_is_pos: JUDGEMENT abs(i: $\mathbb{Z}_{\neq 0}$) HAS_TYPE $\{j: \mathbb{N}_{>0} \mid j \geq i\}$

max(m, n): $\{p: \mathbb{R} \mid p \geq m$ AND $p \geq n\}$ =
IF $m < n$ THEN n ELSE m ENDIF

min(m, n): $\{p: \mathbb{R} \mid p \leq m$ AND $p \leq n\}$ =
IF $m > n$ THEN n ELSE m ENDIF

nzreal_max: JUDGEMENT max(x, y: $\mathbb{R}_{\neq 0}$) HAS_TYPE
$\{z: \mathbb{R}_{\neq 0} \mid z \geq x$ AND $z \geq y\}$

nzreal_min: JUDGEMENT min(x, y: $\mathbb{R}_{\neq 0}$) HAS_TYPE
$\{z: \mathbb{R}_{\neq 0} \mid z \leq x$ AND $z \leq y\}$

nonneg_real_max: JUDGEMENT max(x, y: $\mathbb{R}_{\geq 0}$) HAS_TYPE
$\{z: \mathbb{R}_{\geq 0} \mid z \geq x$ AND $z \geq y\}$

nonneg_real_min: JUDGEMENT min(x, y: $\mathbb{R}_{\geq 0}$) HAS_TYPE
$\{z: \mathbb{R}_{\geq 0} \mid z \leq x$ AND $z \leq y\}$

posreal_max: JUDGEMENT max(x, y: $\mathbb{R}_{>0}$) HAS_TYPE
$\{z: \mathbb{R}_{>0} \mid z \geq x$ AND $z \geq y\}$

posreal_min: JUDGEMENT min(x, y: $\mathbb{R}_{>0}$) HAS_TYPE
{z: $\mathbb{R}_{>0}$ | $z \leq x$ AND $z \leq y$}

rat_max: JUDGEMENT max(q, r: $\mathbb{Q}$) HAS_TYPE
{s: $\mathbb{Q}$ | $s \geq q$ AND $s \geq r$}

rat_min: JUDGEMENT min(q, r: $\mathbb{Q}$) HAS_TYPE
{s: $\mathbb{Q}$ | $s \leq q$ AND $s \leq r$}

nzrat_max: JUDGEMENT max(q, r: $\mathbb{Q}_{\neq 0}$) HAS_TYPE
{s: $\mathbb{Q}_{\neq 0}$ | $s \geq q$ AND $s \geq r$}

nzrat_min: JUDGEMENT min(q, r: $\mathbb{Q}_{\neq 0}$) HAS_TYPE
{s: $\mathbb{Q}_{\neq 0}$ | $s \leq q$ AND $s \leq r$}

nonneg_rat_max: JUDGEMENT max(q, r: $\mathbb{Q}_{\geq 0}$) HAS_TYPE
{s: $\mathbb{Q}_{\geq 0}$ | $s \geq q$ AND $s \geq r$}

nonneg_rat_min: JUDGEMENT min(q, r: $\mathbb{Q}_{\geq 0}$) HAS_TYPE
{s: $\mathbb{Q}_{\geq 0}$ | $s \leq q$ AND $s \leq r$}

posrat_max: JUDGEMENT max(q, r: $\mathbb{Q}_{>0}$) HAS_TYPE
{s: $\mathbb{Q}_{>0}$ | $s \geq q$ AND $s \geq r$}

posrat_min: JUDGEMENT min(q, r: $\mathbb{Q}_{>0}$) HAS_TYPE
{s: $\mathbb{Q}_{>0}$ | $s \leq q$ AND $s \leq r$}

int_max: JUDGEMENT max(i, j: $\mathbb{Z}$) HAS_TYPE
{k: $\mathbb{Z}$ | $i \leq k$ AND $j \leq k$}

int_min: JUDGEMENT min(i, j: $\mathbb{Z}$) HAS_TYPE
{k: $\mathbb{Z}$ | $k \leq i$ AND $k \leq j$}

nzint_max: JUDGEMENT max(i, j: $\mathbb{Z}_{\neq 0}$) HAS_TYPE
{k: $\mathbb{Z}_{\neq 0}$ | $i \leq k$ AND $j \leq k$}

nzint_min: JUDGEMENT min(i, j: $\mathbb{Z}_{\neq 0}$) HAS_TYPE
{k: $\mathbb{Z}_{\neq 0}$ | $k \leq i$ AND $k \leq j$}

nat_max: JUDGEMENT max(i, j: $\mathbb{N}$) HAS_TYPE
{k: $\mathbb{N}$ | $i \leq k$ AND $j \leq k$}

nat_min: JUDGEMENT min(i, j: $\mathbb{N}$) HAS_TYPE
$\{k: \mathbb{N} \mid k \leq i$ AND $k \leq j\}$

posint_max: JUDGEMENT max(i, j: $\mathbb{N}_{>0}$) HAS_TYPE
$\{k: \mathbb{N}_{>0} \mid i \leq k$ AND $j \leq k\}$

posint_min: JUDGEMENT min(i, j: $\mathbb{N}_{>0}$) HAS_TYPE
$\{k: \mathbb{N}_{>0} \mid k \leq i$ AND $k \leq j\}$

a, b, c: VAR $\mathbb{R}$

min_le: LEMMA min(a, b) $\leq c$ IFF ($a \leq c$ OR $b \leq c$)

min_lt: LEMMA min(a, b) $< c$ IFF ($a < c$ OR $b < c$)

min_ge: LEMMA min(a, b) $\geq c$ IFF ($a \geq c$ AND $b \geq c$)

min_gt: LEMMA min(a, b) $> c$ IFF ($a > c$ AND $b > c$)

le_min: LEMMA $a \leq$ min(b, c) IFF ($a \leq b$ AND $a \leq c$)

lt_min: LEMMA $a <$ min(b, c) IFF ($a < b$ AND $a < c$)

ge_min: LEMMA $a \geq$ min(b, c) IFF ($a \geq b$ OR $a \geq c$)

gt_min: LEMMA $a >$ min(b, c) IFF ($a > b$ OR $a > c$)

max_le: LEMMA max(a, b) $\leq c$ IFF ($a \leq c$ AND $b \leq c$)

max_lt: LEMMA max(a, b) $< c$ IFF ($a < c$ AND $b < c$)

max_ge: LEMMA max(a, b) $\geq c$ IFF ($a \geq c$ OR $b \geq c$)

max_gt: LEMMA max(a, b) $> c$ IFF ($a > c$ OR $b > c$)

le_max: LEMMA $a \leq$ max(b, c) IFF ($a \leq b$ OR $a \leq c$)

lt_max: LEMMA $a <$ max(b, c) IFF ($a < b$ OR $a < c$)

ge_max: LEMMA $a \geq$ max(b, c) IFF ($a \geq b$ AND $a \geq c$)

gt_max: LEMMA $a >$ max(b, c) IFF ($a > b$ AND $a > c$)

END real_defs

real_props: THEORY
BEGIN

w, x, y, z: VAR $\mathbb{R}$

n0w, n0x, n0y, n0z: VAR $\mathbb{R}_{\neq 0}$

nnw, nnx, nny, nnz: VAR $\mathbb{R}_{\geq 0}$

pw, px, py, pz: VAR $\mathbb{R}_{>0}$

npw, npx, npy, npz: VAR $\mathbb{R}_{\leq 0}$

nw, nx, ny, nz: VAR $\mathbb{R}_{<0}$

inv_ne_0: LEMMA $1/\text{n0x} \neq 0$

both_sides_plus1: LEMMA $(x + z = y + z)$ IFF $x = y$

both_sides_plus2: LEMMA $(z + x = z + y)$ IFF $x = y$

both_sides_minus1: LEMMA $(x - z = y - z)$ IFF $x = y$

both_sides_minus2: LEMMA $(z - x = z - y)$ IFF $x = y$

both_sides_times1: LEMMA
$(x \times \text{n0z} = y \times \text{n0z})$ IFF $x = y$

both_sides_times2: LEMMA
$(\text{n0z} \times x = \text{n0z} \times y)$ IFF $x = y$

both_sides_div1: LEMMA $(x/\text{n0z} = y/\text{n0z})$ IFF $x = y$

both_sides_div2: LEMMA
$(\text{n0z}/\text{n0x} = \text{n0z}/\text{n0y})$ IFF $\text{n0x} = \text{n0y}$

times_plus: LEMMA
$(x + y) \times (z + w) =$
$x \times z + x \times w + y \times z + y \times w$

times_div1: LEMMA
$x \times (y/\text{n0z}) = (x \times y)/\text{n0z}$

times_div2 : LEMMA
$(x/\text{n0z}) \times y = (x \times y)/\text{n0z}$

div_times : LEMMA
$(x/\text{n0x}) \times (y/\text{n0y}) =$
$(x \times y)/(\text{n0x} \times \text{n0y})$

div_eq_zero : LEMMA $x/\text{n0z} = 0$ IFF $x = 0$

div_simp : LEMMA $\text{n0x}/\text{n0x} = 1$

div_cancel1 : LEMMA $\text{n0z} \times (x/\text{n0z}) = x$

div_cancel2 : LEMMA $(x/\text{n0z}) \times \text{n0z} = x$

div_cancel3 : LEMMA $x/\text{n0z} = y$ IFF $x = y \times \text{n0z}$

div_cancel4 : LEMMA $x = y/\text{n0z}$ IFF $x \times \text{n0z} = y$

cross_mult : LEMMA
$(x/\text{n0x} = y/\text{n0y})$ IFF
$(x \times \text{n0y} = y \times \text{n0x})$

add_div : LEMMA
$(x/\text{n0x}) + (y/\text{n0y}) =$
$(x \times \text{n0y} + y \times \text{n0x})/(\text{n0x} \times \text{n0y})$

minus_div1 : LEMMA
$(x/\text{n0x}) - (y/\text{n0y}) =$
$(x \times \text{n0y} - y \times \text{n0x})/(\text{n0x} \times \text{n0y})$

minus_div2 : LEMMA
$(x/\text{n0x} - y/\text{n0x}) = (x - y)/\text{n0x}$

div_distributes : LEMMA
$(x/\text{n0z}) + (y/\text{n0z}) = (x + y)/\text{n0z}$

div_distributes_minus : LEMMA
$(x/\text{n0z}) - (y/\text{n0z}) = (x - y)/\text{n0z}$

div_div1 : LEMMA

$(x/(\text{n0y}/\text{n0z})) \ = \ ((x \times \text{n0z})/\text{n0y})$

div_div2 : LEMMA
$((x/\text{n0y})/\text{n0z}) \ = \ (x/(\text{n0y} \times \text{n0z}))$

idem_add_is_zero : LEMMA $x + x \ = \ x$ IMPLIES $x \ = \ 0$

zero_times1 : LEMMA $0 \times x \ = \ 0$

zero_times2 : LEMMA $x \times 0 \ = \ 0$

zero_times3 : LEMMA $x \times y \ = \ 0$ IFF $x \ = \ 0$ OR $y \ = \ 0$

neg_times_neg : LEMMA $(-x) \times (-y) \ = \ x \times y$

zero_is_neg_zero : LEMMA $-0 \ = \ 0$

strict_lt : LEMMA strict_total_order?(<)

trich_lt : LEMMA $x \ < \ y$ OR $x \ = \ y$ OR $y \ < \ x$

tri_unique_lt1 : LEMMA $x \ < \ y$ IMPLIES ($x \ \neq \ y$ AND NOT ($y \ < \ x$))

tri_unique_lt2 : LEMMA
$x \ = \ y$ IMPLIES (NOT ($x \ < \ y$) AND NOT ($y \ < \ x$))

zero_not_lt_zero : LEMMA NOT $0 \ < \ 0$

neg_lt : LEMMA $0 \ < \ -x$ IFF $x \ < \ 0$

pos_times_lt : LEMMA
$0 \ < \ x \times y$ IFF
($0 \ < \ x$ AND $0 \ < \ y$) OR ($x \ < \ 0$ AND $y \ < \ 0$)

neg_times_lt : LEMMA
$x \times y \ < \ 0$ IFF
($0 \ < \ x$ AND $y \ < \ 0$) OR ($x \ < \ 0$ AND $0 \ < \ y$)

quotient_pos_lt : FORMULA $0 \ < \ 1/\text{n0x}$ IFF $0 \ < \ \text{n0x}$

quotient_neg_lt : FORMULA $1/\text{n0x} \ < \ 0$ IFF $\text{n0x} \ < \ 0$

pos_div_lt : LEMMA
$0 < x/\text{n0y}$ IFF
$(0 < x$ AND $0 < \text{n0y})$ OR $(x < 0$ AND $\text{n0y} < 0)$

neg_div_lt : LEMMA
$x/\text{n0y} < 0$ IFF
$(0 < x$ AND $\text{n0y} < 0)$ OR $(x < 0$ AND $0 < \text{n0y})$

div_mult_pos_lt1 : LEMMA
$z/\text{py} < x$ IFF $z < x \times \text{py}$

div_mult_pos_lt2 : LEMMA
$x < z/\text{py}$ IFF $x \times \text{py} < z$

div_mult_neg_lt1 : LEMMA
$z/\text{ny} < x$ IFF $x \times \text{ny} < z$

div_mult_neg_lt2 : LEMMA
$x < z/\text{ny}$ IFF $z < x \times \text{ny}$

both_sides_plus_lt1 : LEMMA
$x + z < y + z$ IFF $x < y$

both_sides_plus_lt2 : LEMMA
$z + x < z + y$ IFF $x < y$

both_sides_minus_lt1 : LEMMA
$x - z < y - z$ IFF $x < y$

both_sides_minus_lt2 : LEMMA
$z - x < z - y$ IFF $y < x$

both_sides_times_pos_lt1 : LEMMA
$x \times \text{pz} < y \times \text{pz}$ IFF $x < y$

both_sides_times_pos_lt2 : LEMMA
$\text{pz} \times x < \text{pz} \times y$ IFF $x < y$

both_sides_times_neg_lt1 : LEMMA
$x \times \text{nz} < y \times \text{nz}$ IFF $y < x$

both_sides_times_neg_lt2 : LEMMA

$\text{nz} \times x < \text{nz} \times y$ IFF $y < x$

both_sides_div_pos_lt1 : LEMMA
$x/\text{pz} < y/\text{pz}$ IFF $x < y$

both_sides_div_pos_lt2 : LEMMA
$\text{pz}/\text{px} < \text{pz}/\text{py}$ IFF $\text{py} < \text{px}$

both_sides_div_pos_lt3 : LEMMA
$\text{nz}/\text{px} < \text{nz}/\text{py}$ IFF $\text{px} < \text{py}$

both_sides_div_neg_lt1 : LEMMA
$x/\text{nz} < y/\text{nz}$ IFF $y < x$

both_sides_div_neg_lt2 : LEMMA
$\text{pz}/\text{nx} < \text{pz}/\text{ny}$ IFF $\text{ny} < \text{nx}$

both_sides_div_neg_lt3 : LEMMA
$\text{nz}/\text{nx} < \text{nz}/\text{ny}$ IFF $\text{nx} < \text{ny}$

lt_plus_lt1 : LEMMA
$x \leq y$ AND $z < w$ IMPLIES $x + z < y + w$

lt_plus_lt2 : LEMMA
$x < y$ AND $z \leq w$ IMPLIES $x + z < y + w$

lt_minus_lt1 : LEMMA
$x \leq y$ AND $w < z$ IMPLIES $x - z < y - w$

lt_minus_lt2 : LEMMA
$x < y$ AND $w \leq z$ IMPLIES $x - z < y - w$

lt_times_lt_pos1 : LEMMA
$\text{px} \leq y$ AND $\text{nnz} < w$ IMPLIES $\text{px} \times \text{nnz} < y \times w$

lt_times_lt_pos2 : LEMMA
$\text{nnx} < y$ AND $\text{pz} \leq w$ IMPLIES $\text{nnx} \times \text{pz} < y \times w$

lt_div_lt_pos1 : LEMMA
$\text{px} \leq y$ AND $\text{pz} < w$ IMPLIES $\text{px}/w < y/\text{pz}$

lt_div_lt_pos2 : LEMMA

nnx $<$ y AND pz $\leq$ w IMPLIES nnx$/w$ $<$ $y/$pz

lt_times_lt_neg1 : LEMMA
x $\leq$ ny AND z $<$ npw IMPLIES ny $\times$ npw $<$ $x \times z$

lt_times_lt_neg2 : LEMMA
x $<$ npy AND z $\leq$ nw IMPLIES npy $\times$ nw $<$ $x \times z$

lt_div_lt_neg1 : LEMMA
x $\leq$ ny AND z $<$ nw IMPLIES ny$/z$ $<$ $x/$nw

lt_div_lt_neg2 : LEMMA
x $<$ npy AND z $\leq$ nw IMPLIES npy$/z$ $<$ $x/$nw

total_le : LEMMA total_order?($\leq$)

dich_le : LEMMA x $\leq$ y OR y $\leq$ x

zero_le_zero : LEMMA 0 $\leq$ 0

neg_le : LEMMA 0 $\leq$ $-x$ IFF x $\leq$ 0

pos_times_le : LEMMA
0 $\leq$ $x \times y$ IFF
(0 $\leq$ x AND 0 $\leq$ y) OR (x $\leq$ 0 AND y $\leq$ 0)

neg_times_le : LEMMA
$x \times y$ $\leq$ 0 IFF
(0 $\leq$ x AND y $\leq$ 0) OR (x $\leq$ 0 AND 0 $\leq$ y)

quotient_pos_le : FORMULA 0 $\leq$ $1/$n0x IFF 0 $\leq$ n0x

quotient_neg_le : FORMULA $1/$n0x $\leq$ 0 IFF n0x $\leq$ 0

pos_div_le : LEMMA
0 $\leq$ $x/$n0y IFF
(0 $\leq$ x AND 0 $\leq$ n0y) OR (x $\leq$ 0 AND n0y $\leq$ 0)

neg_div_le : LEMMA
$x/$n0y $\leq$ 0 IFF
(0 $\leq$ x AND n0y $\leq$ 0) OR (x $\leq$ 0 AND 0 $\leq$ n0y)

div_mult_pos_le1 : LEMMA
$z/\text{py} \ \le \ x$ IFF $z \ \le \ x \times \text{py}$

div_mult_pos_le2 : LEMMA
$x \ \le \ z/\text{py}$ IFF $x \times \text{py} \ \le \ z$

div_mult_neg_le1 : LEMMA
$z/\text{ny} \ \le \ x$ IFF $x \times \text{ny} \ \le \ z$

div_mult_neg_le2 : LEMMA
$x \ \le \ z/\text{ny}$ IFF $z \ \le \ x \times \text{ny}$

both_sides_plus_le1 : LEMMA
$x + z \ \le \ y + z$ IFF $x \ \le \ y$

both_sides_plus_le2 : LEMMA
$z + x \ \le \ z + y$ IFF $x \ \le \ y$

both_sides_minus_le1 : LEMMA
$x - z \ \le \ y - z$ IFF $x \ \le \ y$

both_sides_minus_le2 : LEMMA
$z - x \ \le \ z - y$ IFF $y \ \le \ x$

both_sides_times_pos_le1 : LEMMA
$x \times \text{pz} \ \le \ y \times \text{pz}$ IFF $x \ \le \ y$

both_sides_times_pos_le2 : LEMMA
$\text{pz} \times x \ \le \ \text{pz} \times y$ IFF $x \ \le \ y$

both_sides_times_neg_le1 : LEMMA
$x \times \text{nz} \ \le \ y \times \text{nz}$ IFF $y \ \le \ x$

both_sides_times_neg_le2 : LEMMA
$\text{nz} \times x \ \le \ \text{nz} \times y$ IFF $y \ \le \ x$

both_sides_div_pos_le1 : LEMMA
$x/\text{pz} \ \le \ y/\text{pz}$ IFF $x \ \le \ y$

both_sides_div_pos_le2 : LEMMA
$\text{pz}/\text{px} \ \le \ \text{pz}/\text{py}$ IFF $\text{py} \ \le \ \text{px}$

both_sides_div_pos_le3 : LEMMA
  nz/px $\leq$ nz/py IFF px $\leq$ py

both_sides_div_neg_le1 : LEMMA
  x/nz $\leq$ y/nz IFF $y \leq x$

both_sides_div_neg_le2 : LEMMA
  pz/nx $\leq$ pz/ny IFF ny $\leq$ nx

both_sides_div_neg_le3 : LEMMA
  nz/nx $\leq$ nz/ny IFF nx $\leq$ ny

le_plus_le : LEMMA
  $x \leq y$ AND $z \leq w$ IMPLIES $x + z \leq y + w$

le_minus_le : LEMMA
  $x \leq y$ AND $w \leq z$ IMPLIES $x - z \leq y - w$

le_times_le_pos : LEMMA
  nnx $\leq$ y AND nnz $\leq$ w IMPLIES nnx $\times$ nnz $\leq$ $y \times w$

le_div_le_pos : LEMMA
  nnx $\leq$ y AND pz $\leq$ w IMPLIES nnx/w $\leq$ y/pz

le_times_le_neg : LEMMA
  $x \leq$ npy AND $z \leq$ npw IMPLIES npy $\times$ npw $\leq$ $x \times z$

le_div_le_neg : LEMMA
  $x \leq$ npy AND $z \leq$ nw IMPLIES npy/z $\leq$ x/nw

strict_gt : LEMMA strict_total_order?(>)

trich_gt : LEMMA $x > y$ OR $x = y$ OR $y > x$

tri_unique_gt1 : LEMMA $x > y$ IMPLIES ($x \neq y$ AND NOT ($y > x$))

tri_unique_gt2 : LEMMA
  $x = y$ IMPLIES (NOT ($x > y$) AND NOT ($y > x$))

zero_not_gt_zero : LEMMA NOT $0 > 0$

neg_gt : LEMMA $0 > -x$ IFF $x > 0$

pos_times_gt : LEMMA
$x \times y > 0$ IFF
$(0 > x$ AND $0 > y)$ OR $(x > 0$ AND $y > 0)$

neg_times_gt : LEMMA
$0 > x \times y$ IFF
$(0 > x$ AND $y > 0)$ OR $(x > 0$ AND $0 > y)$

quotient_pos_gt : FORMULA $1/\text{n0x} > 0$ IFF $\text{n0x} > 0$

quotient_neg_gt : FORMULA $0 > 1/\text{n0x}$ IFF $0 > \text{n0x}$

pos_div_gt : LEMMA
$x/\text{n0y} > 0$ IFF
$(0 > x$ AND $0 > \text{n0y})$ OR $(x > 0$ AND $\text{n0y} > 0)$

neg_div_gt : LEMMA
$0 > x/\text{n0y}$ IFF
$(0 > x$ AND $\text{n0y} > 0)$ OR $(x > 0$ AND $0 > \text{n0y})$

both_sides_plus_gt1 : LEMMA
$x + z > y + z$ IFF $x > y$

both_sides_plus_gt2 : LEMMA
$z + x > z + y$ IFF $x > y$

both_sides_minus_gt1 : LEMMA
$x - z > y - z$ IFF $x > y$

both_sides_minus_gt2 : LEMMA
$z - x > z - y$ IFF $y > x$

both_sides_times_pos_gt1 : LEMMA
$x \times \text{pz} > y \times \text{pz}$ IFF $x > y$

both_sides_times_pos_gt2 : LEMMA
$\text{pz} \times x > \text{pz} \times y$ IFF $x > y$

both_sides_times_neg_gt1 : LEMMA
$x \times \text{nz} > y \times \text{nz}$ IFF $y > x$

both_sides_times_neg_gt2 : LEMMA
nz $\times x > $ nz $\times y$ IFF $y > x$

both_sides_div_pos_gt1 : LEMMA
$x/$pz $>$ $y/$pz IFF $x > y$

both_sides_div_pos_gt2 : LEMMA
pz$/$px $>$ pz$/$py IFF py $>$ px

both_sides_div_pos_gt3 : LEMMA
nz$/$px $>$ nz$/$py IFF px $>$ py

both_sides_div_neg_gt1 : LEMMA
$x/$nz $>$ $y/$nz IFF $y > x$

both_sides_div_neg_gt2 : LEMMA
pz$/$nx $>$ pz$/$ny IFF ny $>$ nx

both_sides_div_neg_gt3 : LEMMA
nz$/$nx $>$ nz$/$ny IFF nx $>$ ny

gt_plus_gt1 : LEMMA
$x \geq y$ AND $z > w$ IMPLIES $x + z > y + w$

gt_plus_gt2 : LEMMA
$x > y$ AND $z \geq w$ IMPLIES $x + z > y + w$

gt_minus_gt1 : LEMMA
$x \geq y$ AND $w > z$ IMPLIES $x - z > y - w$

gt_minus_gt2 : LEMMA
$x > y$ AND $w \geq z$ IMPLIES $x - z > y - w$

gt_times_gt_pos1 : LEMMA
$x \geq$ py AND $z >$ nnw IMPLIES $x \times z >$ py $\times$ nnw

gt_times_gt_pos2 : LEMMA
$x >$ nny AND $z \geq$ pw IMPLIES $x \times z >$ nny $\times$ pw

gt_div_gt_pos1 : LEMMA
$x \geq$ py AND $z >$ pw IMPLIES $x/$pw $>$ py$/z$

gt_div_gt_pos2 : LEMMA
$x > \text{nny}$ AND $z \geq \text{pw}$ IMPLIES $x/\text{pw} > \text{nny}/z$

gt_times_gt_neg1 : LEMMA
$\text{nx} \geq y$ AND $\text{npz} > w$ IMPLIES $y \times w > \text{nx} \times \text{npz}$

gt_times_gt_neg2 : LEMMA
$\text{npx} > y$ AND $\text{nz} \geq w$ IMPLIES $y \times w > \text{npx} \times \text{nz}$

gt_div_gt_neg1 : LEMMA
$\text{nx} \geq y$ AND $\text{nz} > w$ IMPLIES $y/\text{nz} > \text{nx}/w$

gt_div_gt_neg2 : LEMMA
$\text{npx} > y$ AND $\text{nz} \geq w$ IMPLIES $y/\text{nz} > \text{npx}/w$

total_ge : LEMMA total_order?($\geq$)

dich_ge : LEMMA $x \geq y$ OR $y \geq x$

zero_ge_zero : LEMMA $0 \geq 0$

neg_ge : LEMMA $0 \geq -x$ IFF $x \geq 0$

pos_times_ge : LEMMA
$x \times y \geq 0$ IFF
$(0 \geq x$ AND $0 \geq y)$ OR $(x \geq 0$ AND $y \geq 0)$

neg_times_ge : LEMMA
$0 \geq x \times y$ IFF
$(0 \geq x$ AND $y \geq 0)$ OR $(x \geq 0$ AND $0 \geq y)$

quotient_pos_ge : FORMULA $1/\text{n0x} \geq 0$ IFF $\text{n0x} \geq 0$

quotient_neg_ge : FORMULA $0 \geq 1/\text{n0x}$ IFF $0 \geq \text{n0x}$

pos_div_ge : LEMMA
$x/\text{n0y} \geq 0$ IFF
$(0 \geq x$ AND $0 \geq \text{n0y})$ OR $(x \geq 0$ AND $\text{n0y} \geq 0)$

neg_div_ge : LEMMA
$0 \geq x/\text{n0y}$ IFF
$(0 \geq x$ AND $\text{n0y} \geq 0)$ OR $(x \geq 0$ AND $0 \geq \text{n0y})$

div_mult_pos_ge1 : LEMMA
$z/\text{py} \ \geq \ x$ IFF $z \ \geq \ x \times \text{py}$

div_mult_pos_ge2 : LEMMA
$x \ \geq \ z/\text{py}$ IFF $x \times \text{py} \ \geq \ z$

div_mult_neg_ge1 : LEMMA
$z/\text{ny} \ \geq \ x$ IFF $x \times \text{ny} \ \geq \ z$

div_mult_neg_ge2 : LEMMA
$x \ \geq \ z/\text{ny}$ IFF $z \ \geq \ x \times \text{ny}$

both_sides_plus_ge1 : LEMMA
$x + z \ \geq \ y + z$ IFF $x \ \geq \ y$

both_sides_plus_ge2 : LEMMA
$z + x \ \geq \ z + y$ IFF $x \ \geq \ y$

both_sides_minus_ge1 : LEMMA
$x - z \ \geq \ y - z$ IFF $x \ \geq \ y$

both_sides_minus_ge2 : LEMMA
$z - x \ \geq \ z - y$ IFF $y \ \geq \ x$

both_sides_times_pos_ge1 : LEMMA
$x \times \text{pz} \ \geq \ y \times \text{pz}$ IFF $x \ \geq \ y$

both_sides_times_pos_ge2 : LEMMA
$\text{pz} \times x \ \geq \ \text{pz} \times y$ IFF $x \ \geq \ y$

both_sides_times_neg_ge1 : LEMMA
$x \times \text{nz} \ \geq \ y \times \text{nz}$ IFF $y \ \geq \ x$

both_sides_times_neg_ge2 : LEMMA
$\text{nz} \times x \ \geq \ \text{nz} \times y$ IFF $y \ \geq \ x$

both_sides_div_pos_ge1 : LEMMA
$x/\text{pz} \ \geq \ y/\text{pz}$ IFF $x \ \geq \ y$

both_sides_div_pos_ge2 : LEMMA
$\text{pz}/\text{px} \ \geq \ \text{pz}/\text{py}$ IFF $\text{py} \ \geq \ \text{px}$

both_sides_div_pos_ge3: LEMMA
nz/px $\geq$ nz/py IFF px $\geq$ py

both_sides_div_neg_ge1: LEMMA
x/nz $\geq$ y/nz IFF $y \geq x$

both_sides_div_neg_ge2: LEMMA
pz/nx $\geq$ pz/ny IFF ny $\geq$ nx

both_sides_div_neg_ge3: LEMMA
nz/nx $\geq$ nz/ny IFF nx $\geq$ ny

ge_plus_ge: LEMMA
$x \geq y$ AND $z \geq w$ IMPLIES $x + z \geq y + w$

ge_minus_ge: LEMMA
$x \geq y$ AND $w \geq z$ IMPLIES $x - z \geq y - w$

ge_times_ge_pos: LEMMA
$x \geq$ nny AND $z \geq$ nnw IMPLIES $x \times z \geq$ nny $\times$ nnw

ge_div_ge_pos: LEMMA
$x \geq$ nny AND $z \geq$ pw IMPLIES x/pw $\geq$ nny/z

ge_times_ge_neg: LEMMA
npx $\geq y$ AND npz $\geq w$ IMPLIES $y \times w \geq$ npx $\times$ npz

ge_div_ge_neg: LEMMA
npx $\geq y$ AND nz $\geq w$ IMPLIES y/nz $\geq$ npx/w

nonzero_times1: LEMMA n0x $\times\, y = 0$ IFF $y = 0$

nonzero_times2: LEMMA $x \times$ n0y $= 0$ IFF $x = 0$

nonzero_times3: LEMMA n0x $\times$ n0y $= 0$ IFF FALSE

eq1_gt: FORMULA $x > 1$ AND $x \times y = 1$ IMPLIES $y < 1$

eq1_ge: FORMULA $x \geq 1$ AND $x \times y = 1$ IMPLIES $y \leq 1$

eqm1_gt: FORMULA

$x > 1$ AND $x \times y = -1$ IMPLIES $y > -1$

eqm1_ge: FORMULA
$x \geq 1$ AND $x \times y = -1$ IMPLIES $y \geq -1$

eqm1_lt: FORMULA
$x < -1$ AND $x \times y = -1$ IMPLIES $y < 1$

eqm1_le: FORMULA
$x \leq -1$ AND $x \times y = -1$ IMPLIES $y \leq 1$

sqrt_1: LEMMA $x \times x = 1$ IFF $x = 1$ OR $x = -1$

sqrt_1_lt: LEMMA $x \times x < 1$ IMPLIES $|x| < 1$

sqrt_1_le: LEMMA $x \times x \leq 1$ IMPLIES $|x| \leq 1$

idem_mult: LEMMA $x \times x = x$ IFF $x = 0$ OR $x = 1$

i, j: VAR $\mathbb{Z}$

product_1: LEMMA
$i \geq 0$ AND $j \geq 0$ AND $i \times j = 1$ IMPLIES
$i = 1$ AND $j = 1$

product_m1: LEMMA
$i \geq 0$ AND $j \leq 0$ AND $i \times j = -1$ IMPLIES
$i = 1$ AND $j = -1$

triangle: LEMMA $|x + y| \leq |x| + |y|$

abs_mult: LEMMA $|x \times y| = |x| \times |y|$

abs_div: LEMMA $|x/\text{n0y}| = |x|/|\text{n0y}|$

abs_abs: LEMMA $||x|| = |x|$

abs_square: LEMMA $|x \times x| = x \times x$

abs_limits: LEMMA
$-(|x| + |y|) \leq x + y$ AND
$x + y \leq |x| + |y|$

axiom_of_archimedes: LEMMA $\forall\ (x\colon\ \mathbb{R})\colon\ \exists\ (i\colon\ \mathbb{Z})\colon\ x\ <\ i$

archimedean: LEMMA
$\forall$ (px: $\mathbb{R}_{>0}$): $\exists\ (n\colon\ \mathbb{N}_{>0})\colon\ 1/n\ <$ px

real_lt_is_strict_total_order: JUDGEMENT $<$ HAS_TYPE
(strict_total_order?$[\mathbb{R}]$)

real_le_is_total_order: JUDGEMENT $\leq$ HAS_TYPE (total_order?$[\mathbb{R}]$)

real_gt_is_strict_total_order: JUDGEMENT $>$ HAS_TYPE
(strict_total_order?$[\mathbb{R}]$)

real_ge_is_total_order: JUDGEMENT $\geq$ HAS_TYPE (total_order?$[\mathbb{R}]$)

END real_props

extra_real_props: THEORY
BEGIN

w, x, y, z: VAR $\mathbb{R}$

n0w, n0x, n0y, n0z: VAR $\mathbb{R}_{\neq 0}$

nnw, nnx, nny, nnz: VAR $\mathbb{R}_{\geq 0}$

pw, px, py, pz: VAR $\mathbb{R}_{>0}$

npw, npx, npy, npz: VAR $\mathbb{R}_{\leq 0}$

nw, nx, ny, nz: VAR $\mathbb{R}_{<0}$

pos_neg_split: LEMMA $\exists$ px, nx: n0x = px OR n0x = nx

div_mult_pos_neg_lt1: LEMMA
$z/\text{n0y} < x$ IFF
IF n0y > 0
THEN $z < x \times \text{n0y}$
ELSE $x \times \text{n0y} < z$
ENDIF

div_mult_pos_neg_lt2: LEMMA
$x < z/\text{n0y}$ IFF
IF n0y > 0
THEN $x \times \text{n0y} < z$
ELSE $z < x \times \text{n0y}$
ENDIF

div_mult_pos_neg_le1: LEMMA
$z/\text{n0y} \leq x$ IFF
IF n0y > 0
THEN $z \leq x \times \text{n0y}$
ELSE $x \times \text{n0y} \leq z$
ENDIF

div_mult_pos_neg_le2: LEMMA
$x \leq z/\text{n0y}$ IFF
IF n0y > 0
THEN $x \times \text{n0y} \leq z$

ELSE $z \leq x \times$ n0y
ENDIF

div_mult_pos_neg_gt1 : LEMMA
$z/$n0y $> x$ IFF
IF n0y > 0
THEN $z > x \times$ n0y
ELSE $x \times$ n0y $> z$
ENDIF

div_mult_pos_neg_gt2 : LEMMA
$x > z/$n0y IFF
IF n0y > 0
THEN $x \times$ n0y $> z$
ELSE $z > x \times$ n0y
ENDIF

div_mult_pos_neg_ge1 : LEMMA
$z/$n0y $\geq x$ IFF
IF n0y > 0
THEN $z \geq x \times$ n0y
ELSE $x \times$ n0y $\geq z$
ENDIF

div_mult_pos_neg_ge2 : LEMMA
$x \geq z/$n0y IFF
IF n0y > 0
THEN $x \times$ n0y $\geq z$
ELSE $z \geq x \times$ n0y
ENDIF

both_sides_times_pos_neg_lt1 : LEMMA
IF n0z > 0
THEN $x \times$ n0z $< y \times$ n0z
ELSE $y \times$ n0z $< x \times$ n0z
ENDIF
IFF $x < y$

both_sides_times_pos_neg_lt2 : LEMMA
IF n0z > 0
THEN n0z $\times x <$ n0z $\times y$
ELSE n0z $\times y <$ n0z $\times x$

ENDIF
  IFF $x < y$

both_sides_times_pos_neg_le1 : LEMMA
  IF n0z > 0
    THEN $x \times$ n0z $\leq y \times$ n0z
  ELSE $y \times$ n0z $\leq x \times$ n0z
  ENDIF
  IFF $x \leq y$

both_sides_times_pos_neg_le2 : LEMMA
  IF n0z > 0
    THEN n0z $\times x \leq$ n0z $\times y$
  ELSE n0z $\times y \leq$ n0z $\times x$
  ENDIF
  IFF $x \leq y$

both_sides_times_pos_neg_gt1 : LEMMA
  IF n0z > 0
    THEN $x \times$ n0z $> y \times$ n0z
  ELSE $y \times$ n0z $> x \times$ n0z
  ENDIF
  IFF $x > y$

both_sides_times_pos_neg_gt2 : LEMMA
  IF n0z > 0
    THEN n0z $\times x >$ n0z $\times y$
  ELSE n0z $\times y >$ n0z $\times x$
  ENDIF
  IFF $x > y$

both_sides_times_pos_neg_ge1 : LEMMA
  IF n0z > 0
    THEN $x \times$ n0z $\geq y \times$ n0z
  ELSE $y \times$ n0z $\geq x \times$ n0z
  ENDIF
  IFF $x \geq y$

both_sides_times_pos_neg_ge2 : LEMMA
  IF n0z > 0
    THEN n0z $\times x \geq$ n0z $\times y$
  ELSE n0z $\times y \geq$ n0z $\times x$

ENDIF
IFF $x \geq y$

both_sides_div_pos_neg_lt1 : LEMMA
IF n0z > 0
THEN $x/$n0z $<$ $y/$n0z
ELSE $y/$n0z $<$ $x/$n0z
ENDIF
IFF $x < y$

both_sides_div_pos_neg_lt2 : LEMMA
IF n0z > 0
THEN n0z/px $<$ n0z/py
ELSE n0z/py $<$ n0z/px
ENDIF
IFF py $<$ px

both_sides_div_pos_neg_lt3 : LEMMA
IF n0z > 0
THEN n0z/nx $<$ n0z/ny
ELSE n0z/ny $<$ n0z/nx
ENDIF
IFF ny $<$ nx

both_sides_div_pos_neg_le1 : LEMMA
IF n0z > 0
THEN $x/$n0z $\leq$ $y/$n0z
ELSE $y/$n0z $\leq$ $x/$n0z
ENDIF
IFF $x \leq y$

both_sides_div_pos_neg_le2 : LEMMA
IF n0z > 0
THEN n0z/px $\leq$ n0z/py
ELSE n0z/py $\leq$ n0z/px
ENDIF
IFF py $\leq$ px

both_sides_div_pos_neg_le3 : LEMMA
IF n0z > 0
THEN n0z/nx $\leq$ n0z/ny
ELSE n0z/ny $\leq$ n0z/nx

ENDIF
  IFF ny $\leq$ nx

both_sides_div_pos_neg_gt1 : LEMMA
  IF n0z $>$ 0
    THEN $x/$n0z $>$ $y/$n0z
  ELSE $y/$n0z $>$ $x/$n0z
  ENDIF
  IFF $x > y$

both_sides_div_pos_neg_gt2 : LEMMA
  IF n0z $>$ 0
    THEN n0z/px $>$ n0z/py
  ELSE n0z/py $>$ n0z/px
  ENDIF
  IFF py $>$ px

both_sides_div_pos_neg_gt3 : LEMMA
  IF n0z $>$ 0
    THEN n0z/nx $>$ n0z/ny
  ELSE n0z/ny $>$ n0z/nx
  ENDIF
  IFF ny $>$ nx

both_sides_div_pos_neg_ge1 : LEMMA
  IF n0z $>$ 0
    THEN $x/$n0z $\geq$ $y/$n0z
  ELSE $y/$n0z $\geq$ $x/$n0z
  ENDIF
  IFF $x \geq y$

both_sides_div_pos_neg_ge2 : LEMMA
  IF n0z $>$ 0
    THEN n0z/px $\geq$ n0z/py
  ELSE n0z/py $\geq$ n0z/px
  ENDIF
  IFF py $\geq$ px

both_sides_div_pos_neg_ge3 : LEMMA
  IF n0z $>$ 0
    THEN n0z/nx $\geq$ n0z/ny
  ELSE n0z/ny $\geq$ n0z/nx

ENDIF
IFF ny $\geq$ nx

both_sides_times1_imp: LEMMA
$x = y$ IMPLIES $x \times w = y \times w$

both_sides_times2_imp: LEMMA
$x = y$ IMPLIES $w \times x = w \times y$

both_sides_times_pos_le1_imp: LEMMA
$x \leq y$ IMPLIES $x \times$ nnw $\leq y \times$ nnw

both_sides_times_pos_le2_imp: LEMMA
$x \leq y$ IMPLIES nnw $\times x \leq$ nnw $\times y$

both_sides_times_neg_le1_imp: LEMMA
$y \leq x$ IMPLIES $x \times$ npw $\leq y \times$ npw

both_sides_times_neg_le2_imp: LEMMA
$y \leq x$ IMPLIES npw $\times x \leq$ npw $\times y$

both_sides_times_pos_ge1_imp: LEMMA
$x \geq y$ IMPLIES $x \times$ nnw $\geq y \times$ nnw

both_sides_times_pos_ge2_imp: LEMMA
$x \geq y$ IMPLIES nnw $\times x \geq$ nnw $\times y$

both_sides_times_neg_ge1_imp: LEMMA
$y \geq x$ IMPLIES $x \times$ npw $\geq y \times$ npw

both_sides_times_neg_ge2_imp: LEMMA
$y \geq x$ IMPLIES npw $\times x \geq$ npw $\times y$

both_sides_times_pos_neg_le1_imp: LEMMA
$x \leq y$ IMPLIES
IF $w \geq 0$
THEN $x \times w \leq y \times w$
ELSE $y \times w \leq x \times w$
ENDIF

both_sides_times_pos_neg_le2_imp: LEMMA
$x \leq y$ IMPLIES

IF $w \geq 0$
THEN $w \times x \leq w \times y$
ELSE $w \times y \leq w \times x$
ENDIF

both_sides_times_pos_neg_ge1_imp : LEMMA
$x \geq y$ IMPLIES
IF $w \geq 0$
THEN $x \times w \geq y \times w$
ELSE $y \times w \geq x \times w$
ENDIF

both_sides_times_pos_neg_ge2_imp : LEMMA
$x \geq y$ IMPLIES
IF $w \geq 0$
THEN $w \times x \geq w \times y$
ELSE $w \times y \geq w \times x$
ENDIF

zero_times4 : LEMMA $0 = x \times y$ IFF $x = 0$ OR $y = 0$

times_div_cancel1 : LEMMA $(\text{n0z} \times x)/\text{n0z} = x$

times_div_cancel2 : LEMMA $(x \times \text{n0z})/\text{n0z} = x$

div_mult_pos_gt1 : LEMMA $z/\text{py} > x$ IFF $z > x \times \text{py}$

div_mult_pos_gt2 : LEMMA $x > z/\text{py}$ IFF $x \times \text{py} > z$

div_mult_neg_gt1 : LEMMA $z/\text{ny} > x$ IFF $x \times \text{ny} > z$

div_mult_neg_gt2 : LEMMA $x > z/\text{ny}$ IFF $z > x \times \text{ny}$

lt_cut : LEMMA $x < y$ AND $y < z$ IMPLIES $x < z$

le_cut : LEMMA $x \leq y$ AND $y \leq z$ IMPLIES $x \leq z$

gt_cut : LEMMA $x > y$ AND $y > z$ IMPLIES $x > z$

ge_cut : LEMMA $x \geq y$ AND $y \geq z$ IMPLIES $x \geq z$

le_times_le_any1 : LEMMA

```
IF w ≥ 0 IFF x ≥ 0
   THEN IF y ≥ 0 IFF z ≥ 0
           THEN |w| ≤ |y| AND |x| ≤ |z|
        ELSE (w = 0 OR x = 0) AND (y = 0 OR z = 0)
        ENDIF
ELSE IF y ≥ 0 IFF z ≥ 0
        THEN TRUE
     ELSE |w| ≥ |y| AND |x| ≥ |z|
     ENDIF
ENDIF
  IMPLIES w × x ≤ y × z
```

ge_times_ge_any1 : LEMMA

```
IF y ≥ 0 IFF z ≥ 0
   THEN IF w ≥ 0 IFF x ≥ 0
           THEN |w| ≥ |y| AND |x| ≥ |z|
        ELSE (w = 0 OR x = 0) AND (y = 0 OR z = 0)
        ENDIF
ELSE IF w ≥ 0 IFF x ≥ 0
        THEN TRUE
     ELSE |w| ≤ |y| AND |x| ≤ |z|
     ENDIF
ENDIF
  IMPLIES w × x ≥ y × z
```

lt_times_lt_any1 : LEMMA

```
IF w = 0 OR x = 0
   THEN 0 < y AND 0 < z OR y < 0 AND z < 0
ELSIF w > 0 IFF x > 0
   THEN (y > 0 IFF z > 0) AND
          (|w| ≤ |y| AND |x| < |z| OR
              |w| < |y| AND |x| ≤ |z|)
ELSIF y > 0 IFF z > 0 THEN TRUE
ELSE |w| ≥ |y| AND |x| > |z| OR
       |w| > |y| AND |x| ≥ |z|
ENDIF
  IMPLIES w × x < y × z
```

gt_times_gt_any1 : LEMMA

```
IF y = 0 OR z = 0
   THEN 0 < w AND 0 < x OR w < 0 AND x < 0
ELSIF y > 0 IFF z > 0
```

THEN $(w > 0$ IFF $x > 0)$ AND
$(|w| \geq |y|$ AND $|x| > |z|$ OR
$|w| > |y|$ AND $|x| \geq |z|)$
ELSIF $w > 0$ IFF $x > 0$ THEN TRUE
ELSE $|w| \leq |y|$ AND $|x| < |z|$ OR
$|w| < |y|$ AND $|x| \leq |z|$
ENDIF
IMPLIES $w \times x > y \times z$

le_times_le_any2 : LEMMA
$w \times x \leq y \times z$ IMPLIES
IF $y = 0$ OR $z = 0$
THEN $(0 \geq w$ OR $0 \geq x)$ AND $(w \geq 0$ OR $x \geq 0)$
ELSIF $y > 0$ IFF $z > 0$
THEN $(w > 0$ IFF $x > 0)$ IMPLIES
$(|w| < |y|$ OR $|x| \leq |z|)$ AND
$(|w| \leq |y|$ OR $|x| < |z|)$
ELSE NOT $(w > 0$ IFF $x > 0)$ AND
$(|w| > |y|$ OR $|x| \geq |z|)$ AND
$(|w| \geq |y|$ OR $|x| > |z|)$
ENDIF

ge_times_ge_any2 : LEMMA
$w \times x \geq y \times z$ IMPLIES
IF $w = 0$ OR $x = 0$
THEN $(0 \geq y$ OR $0 \geq z)$ AND $(y \geq 0$ OR $z \geq 0)$
ELSIF $w > 0$ IFF $x > 0$
THEN $(y > 0$ IFF $z > 0)$ IMPLIES
$(|w| > |y|$ OR $|x| \geq |z|)$ AND
$(|w| \geq |y|$ OR $|x| > |z|)$
ELSE NOT $(y > 0$ IFF $z > 0)$ AND
$(|w| < |y|$ OR $|x| \leq |z|)$ AND
$(|w| \leq |y|$ OR $|x| < |z|)$
ENDIF

lt_times_lt_any2 : LEMMA
$w \times x < y \times z$ IMPLIES
IF $y \geq 0$ IFF $z \geq 0$
THEN IF $w \geq 0$ IFF $x \geq 0$
THEN $|w| < |y|$ OR $|x| < |z|$
ELSE $(w \neq 0$ AND $x \neq 0)$ OR $(y \neq 0$ AND $z \neq 0)$
ENDIF

ELSE NOT ($w \geq 0$ IFF $x \geq 0$) AND
($|w| > |y|$ OR $|x| > |z|$)
ENDIF

gt_times_gt_any2: LEMMA
$w \times x > y \times z$ IMPLIES
IF $w \geq 0$ IFF $x \geq 0$
THEN IF $y \geq 0$ IFF $z \geq 0$
THEN $|w| > |y|$ OR $|x| > |z|$
ELSE ($w \neq 0$ AND $x \neq 0$) OR ($y \neq 0$ AND $z \neq 0$)
ENDIF
ELSE NOT ($y \geq 0$ IFF $z \geq 0$) AND
($|w| < |y|$ OR $|x| < |z|$)
ENDIF

END extra_real_props

extra_tegies: THEORY
BEGIN

x, y: VAR $\mathbb{R}$

n0z: VAR $\mathbb{R}_{\neq 0}$

neg_mult: LEMMA $-x \times y = -(x \times y)$

mult_neg: LEMMA $x \times -y = -(x \times y)$

neg_add: LEMMA $-x + y = y - x$

add_neg: LEMMA $x + -y = x - y$

neg_div: LEMMA $-x/\text{n0z} = -(x/\text{n0z})$

div_neg: LEMMA $x/-\text{n0z} = -(x/\text{n0z})$

one_times: LEMMA $1 \times x = x$

neg_one_times: LEMMA $-1 \times x = -x$

zero_div: LEMMA $0/\text{n0z} = 0$

neg_neg: LEMMA $--x = x$

END extra_tegies

rational_props : THEORY
BEGIN

x, y : VAR $\mathbb{R}$

i : VAR $\mathbb{Z}$

n0j : VAR $\mathbb{Z}_{\neq 0}$

p : VAR $\mathbb{N}_{>0}$

r : VAR $\mathbb{Q}$

rational_pred_ax : AXIOM $\exists\ i,$ n0j : $r = i/$n0j

rational_pred_ax2 : LEMMA $\exists\ i,\ p : r = i/p$

density_positive : LEMMA
$0 \leq x$ AND $x < y$ IMPLIES ($\exists\ r : x < r$ AND $r < y$)

density : LEMMA $x < y$ IMPLIES ($\exists\ r : x < r$ AND $r < y$)

END rational_props

integer_props : THEORY
BEGIN

m, n : VAR $\mathbb{N}$

i, j, k : VAR $\mathbb{Z}$

n0j : VAR $\mathbb{Z}_{\neq 0}$

N : VAR (nonempty?[$\mathbb{N}$])

I : VAR (nonempty?[$\mathbb{Z}$])

integer_pred_ax : LEMMA $\exists\ n : i = n$ OR $i = -n$

div_simple : LEMMA
$(\exists\ k : i = k \times \text{n0j}) = \text{integer\_pred}(i/\text{n0j})$

lub_nat : LEMMA
upper_bound?(m, extend[$\mathbb{R}$, $\mathbb{N}$, bool, FALSE](N)) $\Rightarrow$
$\exists$ (n : (N)) :
least_upper_bound?(n, extend[$\mathbb{R}$, $\mathbb{N}$, bool, FALSE](N))

lub_int : LEMMA
upper_bound?(i, extend[$\mathbb{R}$, $\mathbb{Z}$, bool, FALSE](I)) $\Rightarrow$
$\exists$ (j : (I)) :
least_upper_bound?(j, extend[$\mathbb{R}$, $\mathbb{Z}$, bool, FALSE](I))

glb_nat : LEMMA
$\exists$ (n : (N)) :
greatest_lower_bound?(n, extend[$\mathbb{R}$, $\mathbb{N}$, bool, FALSE](N))

glb_int : LEMMA
lower_bound?(i, extend[$\mathbb{R}$, $\mathbb{Z}$, bool, FALSE](I)) $\Rightarrow$
$\exists$ (j : (I)) :
greatest_lower_bound?(j, extend[$\mathbb{R}$, $\mathbb{Z}$, bool, FALSE](I))

END integer_props

floor_ceil: THEORY
BEGIN

x, y: VAR $\mathbb{R}$

py: VAR $\mathbb{R}_{>0}$

j: VAR nonzero_integer

i, k: VAR $\mathbb{Z}$

floor_exists: LEMMA $\exists\ i:\ i \leq x\ \&\ x < i+1$

ceiling_exists: LEMMA $\exists\ i:\ x \leq i\ \&\ i < x+1$

$\lfloor x \rfloor$: $\{i \mid i \leq x\ \&\ x < i+1\}$

fractional(x): $\{x \mid 0 \leq x\ \&\ x < 1\}$ = $x - \lfloor x \rfloor$

$\lceil x \rceil$: $\{i \mid x \leq i\ \&\ i < x+1\}$

floor_def: LEMMA $\lfloor x \rfloor \leq x\ \&\ x < \lfloor x \rfloor + 1$

ceiling_def: LEMMA $x \leq \lceil x \rceil\ \&\ \lceil x \rceil < x+1$

floor_ceiling_reflect1: LEMMA $\lfloor -x \rfloor = -\lceil x \rceil$

floor_ceiling_reflect2: LEMMA $\lceil -x \rceil = -\lfloor x \rfloor$

nonneg_floor_is_nat: JUDGEMENT floor(x: $\mathbb{R}_{\geq 0}$) HAS_TYPE $\mathbb{N}$

nonneg_ceiling_is_nat: JUDGEMENT ceiling(x: $\mathbb{R}_{\geq 0}$) HAS_TYPE $\mathbb{N}$

floor_int: LEMMA $\lfloor i \rfloor = i$

ceiling_int: LEMMA $\lceil i \rceil = i$

floor_plus_int: LEMMA $\lfloor x + i \rfloor = \lfloor x \rfloor + i$

ceiling_plus_int: LEMMA $\lceil x + i \rceil = \lceil x \rceil + i$

floor_ceiling_nonint: LEMMA

$\text{NOT integer?}(x) \Rightarrow \lceil x \rceil - \lfloor x \rfloor = 1$

floor_ceiling_int : LEMMA $\lfloor i \rfloor = \lceil i \rceil$

floor_neg : LEMMA
$\lfloor x \rfloor =$
IF integer?(x)
THEN $-\lfloor -x \rfloor$
ELSE $-\lfloor -x \rfloor - 1$
ENDIF

real_parts : LEMMA $x = \lfloor x \rfloor + \text{fractional}(x)$

floor_plus : LEMMA
$\lfloor x + y \rfloor =$
$\lfloor x \rfloor + \lfloor y \rfloor + \lfloor \text{fractional}(x) + \text{fractional}(y) \rfloor$

ceiling_plus : LEMMA
$\lceil x + y \rceil =$
$\lfloor x \rfloor + \lfloor y \rfloor + \lceil \text{fractional}(x) + \text{fractional}(y) \rceil$

floor_split : LEMMA $i = \lfloor i/2 \rfloor + \lceil i/2 \rceil$

floor_within_1 : LEMMA $x - \lfloor x \rfloor < 1$

ceiling_within_1 : LEMMA $\lceil x \rceil - x < 1$

floor_val : LEMMA
$i \geq k \times j$ AND $i < (k+1) \times j$ IMPLIES
$\lfloor i/j \rfloor = k$

floor_small : LEMMA
$|i| < |j|$ IMPLIES
$\lfloor i/j \rfloor =$
IF $i/j \geq 0$ THEN 0 ELSE -1 ENDIF

floor_eq_0 : LEMMA $\lfloor x \rfloor = 0$ IMPLIES $x \geq 0$ AND $x < 1$

fractional_plus : LEMMA
$\text{fractional}(x + y) =$
$\text{fractional}(\text{fractional}(x) + \text{fractional}(y))$

floor_div : LEMMA
  $\lfloor x/\text{py} \rfloor = i$ IFF
   $\text{py} \times i \leq x$ AND $x < \text{py} \times (i+1)$

floor_0 : LEMMA $\lfloor x \rfloor = 0$ IFF $0 \leq x$ AND $x < 1$

END floor_ceil

exponentiation: THEORY
BEGIN

r: VAR $\mathbb{R}$

q: VAR $\mathbb{Q}$

nnq: VAR $\mathbb{Q}_{\geq 0}$

m, n: VAR $\mathbb{N}$

pm, pn: VAR $\mathbb{N}_{>0}$

i, j: VAR $\mathbb{Z}$

n0i, n0j: VAR $\mathbb{Z}_{\neq 0}$

x, y: VAR $\mathbb{R}$

px, py: VAR $\mathbb{R}_{>0}$

n0x, n0y: VAR $\mathbb{R}_{\neq 0}$

gt1x, gt1y: VAR $\{r\colon \mathbb{R}_{>0} \mid r > 1\}$

lt1x, lt1y: VAR $\{r\colon \mathbb{R}_{>0} \mid r < 1\}$

ge1x, ge1y: VAR $\{r\colon \mathbb{R}_{>0} \mid r \geq 1\}$

le1x, le1y: VAR $\{r\colon \mathbb{R}_{>0} \mid r \leq 1\}$

r^n: RECURSIVE $\mathbb{R}$ = IF $n = 0$ THEN 1 ELSE $r \times r^{n-1}$ ENDIF
MEASURE n;

expt_pos_aux: LEMMA $\text{px}^n > 0$

expt_nonzero_aux: LEMMA $\text{n0x}^n \neq 0$;

nnreal_expt: JUDGEMENT expt(x: $\mathbb{R}_{\geq 0}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{R}_{\geq 0}$

posreal_expt: JUDGEMENT expt(x: $\mathbb{R}_{>0}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{R}_{>0}$

nzreal_expt: JUDGEMENT expt(x: $\mathbb{R}_{\neq 0}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{R}_{\neq 0}$

rat_expt: JUDGEMENT expt(x: $\mathbb{Q}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{Q}$

nnrat_expt: JUDGEMENT expt(x: $\mathbb{Q}_{\geq 0}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{Q}_{\geq 0}$

posrat_expt: JUDGEMENT expt(x: $\mathbb{Q}_{>0}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{Q}_{>0}$

int_expt: JUDGEMENT expt(x: $\mathbb{Z}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{Z}$

nat_expt: JUDGEMENT expt(x: $\mathbb{N}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{N}$

posnat_expt: JUDGEMENT expt(x: $\mathbb{N}_{>0}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{N}_{>0}$

r : $\mathbb{R}^{i:\ \{i:\ \mathbb{Z}\ \mid\ r\ \neq\ 0\ \text{OR}\ i\ \geq\ 0\}}$: $\mathbb{R}$ =
IF $i \geq 0$ THEN r^i ELSE $1/r^{-i}$ ENDIF

expt_pos: LEMMA px^i $>$ 0

expt_nonzero: LEMMA n0x^i $\neq$ 0

nnreal_exp: JUDGEMENT ^(x: $\mathbb{R}_{\geq 0}$, i: $\mathbb{Z}$ | $x \neq 0$ OR $i \geq 0$) HAS_TYPE $\mathbb{R}_{\geq 0}$

posreal_exp: JUDGEMENT ^(x: $\mathbb{R}_{>0}$, i: $\mathbb{Z}$) HAS_TYPE $\mathbb{R}_{>0}$

nzreal_exp: JUDGEMENT ^(x: $\mathbb{R}_{\neq 0}$, i: $\mathbb{Z}$) HAS_TYPE $\mathbb{R}_{\neq 0}$

rat_exp: JUDGEMENT ^(x: $\mathbb{Q}$, i: $\mathbb{Z}$ | $x \neq 0$ OR $i \geq 0$) HAS_TYPE $\mathbb{Q}$

nnrat_exp: JUDGEMENT ^(x: $\mathbb{Q}_{\geq 0}$, i: $\mathbb{Z}$ | $x \neq 0$ OR $i \geq 0$) HAS_TYPE $\mathbb{Q}_{\geq 0}$

posrat_exp: JUDGEMENT ^(x: $\mathbb{Q}_{>0}$, i: $\mathbb{Z}$) HAS_TYPE $\mathbb{Q}_{>0}$

int_exp: JUDGEMENT ^(x: $\mathbb{Z}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{Z}$

nat_exp: JUDGEMENT ^(x: $\mathbb{N}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{N}$

posint_exp: JUDGEMENT ^(x: $\mathbb{N}_{>0}$, n: $\mathbb{N}$) HAS_TYPE $\mathbb{N}_{>0}$

expt_x0_aux : LEMMA $x^0 = 1$

expt_x1_aux : LEMMA $x^1 = x$

expt_1n_aux : LEMMA $1^n = 1$

increasing_expt_aux : LEMMA $\text{gt1x}^{m+2} > \text{gt1x}$

decreasing_expt_aux : LEMMA $\text{lt1x}^{m+2} < \text{lt1x}$

expt_1_aux : LEMMA $\text{px}^{n+1} = 1$ IFF $\text{px} = 1$

expt_plus_aux : LEMMA
$\text{n0x}^{m+n} = \text{n0x}^m \times \text{n0x}^n$

expt_minus_aux : LEMMA
$m \geq n$ IMPLIES $\text{n0x}^{m-n} = \text{n0x}^m / \text{n0x}^n$

expt_times_aux : LEMMA $\text{n0x}^{m \times n} = \text{n0x}^{mn}$

expt_divide_aux : LEMMA
$1/\text{n0x}^{m \times n} = 1/\text{n0x}^{mn}$

both_sides_expt1_aux : LEMMA
$\text{px}^{m+1} = \text{px}^{n+1}$ IFF $m = n$ OR $\text{px} = 1$

both_sides_expt2_aux : LEMMA $\text{px}^{\text{pm}} = \text{py}^{\text{pm}}$ IFF $\text{px} = \text{py}$

both_sides_expt_pos_lt_aux : LEMMA
$\text{px}^{m+1} < \text{py}^{m+1}$ IFF $\text{px} < \text{py}$

both_sides_expt_gt1_lt_aux : LEMMA
$\text{gt1x}^{m+1} < \text{gt1x}^{n+1}$ IFF $m < n$

both_sides_expt_lt1_lt_aux : LEMMA
$\text{lt1x}^{m+1} < \text{lt1x}^{n+1}$ IFF $n < m$

both_sides_expt_pos_le_aux : LEMMA
$\text{px}^{m+1} \leq \text{py}^{m+1}$ IFF $\text{px} \leq \text{py}$

both_sides_expt_gt1_le_aux : LEMMA
$\text{gt1x}^{m+1} \leq \text{gt1x}^{n+1}$ IFF $m \leq n$

both_sides_expt_lt1_le_aux : LEMMA
$\text{lt1x}^{m+1} \le \text{lt1x}^{n+1}$ IFF $n \le m$

both_sides_expt_pos_gt_aux : LEMMA
$\text{px}^{m+1} > \text{py}^{m+1}$ IFF $\text{px} > \text{py}$

both_sides_expt_gt1_gt_aux : LEMMA
$\text{gt1x}^{m+1} > \text{gt1x}^{n+1}$ IFF $m > n$

both_sides_expt_lt1_gt_aux : LEMMA
$\text{lt1x}^{m+1} > \text{lt1x}^{n+1}$ IFF $n > m$

both_sides_expt_pos_ge_aux : LEMMA
$\text{px}^{m+1} \ge \text{py}^{m+1}$ IFF $\text{px} \ge \text{py}$

both_sides_expt_gt1_ge_aux : LEMMA
$\text{gt1x}^{m+1} \ge \text{gt1x}^{n+1}$ IFF $m \ge n$

both_sides_expt_lt1_ge_aux : LEMMA
$\text{lt1x}^{m+1} \ge \text{lt1x}^{n+1}$ IFF $n \ge m$

expt_of_mult : LEMMA
$x \times y^n = x^n \times y^n$

expt_of_div : LEMMA
$x/\text{n0y}^n = x^n/\text{n0y}^n$

expt_of_inv : LEMMA $1/\text{n0x}^n = 1/\text{n0x}^n$

expt_of_abs : LEMMA $|x|^n = |x^n|$

abs_of_expt_inv : LEMMA
$|1/\text{n0x}^n| = 1/|\text{n0x}|^n$

expt_x0 : LEMMA $x^0 = 1$

expt_x1 : LEMMA $x^1 = x$

expt_x2 : LEMMA $x^2 = x \times x$

expt_x3 : LEMMA $x^3 = x \times x \times x$

expt_x4: LEMMA $x^4 = x \times x \times x \times x$

expt_1i: LEMMA $1^i = 1$

expt_eq_0: LEMMA $x^{\mathrm{pn}} = 0$ IFF $x = 0$

expt_plus: LEMMA
$\mathrm{n0x}^{(i+j)} =$
$\mathrm{n0x}^i \times \mathrm{n0x}^j$

expt_times: LEMMA
$\mathrm{n0x}^{(i \times j)} = (\mathrm{n0x}^i)^j$

expt_inverse: LEMMA
$\mathrm{n0x}^{(-i)} = 1/(\mathrm{n0x}^i)$

expt_div: LEMMA
$\mathrm{n0x}^i/\mathrm{n0x}^j =$
$\mathrm{n0x}^{(i-j)}$

both_sides_expt1: LEMMA
$\mathrm{px}^{\mathrm{n0i}} = \mathrm{px}^{\mathrm{n0j}}$ IFF $\mathrm{n0i} = \mathrm{n0j}$ OR $\mathrm{px} = 1$

both_sides_expt2: LEMMA $\mathrm{px}^{\mathrm{n0i}} = \mathrm{py}^{\mathrm{n0i}}$ IFF $\mathrm{px} = \mathrm{py}$

b: VAR above(1)

pos_expt_gt: LEMMA $n < b^n$

expt_ge1: LEMMA $b^n \geq 1$

both_sides_expt_pos_lt: LEMMA
$\mathrm{px}^{\mathrm{pm}} < \mathrm{py}^{\mathrm{pm}}$ IFF $\mathrm{px} < \mathrm{py}$

both_sides_expt_gt1_lt: LEMMA
$\mathrm{gt1x}^i < \mathrm{gt1x}^j$ IFF $i < j$

both_sides_expt_lt1_lt: LEMMA
$\mathrm{lt1x}^i < \mathrm{lt1x}^j$ IFF $j < i$

both_sides_expt_pos_le: LEMMA

$\text{px}^{\text{pm}} \leq \text{py}^{\text{pm}}$ IFF $\text{px} \leq \text{py}$

both_sides_expt_gt1_le: LEMMA
$\text{gt1x}^{i} \leq \text{gt1x}^{j}$ IFF $i \leq j$

both_sides_expt_lt1_le: LEMMA
$\text{lt1x}^{i} \leq \text{lt1x}^{j}$ IFF $j \leq i$

both_sides_expt_pos_gt: LEMMA
$\text{px}^{\text{pm}} > \text{py}^{\text{pm}}$ IFF $\text{px} > \text{py}$

both_sides_expt_gt1_gt: LEMMA
$\text{gt1x}^{i} > \text{gt1x}^{j}$ IFF $i > j$

both_sides_expt_lt1_gt: LEMMA
$\text{lt1x}^{i} > \text{lt1x}^{j}$ IFF $j > i$

both_sides_expt_pos_ge: LEMMA
$\text{px}^{\text{pm}} \geq \text{py}^{\text{pm}}$ IFF $\text{px} \geq \text{py}$

both_sides_expt_gt1_ge: LEMMA
$\text{gt1x}^{i} \geq \text{gt1x}^{j}$ IFF $i \geq j$

both_sides_expt_lt1_ge: LEMMA
$\text{lt1x}^{i} \geq \text{lt1x}^{j}$ IFF $j \geq i$

expt_gt1_pos: LEMMA $\text{gt1x}^{\text{pm}} \geq \text{gt1x}$

expt_gt1_neg: LEMMA $\text{gt1x}^{(-\text{pm})} < 1$

expt_gt1_nonpos: LEMMA $\text{gt1x}^{(-m)} \leq 1$

mult_expt: LEMMA
$(\text{n0x} \times \text{n0y})^{i} = \text{n0x}^{i} \times \text{n0y}^{i}$

div_expt: LEMMA
$(\text{n0x}/\text{n0y})^{i} = \text{n0x}^{i}/\text{n0y}^{i}$

inv_expt: LEMMA
$(1/\text{n0x})^{i} = 1/\text{n0x}^{i}$

abs_expt: LEMMA $|\text{n0x}|^{i} = \left|\text{n0x}^{i}\right|$

abs_hat_nat: LEMMA $|x|^n = |x^n|$

expt_minus1_abs: LEMMA $\left|(-1)^i\right| = 1$

even_m1_pow: LEMMA even?(i) IMPLIES $(-1)^i = 1$

not_even_m1_pow: LEMMA
NOT even?(i) IMPLIES $(-1)^i = -1$

expt_lt1_bound1: LEMMA $\text{lt1x}^n \leq 1$

expt_lt1_bound2: LEMMA $\text{lt1x}^{\text{pn}} < 1$

expt_gt1_bound1: LEMMA $1 \leq \text{gt1x}^n$

expt_gt1_bound2: LEMMA $1 < \text{gt1x}^{\text{pn}}$

large_expt: LEMMA $1 < \text{px}$ IMPLIES $(\forall\ \text{py}: \exists\ n: \text{py} < \text{px}^n)$

small_expt: LEMMA $\text{px} < 1$ IMPLIES $(\forall\ \text{py}: \exists\ n: \text{px}^n < \text{py})$

exponent_adjust: LEMMA
$b^i + b^{(i-\text{pm})} <$
$b^{(i+1)}$

exp_of_exists: LEMMA
$\exists\ i: b^i \leq \text{py}\ \&\ \text{py} < b^{(i+1)}$

END exponentiation

euclidean_division : THEORY
BEGIN

$a, \ i$: VAR $\mathbb{N}$

b : VAR $\mathbb{N}_{>0}$

n : VAR $\mathbb{Z}$

mod(b) : TYPE+ = $\{i \mid i < b\}$

euclid_nat : LEMMA
$\exists \ (q : \mathbb{N}), \ (r : \text{mod}(b)) : a = b \times q + r$

euclid_int : PROPOSITION
$\exists \ (q : \mathbb{Z}), \ (r : \text{mod}(b)) : n = b \times q + r$

unique_quotient : PROPOSITION
$\forall \ (q_1, \ q_2 : \mathbb{Z}), \ (r_1, \ r_2 : \text{mod}(b))$:
$b \times q_1 + r_1 = b \times q_2 + r_2$ IMPLIES $q_1 = q_2$

unique_remainder : COROLLARY
$\forall \ (q_1, \ q_2 : \mathbb{Z}), \ (r_1, \ r_2 : \text{mod}(b))$:
$b \times q_1 + r_1 = b \times q_2 + r_2$ IMPLIES $r_1 = r_2$

unique_division : COROLLARY
$\forall \ (q_1, \ q_2 : \mathbb{Z}), \ (r_1, \ r_2 : \text{mod}(b))$:
$b \times q_1 + r_1 = b \times q_2 + r_2$ IMPLIES
$q_1 = q_2$ AND $r_1 = r_2$

END euclidean_division

divides : THEORY
BEGIN

n, m, l, x, y: VAR $\mathbb{Z}$

p, q: VAR $\mathbb{N}$

nz: VAR $\mathbb{Z}_{\neq 0}$

divides(n, m): bool = $\exists\ x: m = n \times x$

divides(n)(m): bool = divides(n, m)

mult_divides1 : JUDGEMENT $\times(n, m)$ HAS_TYPE (divides(n))

mult_divides2 : JUDGEMENT $\times(n, m)$ HAS_TYPE (divides(m))

divides_sum : LEMMA
divides(x, n) AND divides(x, m) IMPLIES divides(x, $n + m$)

divides_diff : LEMMA
divides(x, n) AND divides(x, m) IMPLIES divides(x, $n - m$)

divides_opposite : LEMMA divides(x, $-n$) IFF divides(x, n)

opposite_divides : LEMMA divides($-x$, n) IFF divides(x, n)

divides_prod1 : LEMMA divides(x, n) IMPLIES divides(x, $n \times m$)

divides_prod2 : LEMMA divides(x, n) IMPLIES divides(x, $m \times n$)

divides_prod_elim1 : LEMMA
divides(nz $\times$ n, nz $\times$ m) IFF divides(n, m)

divides_prod_elim2 : LEMMA
divides(n $\times$ nz, m $\times$ nz) IFF divides(n, m)

divides_reflexive : LEMMA divides(n, n)

divides_transitive : LEMMA
divides(n, m) AND divides(m, l) IMPLIES divides(n, l)

product_one : LEMMA
  $x \times y = 1$ IFF
   $(x = 1$ AND $y = 1)$ OR $(x = -1$ AND $y = -1)$

mutual_divisors : LEMMA
  divides(n, m) AND divides(m, n) IMPLIES $n = m$ OR $n = -m$

mutual_divisors_nat : LEMMA
  divides(p, q) AND divides(q, p) IMPLIES $p = q$

one_divides : LEMMA divides(1, n)

divides_zero : LEMMA divides(n, 0)

zero_div_zero : LEMMA divides(0, n) IFF $n = 0$

divisors_of_one : LEMMA divides(n, 1) IFF $n = 1$ OR $n = -1$

one_div_one : LEMMA divides(p, 1) IFF $p = 1$

divisor_smaller : LEMMA divides(p, q) IMPLIES $q = 0$ OR $p \leq q$

divides_next : LEMMA
  divides(n, $n + 1$) IFF $n = 1$ OR $n = -1$

p_1 : VAR above(1)

divides_plus_1 : LEMMA
  divides(p_1, nz) $\Rightarrow$ NOT divides(p_1, nz $+ 1$)

END divides

modulo_arithmetic : THEORY
BEGIN

x, y, z, t, q, i : VAR $\mathbb{Z}$

n0x : VAR $\mathbb{Z}_{\neq 0}$

nx : VAR $\mathbb{N}$

px, py, b : VAR $\mathbb{N}_{>0}$

n : VAR $\mathbb{N}$

nrem(x, b): $\{r$: mod(b) $\mid \exists\ q: x = b \times q + r\}$

rem(b)(x): $\{r$: mod(b) $\mid \exists\ q: x = b \times q + r\}$

rem_def : LEMMA
$\forall$ (r : mod(b)):
rem(b)(x) $= r$ IFF $\exists\ q: x = b \times q + r$

rem_def2 : LEMMA
$\forall$ (r : mod(b)):
rem(b)(x) $= r$ IFF divides(b, $x - r$)

rem_def3 : LEMMA
$\forall$ (r : mod(b)):
rem(b)(x) $= r$ IFF divides(b, $r - x$)

rem_nrem0 : LEMMA rem $= \lambda\ b: \lambda\ x$: nrem(x, b)

rem_nrem : LEMMA rem(b)(x) = nrem(x, b)

rem_mod : LEMMA $\forall$ (r : mod(b)): rem(b)(r) $= r$

rem_mod2 : LEMMA $0 \leq x$ AND $x < b$ IMPLIES rem(b)(x) $= x$

rem_zero : LEMMA rem(b)(0) $= 0$

rem_self : LEMMA rem(b)(b) $= 0$

rem_multiple1 : LEMMA rem(b)($b \times x$) $= 0$

rem_multiple2 : LEMMA rem(b)($x \times b$) = 0

rem_one : LEMMA $b \neq 1$ IMPLIES rem(b)(1) = 1

rem_minus_one : LEMMA rem(b)(-1) = $b - 1$

same_remainder : LEMMA
rem(b)(x) = rem(b)(y) IFF divides(b, $x - y$)

rem_rem : LEMMA rem(b)(rem(b)(x)) = rem(b)(x)

rem_sum : LEMMA
rem(b)(x) = rem(b)(y) AND rem(b)(z) = rem(b)(t) IMPLIES
rem(b)($x + z$) = rem(b)($y + t$)

rem_sum1 : LEMMA
rem(b)(rem(b)(x) + y) = rem(b)($x + y$)

rem_sum2 : LEMMA
rem(b)(x + rem(b)(y)) = rem(b)($x + y$)

rem_diff : LEMMA
rem(b)(x) = rem(b)(y) AND rem(b)(z) = rem(b)(t) IMPLIES
rem(b)($x - z$) = rem(b)($y - t$)

rem_diff1 : LEMMA
rem(b)(rem(b)(x) $-$ y) = rem(b)($x - y$)

rem_diff2 : LEMMA
rem(b)(x $-$ rem(b)(y)) = rem(b)($x - y$)

rem_prod1 : LEMMA
rem(b)(rem(b)(x) $\times$ y) = rem(b)($x \times y$)

rem_prod2 : LEMMA
rem(b)(x $\times$ rem(b)(y)) = rem(b)($x \times y$)

rem_prod : LEMMA
rem(b)(x) = rem(b)(y) AND rem(b)(z) = rem(b)(t) IMPLIES
rem(b)($x \times z$) = rem(b)($y \times t$)

rem_expt : LEMMA
  $\text{rem}(b)(x) = \text{rem}(b)(y)$ IMPLIES
    $\text{rem}(b)(x^n) = \text{rem}(b)(y^n)$

rem_expt1 : LEMMA
  $\text{rem}(b)(\text{rem}(b)(x)^n) = \text{rem}(b)(x^n)$

rem_sum_elim1 : LEMMA
  $\text{rem}(b)(x + y) = \text{rem}(b)(x + z)$ IFF
    $\text{rem}(b)(y) = \text{rem}(b)(z)$

rem_sum_elim2 : LEMMA
  $\text{rem}(b)(y + x) = \text{rem}(b)(z + x)$ IFF
    $\text{rem}(b)(y) = \text{rem}(b)(z)$

rem_diff_elim1 : LEMMA
  $\text{rem}(b)(x - y) = \text{rem}(b)(x - z)$ IFF
    $\text{rem}(b)(y) = \text{rem}(b)(z)$

rem_diff_elim2 : LEMMA
  $\text{rem}(b)(y - x) = \text{rem}(b)(z - x)$ IFF
    $\text{rem}(b)(y) = \text{rem}(b)(z)$

rem_opposite_elim : LEMMA
  $\text{rem}(b)(-x) = \text{rem}(b)(-y)$ IFF
    $\text{rem}(b)(x) = \text{rem}(b)(y)$

$\text{ndiv}(x,\ b) : \{q : \mathbb{Z} \mid x = b \times q + \text{rem}(b)(x)\}$

ndiv_lt : LEMMA $\text{ndiv}(x,\ b) \leq x/b$

JUDGEMENT $\text{ndiv}(n,\ b)$ HAS_TYPE $\text{upto}(n)$

rem_floor : LEMMA
  $\forall\ b,\ x :\ x = \text{rem}(b)(x) + b \times \lfloor x/b \rfloor$

rem_base : LEMMA
  $\forall\ b,\ x,\ i,\ n :$
    $\text{rem}(b)(x) = \text{rem}(b + n)(x + i)$ IFF
      $\text{divides}(b + n,\ i - n \times \lfloor x/b \rfloor)$

rem_sum_floor : LEMMA

$\forall$ b, x, i:
  rem(b)($x + i$) $=$
    rem(b)(x) $+ i - b \times \lfloor$(rem(b)(x) $+ i)/b\rfloor$

rem_sum_assoc: COROLLARY
  $\forall$ b, x, n:
    rem(b)($x + n$) $=$ rem(b)(x) $+ n$ IFF
      rem(b)(x) $<$ $b - n$

rem_add_one: LEMMA
  $\forall$ b, x:
    rem(b)($x + 1$) $=$ rem(b)(x) $+ 1$ OR
      (rem(b)(x) $=$ $b - 1$ AND
        rem(b)($x + 1$) $=$ 0)

rem_wrap: LEMMA
  $\forall$ b, x, (n: below(b)):
    rem(b)(x) $<$ rem(b)($x + n$) IFF
      rem(b)(x) $<$ $b - n$ AND $n > 0$

rem_wrap_eq: COROLLARY
  $\forall$ b, x, (n: below(b)):
    rem(b)(x) $\leq$ rem(b)($x + n$) IFF
      rem(b)(x) $<$ $b - n$ OR divides(b, n)

END modulo_arithmetic

subrange_inductions$[i\colon \mathbb{Z},\ j\colon \text{upfrom}(i)]$: THEORY
BEGIN

$k,\ m$: VAR subrange$(i,\ j)$

p: VAR pred[subrange$(i,\ j)$]

subrange_induction: LEMMA
$(p(i)$ AND $(\forall\ k\colon k < j$ AND $p(k)$ IMPLIES $p(k+1)))$ IMPLIES
$(\forall\ k\colon p(k))$

SUBRANGE_induction: LEMMA
$(\forall\ k\colon (\forall\ m\colon m < k$ IMPLIES $p(m))$ IMPLIES $p(k))$ IMPLIES
$(\forall\ k\colon p(k))$

END subrange_inductions

```
bounded_int_inductions[m: ℤ]: THEORY
 BEGIN

  pf: VAR pred[upfrom(m)]

  jf, kf: VAR upfrom(m)

  upfrom_induction: LEMMA
    (pf(m) AND (∀ jf: pf(jf) IMPLIES pf(jf + 1))) IMPLIES
      (∀ jf: pf(jf))

  UPFROM_induction: LEMMA
    (∀ jf: (∀ kf: kf < jf IMPLIES pf(kf)) IMPLIES pf(jf)) IMPLIES
      (∀ jf: pf(jf))

  pa: VAR pred[above(m)]

  ja, ka: VAR above(m)

  above_induction: LEMMA
    (pa(m + 1) AND (∀ ja: pa(ja) IMPLIES pa(ja + 1))) IMPLIES
      (∀ ja: pa(ja))

  ABOVE_induction: LEMMA
    (∀ ja: (∀ ka: ka < ja IMPLIES pa(ka)) IMPLIES pa(ja)) IMPLIES
      (∀ ja: pa(ja))

 END bounded_int_inductions
```

```
bounded_nat_inductions[m: ℕ]: THEORY
 BEGIN

  pt: VAR pred[upto(m)]

  jt, kt: VAR upto(m)

  upto_induction: LEMMA
    (pt(0) AND (∀ jt: jt < m AND pt(jt) IMPLIES pt(jt + 1))) IMPLIES
      (∀ jt: pt(jt))

  UPTO_induction: LEMMA
    (∀ jt: (∀ kt: kt < jt IMPLIES pt(kt)) IMPLIES pt(jt)) IMPLIES
      (∀ jt: pt(jt))

  pb: VAR pred[below(m)]

  jb, kb: VAR below(m)

  below_induction: LEMMA
    ((m > 0 IMPLIES pb(0)) AND
         (∀ jb: jb < m − 1 AND pb(jb) IMPLIES pb(jb + 1)))
      IMPLIES (∀ jb: pb(jb))

  BELOW_induction: LEMMA
    (∀ jb: (∀ kb: kb < jb IMPLIES pb(kb)) IMPLIES pb(jb)) IMPLIES
      (∀ jb: pb(jb))

 END bounded_nat_inductions
```

subrange_type[m, n: $\mathbb{Z}$]: THEORY
BEGIN

subrange: TYPE = subrange(m, n)

END subrange_type

int_types$[m: \mathbb{Z}]$: THEORY
BEGIN

upfrom: TYPE+ = upfrom(m)

above: TYPE+ = above(m)

END int_types

nat_types$[m: \mathbb{N}]$: THEORY
BEGIN

upto: TYPE+ = upto(m)

below: TYPE = below(m)

END nat_types

nat_fun_props : THEORY
BEGIN

$n, \ m$: VAR $\mathbb{N}$

injection_n_to_m : THEOREM
$(\exists \ (f: \ [\text{below}(n) \ \to \ \text{below}(m)]): \ \text{injective?}(f))$ IMPLIES
$n \ \leq \ m$

injection_n_to_m_var : THEOREM
$(\exists \ (f: \ [\text{below}(n) \ \to \ \text{below}(m)]): \ \text{injective?}(f))$ IFF
$n \ \leq \ m$

surjection_n_to_m : THEOREM
$(\exists \ (f: \ [\text{below}(n) \ \to \ \text{below}(m)]): \ \text{surjective?}(f))$ IMPLIES
$m \ \leq \ n$

surjection_n_to_m_var : THEOREM
$(\exists \ (f: \ [\text{below}(n) \ \to \ \text{below}(m)]): \ \text{surjective?}(f))$ IFF
$(m \ > \ 0$ AND $m \ \leq \ n)$ OR $(m \ = \ 0$ AND $n \ = \ 0)$

bijection_n_to_m : THEOREM
$(\exists \ (f: \ [\text{below}(n) \ \to \ \text{below}(m)]): \ \text{bijective?}(f))$ IFF
$n \ = \ m$

injection_n_to_m2 : THEOREM
$(\exists \ (f: \ [\text{upto}(n) \ \to \ \text{upto}(m)]): \ \text{injective?}(f))$ IFF
$n \ \leq \ m$

surjection_n_to_m2 : THEOREM
$(\exists \ (f: \ [\text{upto}(n) \ \to \ \text{upto}(m)]): \ \text{surjective?}(f))$ IFF
$m \ \leq \ n$

bijection_n_to_m2 : THEOREM
$(\exists \ (f: \ [\text{upto}(n) \ \to \ \text{upto}(m)]): \ \text{bijective?}(f))$ IFF
$n \ = \ m$

surj_equiv_inj : THEOREM
$\forall \ (f: \ [\text{below}(n) \ \to \ \text{below}(n)])$:
surjective?(f) IFF injective?(f)

inj_equiv_bij : THEOREM

$\forall$ (f: [below(n) $\rightarrow$ below(n)]):
bijective?(f) IFF injective?(f)

surj_equiv_bij: THEOREM
$\forall$ (f: [below(n) $\rightarrow$ below(n)]):
bijective?(f) IFF surjective?(f)

surj_equiv_inj2: THEOREM
$\forall$ (f: [upto(n) $\rightarrow$ upto(n)]):
surjective?(f) IFF injective?(f)

inj_equiv_bij2: THEOREM
$\forall$ (f: [upto(n) $\rightarrow$ upto(n)]):
bijective?(f) IFF injective?(f)

surj_equiv_bij2: THEOREM
$\forall$ (f: [upto(n) $\rightarrow$ upto(n)]):
bijective?(f) IFF surjective?(f)

END nat_fun_props

finite_sets[T: TYPE]: THEORY
BEGIN

x, y, z: VAR T

s: VAR set[T]

N: VAR $\mathbb{N}$

is_finite(s): bool =
(∃ N, (f: [(s) → below[N]]): injective?(f))

finite_set: TYPE = (is_finite) CONTAINING $\emptyset$[T]

non_empty_finite_set: TYPE = {s: finite_set | NOT empty?(s)}

is_finite_surj: LEMMA
(∃ (N: $\mathbb{N}$), (f: [below[N] → (s)]): surjective?(f)) IFF
is_finite(s)

A, B: VAR finite_set

NA, NB: VAR non_empty_finite_set

finite_subset: LEMMA $(s \subseteq A)$ IMPLIES is_finite(s)

finite_intersection: LEMMA is_finite($(A \cap B)$)

finite_add: LEMMA is_finite($(A \cup \{x\})$)

nonempty_finite_is_nonempty: JUDGEMENT non_empty_finite_set SUBTYPE_OF
(nonempty?[T])

finite_singleton: JUDGEMENT singleton(x) HAS_TYPE finite_set

finite_union: JUDGEMENT union(A, B) HAS_TYPE finite_set

finite_intersection1: JUDGEMENT intersection(s, A) HAS_TYPE
finite_set

finite_intersection2: JUDGEMENT intersection(A, s) HAS_TYPE
finite_set

finite_difference: JUDGEMENT difference(A, s) HAS_TYPE finite_set

nonempty_finite_union1: JUDGEMENT union(NA, B) HAS_TYPE
non_empty_finite_set

nonempty_finite_union2: JUDGEMENT union(A, NB) HAS_TYPE
non_empty_finite_set

nonempty_add_finite: JUDGEMENT add(x, A) HAS_TYPE
non_empty_finite_set

finite_remove: JUDGEMENT remove(x, A) HAS_TYPE finite_set

finite_rest: JUDGEMENT rest(A) HAS_TYPE finite_set

finite_emptyset: JUDGEMENT $\emptyset$ HAS_TYPE finite_set

nonempty_singleton_finite: JUDGEMENT singleton(x) HAS_TYPE
non_empty_finite_set

is_finite_type: bool =
($\exists$ N, (g: $[T \rightarrow$ below$[N]]$): injective?(g))

finite_full: LEMMA is_finite_type IFF is_finite(fullset$[T]$)

finite_type_set: LEMMA is_finite_type IMPLIES is_finite(s)

finite_complement: LEMMA is_finite_type IMPLIES is_finite($\overline{s}$)

S, S_2: VAR finite_set

n, m: VAR $\mathbb{N}$

p: VAR $\mathbb{N}_{>0}$

inj_set(S): (nonempty?$[\mathbb{N}]$) =
$\{n \mid \exists$ (f: $[(S) \rightarrow$ below$[n]]$): injective?(f)$\}$

Card(S): $\mathbb{N}$ = min(inj_set(S))

inj_Card: LEMMA

Card(S) = n IMPLIES
  (∃ (f: [(S) → below[n]]): injective?(f))

reduce_inj: LEMMA
  (∀ (f: [(S) → below[p]]):
    injective?(f) AND NOT surjective?(f) IMPLIES
      (∃ (g: [(S) → below[$p - 1$]]):
        injective?(g)))

Card_injection: LEMMA
  (∃ (f: [(S) → below[n]]): injective?(f)) IMPLIES
    Card(S) $\leq$ n

Card_surjection: LEMMA
  (∃ (f: [(S) → below[n]]): surjective?(f)) IMPLIES
    n $\leq$ Card(S)

Card_bijection: THEOREM
  Card(S) = n IFF
    (∃ (f: [(S) → below[n]]): bijective?(f))

Card_disj_union: THEOREM
  disjoint?(S, S_2) IMPLIES
    Card(($S \cup S_2$)) = Card(S) + Card(S_2)

card(S): {n: ℕ | n = Card(S)}

card_def: THEOREM card(S) = Card(S)

card_emptyset: THEOREM card(∅[T]) = 0

empty_card: THEOREM empty?(S) IFF card(S) = 0

card_empty?: THEOREM (card(S) = 0) = empty?(S)

card_is_0: THEOREM (card(S) = 0) = (S = ∅)

nonempty_card: THEOREM nonempty?(S) IFF card(S) > 0

card_singleton: THEOREM card(singleton(x)) = 1

card_one: THEOREM card(S) = 1 IFF (∃ x: S = singleton(x))

card_disj_union : THEOREM
  disjoint?(A, B) IMPLIES
    card(($A \cup B$)) = card(A) + card(B)

card_diff_subset : THEOREM
  ($A \subseteq B$) IMPLIES
    card(($B \setminus A$)) = card(B) − card(A)

card_subset : THEOREM ($A \subseteq B$) IMPLIES card(A) $\leq$ card(B)

card_plus : THEOREM
  card(A) + card(B) =
    card(($A \cup B$)) + card(($A \cap B$))

card_union : THEOREM
  card(($A \cup B$)) =
    card(A) + card(B) − card(($A \cap B$))

card_add : THEOREM
  card(($S \cup \{x\}$)) =
    card(S) + IF $S(x)$ THEN 0 ELSE 1 ENDIF

card_add_gt0 : THEOREM card(($S \cup \{x\}$)) > 0

card_remove : THEOREM
  card(($S \setminus \{x\}$)) =
    card(S) − IF $S(x)$ THEN 1 ELSE 0 ENDIF

card_rest : THEOREM
  NOT empty?(S) IMPLIES card(rest(S)) = card(S) − 1

same_card_subset : THEOREM
  ($A \subseteq B$) AND card(A) = card(B) IMPLIES $A = B$

smaller_card_subset : THEOREM
  ($A \subseteq B$) AND card(A) < card(B) IMPLIES
    ($\exists$ x : ($x \in B$) AND NOT ($x \in A$))

card_strict_subset : THEOREM ($A \subset B$) IMPLIES card(A) < card(B)

card_1_has_1 : THEOREM card(S) $\geq$ 1 IMPLIES ($\exists$ (x : T) : $S(x)$)

card_2_has_2 : THEOREM
card(S) $\geq$ 2 IMPLIES
($\exists$ (x, y: T): $x \neq y$ AND $S(x)$ AND $S(y)$)

card_intersection_le : THEOREM
card(($A \cap B$)) $\leq$ card(A) AND
card(($A \cap B$)) $\leq$ card(B)

card_bij : THEOREM
card(S) $=$ N IFF
($\exists$ (f: [(S) $\rightarrow$ below[N]]): bijective?(f))

card_bij_inv : THEOREM
card(S) $=$ N IFF
($\exists$ (f: [below[N] $\rightarrow$ (S)]): bijective?(f))

bij_exists : COROLLARY
($\exists$ (f: [(S) $\rightarrow$ below(card(S))]): bijective?(f))

bij(S: finite_set):
{f: [(S) $\rightarrow$ below(card(S))] | bijective?(f)}

ibij(S: non_empty_finite_set): {f: [below(card(S)) $\rightarrow$ (S)] | bijective?(f)} =
inverse(bij(S))

bij_ibij : LEMMA
$\forall$ (S: non_empty_finite_set, ii: below(card(S))):
bij(S)(ibij(S)(ii)) $=$ ii

ibij_bij : LEMMA
$\forall$ (S: non_empty_finite_set, x: T):
$S(x)$ IMPLIES ibij(S)(bij(S)(x)) $=$ x

is_finite_exists_N : LEMMA
$\forall$ (g: [below[N] $\rightarrow$ T]):
is_finite({r: T | $\exists$ (n: below[N]): r $=$ $g(n)$})

P, P_1, P_2: VAR pred[T]

finite_pred : LEMMA
is_finite(fullset[T]) IMPLIES

is_finite[T]({x: T | ℙ(x)})

finite_pred2: LEMMA
is_finite(P) IMPLIES is_finite[T]({x: T | ℙ(x)})

card_implies: LEMMA
is_finite(fullset[T]) AND (∀ (x: T): $P_1(x)$ IMPLIES $P_2(x)$) IMPLIES
card({x: T | $P_1(x)$}) ≤ card({x: T | $P_2(x)$})

finite_induction: THEOREM
∀ (p: pred[set[T]]):
(∀ (n: ℕ), (S: set[T]):
(∃ (f: [(S) → below[n]]): injective?(f)) ⇒ p(S))
⇒ (∀ (FS: finite_set): p(FS))

END finite_sets

restrict_set_props[T: TYPE, S: TYPE FROM T]: THEORY
BEGIN

restrict_finite: LEMMA
∀ (a: set[T]):
is_finite(a) ⇒ is_finite(restrict[T, S, bool](a))

finite_restrict: JUDGEMENT restrict[T, S, bool](a: finite_set[T]) HAS_TYPE
finite_set[S]

empty_restrict: JUDGEMENT restrict[T, S, bool](a: (empty?[T])) HAS_TYPE
(empty?[S])

card_restrict: LEMMA
∀ (a: finite_set[T]):
card(restrict[T, S, bool](a)) ≤ card(a)

END restrict_set_props

extend_set_props[T: TYPE, S: TYPE FROM T]: THEORY
BEGIN

finite_extension: LEMMA
$\forall$ (a: set[S]):
is_finite(extend[T, S, bool, FALSE](a)) IFF is_finite(a)

finite_extend: JUDGEMENT extend[T, S, bool, FALSE](a: finite_set[S]) HAS_TYPE
finite_set[T]

empty_extend: JUDGEMENT extend[T, S, bool, FALSE](a: (empty?[S])) HAS_TYPE
(empty?[T])

nonempty_extend: JUDGEMENT extend[T, S, bool, FALSE](a: (nonempty?[S])) HAS_TYPE
(nonempty?[T])

singleton_extend: JUDGEMENT extend[T, S, bool, FALSE](a: (singleton?[S])) HAS_TYPE
(singleton?[T])

card_extend: LEMMA
$\forall$ (a: finite_set[S]):
card(extend[T, S, bool, FALSE](a)) = card(a)

empty?_extend: LEMMA
$\forall$ (a: set[S]):
empty?(extend[T, S, bool, FALSE](a)) IFF empty?(a)

nonempty?_extend: LEMMA
$\forall$ (a: set[S]):
nonempty?(extend[T, S, bool, FALSE](a)) IFF nonempty?(a)

singleton?_extend: LEMMA
$\forall$ (a: set[S]):
singleton?(extend[T, S, bool, FALSE](a)) IFF singleton?(a)

subset_extend: LEMMA
$\forall$ (a, b: set[S]):
(extend[T, S, bool, FALSE](a) $\subseteq$ extend[T, S, bool, FALSE](b)) IFF
($a \subseteq b$)

union_extend: LEMMA
$\forall$ (a, b: set[S]):

$(\text{extend}[T, S, \text{bool}, \text{FALSE}](a) \cup \text{extend}[T, S, \text{bool}, \text{FALSE}](b)) =$
$\text{extend}[T, S, \text{bool}, \text{FALSE}]((a \cup b))$

intersection_extend: LEMMA
$\forall (a, b: \text{set}[S]):$
$(\text{extend}[T, S, \text{bool}, \text{FALSE}](a) \cap \text{extend}[T, S, \text{bool}, \text{FALSE}](b)) =$
$\text{extend}[T, S, \text{bool}, \text{FALSE}]((a \cap b))$

difference_extend: LEMMA
$\forall (a, b: \text{set}[S]):$
$(\text{extend}[T, S, \text{bool}, \text{FALSE}](a) \setminus \text{extend}[T, S, \text{bool}, \text{FALSE}](b)) =$
$\text{extend}[T, S, \text{bool}, \text{FALSE}]((a \setminus b))$

add_extend: LEMMA
$\forall (x: S, a: \text{set}[S]):$
$(\text{extend}[T, S, \text{bool}, \text{FALSE}](a) \cup \{x\}) =$
$\text{extend}[T, S, \text{bool}, \text{FALSE}]((a \cup \{x\}))$

remove_extend: LEMMA
$\forall (x: S, a: \text{set}[S]):$
$(\text{extend}[T, S, \text{bool}, \text{FALSE}](a) \setminus \{x\}) =$
$\text{extend}[T, S, \text{bool}, \text{FALSE}]((a \setminus \{x\}))$

END extend_set_props

```
function_image_aux[D: TYPE, R: TYPE]: THEORY
 BEGIN

  S: VAR finite_set[D]

  f: VAR [D → R]

  inj: VAR (injective?[D, R])

  finite_image: JUDGEMENT image(f, S) HAS_TYPE finite_set[R]

  card_image: LEMMA ∀ f, S: card(image(f, S)) ≤ card(S)

  card_injective_image: LEMMA
    ∀ inj, S: card(image(inj, S)) = card(S)

  bijective_image: LEMMA
    ∀ inj: bijective?[D, (image(inj, fullset[D]))](inj)

 END function_image_aux
```

```
function_iterate[T: TYPE]: THEORY
 BEGIN

  f: VAR [T -> T]

  m, n: VAR ℕ

  x: VAR T

  iterate(f, n)(x): RECURSIVE T =
    IF n = 0 THEN x ELSE f(iterate(f, n - 1)(x)) ENDIF
     MEASURE n

  iterate_add: LEMMA
    iterate(f, m) o iterate(f, n) =
     iterate(f, m + n)

  iterate_add_applied: LEMMA
    iterate(f, m)(iterate(f, n)(x)) =
     iterate(f, m + n)(x)

  iterate_add_one: LEMMA
    iterate(f, n)(f(x)) = iterate(f, n + 1)(x)

  iterate_mult: LEMMA
    iterate(iterate(f, m), n) = iterate(f, m × n)

  iterate_invariant: LEMMA
    f(iterate(f, n)(x)) = iterate(f, n)(f(x))

 END function_iterate
```

```
sequences[T: TYPE]: THEORY
 BEGIN

  sequence: TYPE = [ℕ → T]

  i, n: VAR ℕ

  x: VAR T

  p: VAR pred[T]

  seq: VAR sequence

  Trel: VAR PRED[[T]]

  nth(seq, n): T = seq(n)

  suffix(seq, n): sequence = (λ i: seq(i + n))

  first(seq): T = nth(seq, 0)

  rest(seq): sequence = suffix(seq, 1)

  delete(n, seq): sequence =
        (λ i:
              (IF i < n THEN seq(i) ELSE seq(i + 1) ENDIF))

  insert(x, n, seq): sequence =
        (λ i:
              (IF i < n
                    THEN seq(i)
                 ELSIF i = n THEN x
                 ELSE seq(i − 1)
                 ENDIF))

  add(x, seq): sequence = insert(x, 0, seq)

  insert_delete: LEMMA
    insert(nth(seq, n), n, delete(n, seq)) = seq

  add_first_rest: LEMMA add(first(seq), rest(seq)) = seq
```

every(p)(seq): bool = ($\forall$ n: p(nth(seq, n)))

every(p, seq): bool = ($\forall$ n: p(nth(seq, n)))

some(p)(seq): bool = ($\exists$ n: p(nth(seq, n)))

some(p, seq): bool = ($\exists$ n: p(nth(seq, n)))

sequence_induction: LEMMA
  p(nth(seq, 0)) AND ($\forall$ n: p(nth(seq, n)) IMPLIES p(nth(seq, $n+1$))) IMPLIES
    every(p)(seq)

ascends?(seq, Trel): bool =
    preserves(seq, (λ i, n: $i \leq n$), Trel)

descends?(seq, Trel): bool =
    inverts(seq, (λ i, n: $i \leq n$), Trel)

END sequences

seq_functions$[D, R:$ TYPE$]:$ THEORY

BEGIN

$f:$ VAR $[D \rightarrow R]$

$s:$ VAR sequence$[D]$

$n:$ VAR $\mathbb{N}$

map$(f)(s)$: sequence$[R]$ = $(\lambda\ n:\ f(\text{nth}(s, n)))$

map(f, s): sequence$[R]$ = $(\lambda\ n:\ f(\text{nth}(s, n)))$

END seq_functions

```
finite_sequences[T: TYPE]: THEORY
 BEGIN

  finite_sequence: TYPE = [#length: ℕ, seq: [below[length] → T]#]

  finseq: TYPE = finite_sequence

  fs, fs1, fs2, fs3: VAR finseq

  m, n: VAR ℕ

  empty_seq: finseq =
        (#length := 0,
          seq := (λ (x: below[0]): ε! (t: T): TRUE)#)

  finseq_appl(fs): [below[length(fs)] → T] = fs`seq;

  CONVERSION finseq_appl


  fs1 ∘ fs2: finseq =
        LET l_1 = fs1`length, lsum = l_1 + fs2`length IN
          (#length := lsum,
            seq
              := (λ (n: below[lsum]):
                         IF n < l_1
                           THEN fs1`seq(n)
                         ELSE fs2`seq(n − l_1)
                         ENDIF)#);

  p: VAR [ℕ]

  fs^p: finseq =
        LET (m, n) = p IN
          IF m > n OR m ≥ fs`length
            THEN empty_seq
          ELSE LET len = min(n − m + 1, fs`length − m) IN
                 (#length := len, seq := (λ (x: below[len]): fs`seq(x + m))#)
          ENDIF;

  ^^(fs, p): finseq =
        LET (m, n) = p IN
```

```
IF m ≥ n OR m ≥ fs`length
   THEN empty_seq
ELSE LET len = min(n − m, fs`length − m) IN
        (#length := len, seq := (λ (x: below[len]): fs`seq(x + m))#)
ENDIF

extract1(fs: {fs | fs`length = 1}): T = fs`seq(0)

CONVERSION extract1


o_assoc: LEMMA
  fs1 ∘ (fs2 ∘ fs3) = (fs1 ∘ fs2) ∘ fs3

END finite_sequences
```

```
more_finseq[T: TYPE]: THEORY
 BEGIN

  seq: TYPE = finseq[T]

  rr, ss, tt: VAR seq

  x, y, z: VAR T

  prefix?(rr, ss): bool =
        rr`length ≤ ss`length AND
         (∀ (i: below(rr`length)): rr`seq(i) = ss`seq(i))

  prefix_closed?(X: set[seq]): bool =
        ∀ ss: X(ss) IMPLIES (∀ (rr: seq | prefix?(rr, ss)): X(rr))

  add(x, rr): finseq[T] =
        rr WITH [`length := rr`length + 1, `seq(rr`length) := x]

 END more_finseq
```

```
ordstruct : DATATYPE
  BEGIN
    zero : zero?
    add(coef : ℕ_{>0}, exp : ordstruct,
              rest : ordstruct)
          : nonzero?
  END ordstruct
```

```
ordinals: THEORY
 BEGIN

  i, j, k: VAR ℕ>0

  m, n, O: VAR ℕ

  u, v, w, x, y, z: VAR ordstruct

  size: [ordstruct → ℕ] =
      reduce_nat(0, (λ i, m, n: 1 + m + n));

  <(x, y): RECURSIVE bool =
    CASES x OF
      zero: NOT zero?(y),
      add(i, u, v):
        CASES y OF
          zero: FALSE,
          add(j, z, w):
            (u < z) OR
             (u = z) AND (i < j) OR (u = z) AND (i = j) AND (v < w)
          ENDCASES
      ENDCASES
     MEASURE size(x);

  >(x, y): bool = y < x;

  ≤(x, y): bool = x < y OR x = y;

  ≥(x, y): bool = y < x OR y = x

  ordinal?(x): RECURSIVE bool =
    CASES x OF
      zero: TRUE,
      add(i, u, v):
        (ordinal?(u) AND
            ordinal?(v) AND CASES v OF zero: TRUE, add(k, r, s): r < u ENDCASES)
      ENDCASES
     MEASURE size

  ordinal: TYPE+ = (ordinal?)
```

r, s, t: VAR ordinal

ordinal_irreflexive: LEMMA NOT $r < r$

ordinal_antisym: LEMMA $r < s$ IMPLIES NOT $s < r$

ordinal_antisymmetric: LEMMA $r \leq s$ AND $s \leq r$ IMPLIES $r = s$

ordinal_transitive: LEMMA $r < s$ AND $s < t$ IMPLIES $r < t$

ordinal_trichotomy: LEMMA $r < s$ OR $r = s$ OR $s < r$

p: VAR pred[ordinal]

ordinal_induction: AXIOM
  ($\forall$ r: ($\forall$ s: $s < r$ IMPLIES $p(s)$) IMPLIES $p(r)$) IMPLIES
    ($\forall$ r: $p(r)$)

well_founded_le: LEMMA
  well_founded?(λ (r, s: (ordinal?)): $r < s$)

END ordinals

```
lex2: THEORY
 BEGIN

  i, j, m, n: VAR ℕ

  lex2(m, n): ordinal =
        (IF m = 0
            THEN IF n = 0 THEN zero ELSE add(n, zero, zero) ENDIF
         ELSIF n = 0 THEN add(m, add(1, zero, zero), zero)
         ELSE add(m, add(1, zero, zero), add(n, zero, zero))
         ENDIF)

  lex2_lt: LEMMA
    (lex2(i, j) < lex2(m, n)) =
     (i < m OR (i = m AND j < n))

 END lex2
```

```
list[T: TYPE]: DATATYPE
  BEGIN
   null: null?
   cons(car: T, cdr: list): cons?
  END list
```

```
list_props[T: TYPE]: THEORY
 BEGIN

  l, l1, l2, l3: VAR list[T]

  x: VAR T

  P, Q: VAR PRED[T]

  length(l): RECURSIVE ℕ =
    CASES l OF null: 0, cons(x, y): length(y) + 1 ENDCASES
     MEASURE reduce_nat(0, (λ (x: T), (n: ℕ): n + 1))

  member(x, l): RECURSIVE bool =
    CASES l OF null: FALSE, cons(hd, tl): x = hd OR member(x, tl) ENDCASES
     MEASURE length(l)

  member_null: LEMMA member(x, l) IMPLIES NOT null?(l)

  nth(l, (n: below[length(l)])): RECURSIVE T =
    IF n = 0 THEN car(l) ELSE nth(cdr(l), n − 1) ENDIF
     MEASURE length(l)

  append(l1, l2): RECURSIVE list[T] =
    CASES l1 OF null: l2, cons(x, y): cons(x, append(y, l2)) ENDCASES
     MEASURE length(l1)

  reverse(l): RECURSIVE list[T] =
    CASES l OF null: l, cons(x, y): append(reverse(y), cons(x, null)) ENDCASES
     MEASURE length

  append_null: LEMMA append(l, null) = l

  append_assoc: LEMMA
    append(append(l1, l2), l3) = append(l1, append(l2, l3))

  reverse_append: LEMMA
    reverse(append(l1, l2)) = append(reverse(l2), reverse(l1))

  reverse_reverse: LEMMA reverse(reverse(l)) = l

  length_append: LEMMA
```

length(append(l_1, l_2)) = length(l_1) + length(l_2)

length_reverse: LEMMA length(reverse(l)) = length(l)

a, b, c: VAR T

list_rep: LEMMA
(:a, b, c:) = cons(a, cons(b, cons(c, null)))

every_append: LEMMA
every(P)(append(l_1, l_2)) IFF
(every(P)(l_1) AND every(P)(l_2))

every_disjunct1: LEMMA
every(P)(l) IMPLIES
every(λ (x: T): $\mathbb{P}(x)$ OR $Q(x)$)(l)

every_disjunct2: LEMMA
every(Q)(l) IMPLIES
every(λ (x: T): $\mathbb{P}(x)$ OR $Q(x)$)(l)

every_conjunct: LEMMA
every(λ (x: T): $\mathbb{P}(x)$ AND $Q(x)$)(l) $\Rightarrow$
(every(P)(l) AND every(Q)(l))

every_conjunct2: LEMMA
(every(P)(l) AND every(Q)(l)) $\Rightarrow$
every(λ (x: T): $\mathbb{P}(x)$ AND $Q(x)$)(l)

every_member: LEMMA every({c: T | member(c, l)})(l)

every_nth: LEMMA
every(P)(l) IFF
$\forall$ (i: below(length(l))): $\mathbb{P}$(nth(l, i))

END list_props

map_props$[T_1, \; T_2, \; T_3 :$ TYPE$]$: THEORY
BEGIN

f_1 : VAR $[T_1 \rightarrow T_2]$

f_2 : VAR $[T_2 \rightarrow T_3]$

s : VAR sequence$[T_1]$

l : VAR list$[T_1]$

map_list_composition : LEMMA
map(f_2)(map$(f_1)(l)$) $=$ map$(f_2 \circ f_1)(l)$

map_seq_composition : LEMMA
map(f_2)(map$(f_1)(s)$) $=$ map$(f_2 \circ f_1)(s)$

END map_props

more_map_props$[T_1,\ T_2:$ TYPE$]:$ THEORY
BEGIN

$f:$ VAR $[T_1 \rightarrow T_2]$

$l:$ VAR list$[T_1]$

map_length: LEMMA length(map(f)(l)) $=$ length(l)

map_nth_rw: LEMMA
$\forall$ (i: below(length(l))):
nth(map(f)(l), i) $=$ f(nth(l, i))

END more_map_props

```
filters[T: TYPE]: THEORY
 BEGIN

  s: VAR set[T]

  l: VAR list[T]

  p: VAR pred[T]

  filter(s, p): set[T] = {x: T | s(x) & p(x)}

  filter(p)(s): set[T] = {x: T | s(x) & p(x)}

  filter(l, p): RECURSIVE list[T] =
    CASES l OF
      null: null,
      cons(x, y): IF p(x) THEN cons(x, filter(y, p)) ELSE filter(y, p) ENDIF
      ENDCASES
     MEASURE length(l)

  filter(p)(l): RECURSIVE list[T] =
    CASES l OF
      null: null,
      cons(x, y): IF p(x) THEN cons(x, filter(p)(y)) ELSE filter(p)(y) ENDIF
      ENDCASES
     MEASURE length(l)

 END filters
```

```
list2finseq[T: TYPE]: THEORY
 BEGIN

  l: VAR list[T]

  fs: VAR finseq[T]

  n: VAR ℕ

  list2finseq(l): finseq[T] =
        (#length := length(l),
           seq := (λ (x: below[length(l)]): nth(l, x))#)

  finseq2list_rec(fs, (n: ℕ | n ≤ length(fs))): RECURSIVE list[T] =
    IF n = 0
      THEN null
    ELSE cons(fs`seq(length(fs) − n), finseq2list_rec(fs, n − 1))
    ENDIF
     MEASURE n

  finseq2list(fs): list[T] = finseq2list_rec(fs, length(fs))

  CONVERSION list2finseq


  CONVERSION finseq2list


 END list2finseq
```

```
list2set[T: TYPE]: THEORY
 BEGIN

  l: VAR list[T]

  x: VAR T

  list2set(l): RECURSIVE set[T] =
    CASES l OF null: ∅[T], cons(x, y): (list2set(y) ∪ {x}) ENDCASES
      MEASURE length

  CONVERSION list2set


 END list2set
```

```
disjointness: THEORY
 BEGIN

  l: VAR list[bool]

  pairwise_disjoint?(l): RECURSIVE boolean =
    CASES l OF
      null: TRUE,
      cons(x, y): every(λ (z: bool): NOT (x AND z))(y) AND pairwise_disjoint?(y)
      ENDCASES
     MEASURE length(l)

 END disjointness
```

```
character: DATATYPE
  BEGIN
    char(code: below[256]): char?
  END character
```

strings: THEORY
BEGIN

char: TYPE = (char?)

string: TYPE = finite_sequence[char]

l_1, l_2: VAR list[char]

c_1, c_2: VAR char

fseq_lem: LEMMA
(list2finseq(l_1) = list2finseq(l_2)) = ($l_1 = l_2$)

cons_lem: LEMMA
(cons(c_1, l_1) = cons(c_2, l_2)) = ($c_1 = c_2$ & $l_1 = l_2$)

char_lem: LEMMA ($c_1 = c_2$) = (code(c_1) = code(c_2))

END strings

```
lift[T: TYPE]: DATATYPE
 BEGIN
  bottom: bottom?
  up(down: T): up?
 END lift
```

```
union[T₁, T₂: TYPE]: DATATYPE
  BEGIN
    inl(left: T₁): inl?
    inr(right: T₂): inr?
  END union
```

mucalculus$[T$: TYPE$]$: THEORY
BEGIN

s: VAR T

p, p_1, p_2: VAR pred$[T]$

predicate_transformer: TYPE = $[$pred$[T] \rightarrow$ pred$[T]]$

pt: VAR predicate_transformer

setofpred: VAR pred$[$pred$[T]]$

$\leq(p_1, p_2)$: bool = $\forall$ s: $p_1(s)$ IMPLIES $p_2(s)$

monotonic?(pt): bool = $\forall$ p_1, p_2: $p_1 \leq p_2$ IMPLIES pt(p_1) $\leq$ pt(p_2)

pp: VAR (monotonic?)

fixpoint?(pp, p): bool = (pp(p) $=$ p)

fixpoint?(pp)(p): bool = fixpoint?(pp, p)

glb(setofpred): pred$[T]$ =
λ s: ($\forall$ p: ($p \in$ setofpred) IMPLIES $p(s)$)

lfp(pp): pred$[T]$ = glb($\{p \mid$ pp(p) $\leq p\}$)

lfp_induction: FORMULA pp(p) $\leq p$ IMPLIES lfp(pp) $\leq p$

μ(pp): pred$[T]$ = lfp(pp)

lfp?(pp, p_1): bool =
fixpoint?(pp, p_1) AND $\forall$ p_2: fixpoint?(pp, p_2) IMPLIES $p_1 \leq p_2$

lfp?(pp)(p_1): bool = lfp?(pp, p_1)

lub(setofpred): pred$[T]$ =
λ s: $\exists$ p: ($p \in$ setofpred) AND $p(s)$

gfp(pp): pred$[T]$ = lub($\{p \mid p \leq$ (pp(p))$\}$)

gfp_induction: FORMULA $p \leq$ pp(p) IMPLIES $p \leq$ gfp(pp)

ν(pp): pred$[T]$ = gfp(pp)

gfp?(pp, p_1): bool =
fixpoint?(pp, p_1) AND $\forall$ p_2: fixpoint?(pp, p_2) IMPLIES $p_2 \leq p_1$

gfp?(pp)(p_1): bool = gfp?(pp, p_1)

END mucalculus

```
ctlops[state: TYPE]: THEORY
 BEGIN

  u, v, w: VAR state

  f, g, Q, P, p1, p2: VAR pred[state]

  Z: VAR pred[[state]]

  N: VAR [state, state → bool]

  CONVERSION+ K_conversion


  EX(N, f)(u): bool = (∃ v: (f(v) AND N(u, v)))

  EG(N, f): pred[state] =
      ν(λ Q: (λ (s: state): f(s) AND EX(N, Q)(s)))

  EU(N, f, g): pred[state] =
      μ(λ Q:
            (λ (s1: state):
                  g(s1) OR
                   (λ (s: state): f(s) AND EX(N, Q)(s))
                        (s1)))

  EF(N, f): pred[state] =
      EU(N, K_conversion[boolean, state](TRUE), f)

  AX(N, f): pred[state] =
      λ (s1: state): NOT EX(N, λ (s: state): NOT f(s))(s1)

  AF(N, f): pred[state] =
      λ (s1: state): NOT EG(N, λ (s: state): NOT f(s))(s1)

  AG(N, f): pred[state] =
      λ (s1: state): NOT EF(N, λ (s: state): NOT f(s))(s1)

  AU(N, f, g): pred[state] =
      λ (s3: state):
        (λ (s2: state):
            NOT EU(N, λ (s: state): NOT g(s),
```

```
                        (λ (s1: state):
                              (λ (s: state): NOT f(s))(s1) AND
                                (λ (s: state): NOT g(s))(s1)))
                     (s2))
             (s3)
         AND AF(N, g)(s3)

 CONVERSION- K_conversion


END ctlops
```

```
fairctlops[state: TYPE]: THEORY
 BEGIN

  u, v, w: VAR state

  f, g, Q, P, p1, p2: VAR pred[state]

  N: VAR [state, state -> bool]

  Ff: VAR pred[state]

  CONVERSION+ K_conversion


  fairEG(N, f)(Ff): pred[state] =
        ν(λ P:
              EU(N, f,
                    λ (s1: state):
                      f(s1) AND
                       (λ (s: state): Ff(s) AND EX(N, P)(s))
                            (s1)))

  fairAF(N, f)(Ff): pred[state] =
        λ (s1: state):
          NOT fairEG(N, λ (s: state): NOT f(s))(Ff)(s1)

  fair?(N, Ff): pred[state] = fairEG(N, λ u: TRUE)(Ff)

  fairEX(N, f)(Ff): pred[state] =
        EX(N, λ (s: state): f(s) AND fair?(N, Ff)(s))

  fairEU(N, f, g)(Ff): pred[state] =
        EU(N, f, λ (s: state): g(s) AND fair?(N, Ff)(s))

  fairEF(N, f)(Ff): pred[state] =
        EF(N, λ (s: state): f(s) AND fair?(N, Ff)(s))

  fairAX(N, f)(Ff): pred[state] =
        λ (s1: state):
          NOT fairEX(N, λ (s: state): NOT f(s))(Ff)(s1)

  fairAG(N, f)(Ff): pred[state] =
```

```
      λ (s1: state):
        NOT fairEF(N, λ (s: state): NOT f(s))(Ff)(s1)

  fairAU(N, f, g)(Ff): pred[state] =
      λ (s3: state):
        (λ (s2: state):
            NOT fairEU(N, λ (s: state): NOT g(s),
                             λ (s1: state):
                               (λ (s: state): NOT f(s))(s1) AND
                                (λ (s: state): NOT g(s))(s1))
                          (Ff)(s2))
            (s3)
          AND fairAF(N, g)(Ff)(s3)

  CONVERSION- K_conversion


END fairctlops
```

```
Fairctlops[state: TYPE]: THEORY
 BEGIN

  u, v, w: VAR state

  f, g, Q, P, p1, p2: VAR pred[state]

  N: VAR [state, state -> bool]

  Fflist, Gflist: VAR list[pred[state]]

  CONVERSION+ K_conversion


  CheckFair(Q, N, f, Fflist): RECURSIVE pred[state] =
    (CASES Fflist OF
        cons(Ff, Gflist):
          EU(N, f,
              λ (s1: state):
                f(s1) AND
                 (λ (s: state): Ff(s) AND EX(N, CheckFair(Q, N, f, Gflist))(s))
                     (s1)),
        null: Q
        ENDCASES)
     MEASURE length(Fflist)

  FairEG(N, f)(Fflist): pred[state] =
       ν(λ P: CheckFair(P, N, f, Fflist))

  FairAF(N, f)(Fflist): pred[state] =
       λ (s1: state):
         NOT FairEG(N, λ (s: state): NOT f(s))(Fflist)(s1)

  Fair?(N, Fflist): pred[state] = FairEG(N, λ u: TRUE)(Fflist)

  FairEX(N, f)(Fflist): pred[state] =
       EX(N, λ (s: state): f(s) AND Fair?(N, Fflist)(s))

  FairEU(N, f, g)(Fflist): pred[state] =
       EU(N, f, λ (s: state): g(s) AND Fair?(N, Fflist)(s))

  FairEF(N, f)(Fflist): pred[state] =
```

```
    EF(N, λ (s: state): f(s) AND Fair?(N, Fflist)(s))

  FairAX(N, f)(Fflist): pred[state] =
      λ (s1: state):
        NOT FairEX(N, λ (s: state): NOT f(s))(Fflist)(s1)

  FairAG(N, f)(Fflist): pred[state] =
      λ (s1: state):
        NOT FairEF(N, λ (s: state): NOT f(s))(Fflist)(s1)

  FairAU(N, f, g)(Fflist): pred[state] =
      λ (s3: state):
        (λ (s2: state):
            NOT FairEU(N, λ (s: state): NOT g(s),
                          λ (s1: state):
                            (λ (s: state): NOT f(s))(s1) AND
                             (λ (s: state): NOT g(s))(s1))
                       (Fflist)(s2))
            (s3)
          AND FairAF(N, g)(Fflist)(s3)

  CONVERSION- K_conversion


END Fairctlops
```

bit: THEORY
BEGIN

bit: TYPE = bool

nbit: TYPE = below(2)

b: VAR bit

bit_cases: LEMMA b = FALSE OR b = TRUE

b_0: [below(1) → bit] = (λ (i: below(1)): FALSE)

b_1: [below(1) → bit] = (λ (i: below(1)): TRUE)

b2n(b: bool): nbit = IF b THEN 1 ELSE 0 ENDIF

n2b(nb: nbit): bool = (nb = 1)

END bit

bv$[N: \mathbb{N}]$: THEORY
BEGIN

CONVERSION+ b2n

bvec: TYPE = [below(N) $\rightarrow$ bit]

b: VAR bit

bv: VAR bvec

i: VAR below$[N]$

bvec0(i): bit = FALSE

bvec1(i): bit = TRUE

fill(b)(i): bit = b;

bv^i: bit = bv(i)

CONVERSION- b2n

END bv

```
exp2: THEORY
 BEGIN

  n, m, x1, x2: VAR ℕ

  exp2(n: ℕ): RECURSIVE ℕ>0 = IF n = 0 THEN 1 ELSE 2 × exp2(n − 1) ENDIF
      MEASURE n

  JUDGEMENT exp2(n: ℕ) HAS_TYPE above(n)

  exp2_def: LEMMA exp2(n) = 2^n

  exp2_pos: LEMMA exp2(n) > 0

  exp2_n: LEMMA exp2(n + 1) = 2 × exp2(n)

  exp2_sum: LEMMA exp2(n + m) = exp2(n) × exp2(m)

  exp2_minus: LEMMA
    (∀ n, (k: upto(n)):
          exp2(n − k) = exp2(n)/exp2(k))

  exp2_strictpos: LEMMA n > 0 IMPLIES exp2(n) > 1

  exp2_lt: LEMMA n < m IMPLIES exp2(n) < exp2(m)

  exp_prop: LEMMA
    x1 < exp2(n) AND x2 < exp2(m) IMPLIES
     x1 × exp2(m) + x2 < exp2(n + m)

 END exp2
```

bv_concat_def[n: ℕ, m: ℕ]: THEORY
BEGIN

bvn : bvec[n] ∘ bvm : bvec[m]: bvec[$n + m$] =
(λ (nm: below($n + m$)):
IF nm $<$ m THEN bvm(nm) ELSE bvn(nm $-$ m) ENDIF)

END bv_concat_def

```
bv_bitwise[N: ℕ]: THEORY
 BEGIN

  i: VAR below(N)

  OR(bv1, bv2: bvec[N]): bvec[N] =
        (λ i: bv1(i) OR bv2(i));

  AND(bv1, bv2: bvec[N]): bvec[N] =
        (λ i: bv1(i) AND bv2(i));

  IFF(bv1, bv2: bvec[N]): bvec[N] =
        (λ i: bv1(i) IFF bv2(i));

  NOT(bv: bvec[N]): bvec[N] = (λ i: NOT bv(i));

  XOR(bv1, bv2: bvec[N]): bvec[N] =
        (λ i: XOR(bv1(i), bv2(i)));

  bv, bv1, bv2: VAR bvec[N]

  bv_OR: LEMMA
    (bv1 OR bv2)^i = (bv1^i OR bv2^i)

  bv_AND: LEMMA
    (bv1 AND bv2)^i = (bv1^i AND bv2^i)

  bv_IFF: LEMMA
    (bv1 IFF bv2)^i = (bv1^i IFF bv2^i)

  bv_XOR: LEMMA
    XOR(bv1, bv2)^i = XOR(bv1^i, bv2^i)

  bv_NOT: LEMMA (NOT bv)^i = NOT (bv^i)

 END bv_bitwise
```

```
bv_nat[N: ℕ]: THEORY
 BEGIN

  CONVERSION+ b2n


  bv2nat_rec(n: upto(N), bv: bvec[N]): RECURSIVE ℕ =
    IF n = 0
      THEN 0
    ELSE exp2(n − 1) × b2n(bv^(n−1)) + bv2nat_rec(n − 1, bv)
    ENDIF
     MEASURE n

  bv_lem: LEMMA
    ∀ (n: below(N), bv: bvec[N]):
      bv(n) = FALSE OR bv(n) = TRUE

  bv2nat_rec_bound: LEMMA
    ∀ (n: upto(N), bv: bvec[N]):
      bv2nat_rec(n, bv) < exp2(n)

  bv2nat(bv: bvec[N]): below(exp2(N)) = bv2nat_rec(N, bv)

  n: VAR upto(N)

  val: VAR below(exp2(N))

  bv, bv1, bv2: VAR bvec[N]

  bv2nat_inj_rec: LEMMA
    bv2nat_rec(n, bv1) = bv2nat_rec(n, bv2) ⇔
     (∀ (m: below(N)): (m < n) IMPLIES bv1(m) = bv2(m))

  bv2nat_surj_rec: LEMMA
    ∀ (y: below(exp2(n))): ∃ bv: bv2nat_rec(n, bv) = y

  bv2nat_inj: THEOREM
    (∀ (x, y: bvec[N]):
        (bv2nat(x) = bv2nat(y) IMPLIES (x = y)))

  bv2nat_surj: THEOREM
    (∀ (y: below(exp2(N))):
```

$(\exists\ (x:\ \text{bvec}[N]):\ \text{bv2nat}(x) = y))$

bv2nat_bij: THEOREM bijective?(bv2nat)

bv2nat_rec_fill_F: LEMMA bv2nat_rec(n, fill[N](FALSE)) = 0

bv2nat_rec_fill_T: LEMMA
bv2nat_rec(n, fill[N](TRUE)) = exp2(n) $-$ 1

bv2nat_fill_F: LEMMA bv2nat(fill[N](FALSE)) = 0

bv2nat_fill_T: LEMMA
bv2nat(fill[N](TRUE)) = exp2(N) $-$ 1

bv2nat_eq0: LEMMA bv2nat(bv) = 0 IMPLIES (bv = fill[N](FALSE))

bv2nat_eq_max: LEMMA
bv2nat(bv) = exp2(N) $-$ 1 IMPLIES bv = (fill[N](TRUE))

bv2nat_top_bit: THEOREM
$N > 0$ IMPLIES
IF bv2nat(bv) $<$ exp2($N-1$)
THEN bv$^{(N-1)}$ = FALSE
ELSE bv$^{(N-1)}$ = TRUE
ENDIF

bv2nat_topbit: THEOREM
$N > 0$ IMPLIES
bv$^{(N-1)}$ = (bv2nat(bv) $\geq$ exp2($N-1$))

nat2bv(val: below(exp2(N))):
{bv: bvec[N] | bv2nat(bv) = val}

nat2bv_def: LEMMA nat2bv = inverse(bv2nat)

nat2bv_bij: THEOREM
bijective?[below(exp2(N)), bvec[N]](nat2bv)

nat2bv_inv: THEOREM nat2bv(bv2nat(bv)) = bv

nat2bv_rew: LEMMA nat2bv(val) = bv IFF bv2nat(bv) = val bv2nat_inv : THEOREM bv2nat(nat2bv(val)) = val

CONVERSION- b2n

END bv_nat

empty_bv: THEORY
BEGIN

empty_bv: $[\text{below}[0] \rightarrow \text{bool}]$ = (λ (x: below[0]): TRUE);

END empty_bv

bv_caret$[N: \mathbb{N}]$: THEORY
BEGIN

bv : bvec$[N]^{\text{sp}: \ [\text{upto}(i_1)]}$: bvec$[\text{PROJ_1(sp)} - \text{PROJ_2(sp)} + 1]$ =
(λ (ii : below(PROJ_1(sp) − PROJ_2(sp) + 1)) :
bv(ii + PROJ_2(sp)));

bv : VAR bvec$[N]$

bv_caret_all : LEMMA $N > 0$ IMPLIES $\text{bv}^{(N-1, \ 0)} = \text{bv}$

bv_caret_ii_0 : LEMMA
($\forall$ (i : below(N)) :
$\text{bv}^{(i, \ i)^0} = \text{bv}^i$)

bv_caret_elim : LEMMA
($\forall$ (i : below(N), j : upto(i), k : below($i - j + 1$)) :
$\text{bv}^{(i, \ j)^k} = \text{bv}^{(j+k)}$)

END bv_caret

mod: THEORY
BEGIN

i, k, cc: VAR $\mathbb{Z}$

m: VAR $\mathbb{N}_{>0}$

n, a, b, c, x: VAR $\mathbb{N}$

j: VAR nonzero_integer

ml3: LEMMA $|i - m \times \lfloor i/m \rfloor| < m$

ml4: LEMMA $|i + m \times \lfloor -i/m \rfloor| < m$

mod(i, j): $\{k \mid |k| < |j|\}$ =
$\quad i - j \times \lfloor i/j \rfloor$

mod_pos: LEMMA mod(i, m) ≥ 0 AND mod(i, m) $< m$

JUDGEMENT mod(i: $\mathbb{Z}$, m: $\mathbb{N}_{>0}$) HAS_TYPE below(m)

mod_even: LEMMA integer_pred(i/j) IMPLIES mod(i, j) $= 0$

mod_neg: LEMMA
mod($-i$, j) $=$
IF integer_pred(i/j)
THEN 0
ELSE $j -$ mod(i, j)
ENDIF

mod_neg_d: LEMMA
mod(i, $-j$) $=$
IF integer_pred(i/j)
THEN 0
ELSE mod(i, j) $- j$
ENDIF

mod_eq_arg: LEMMA mod(j, j) $= 0$

mod_lt: LEMMA
$|i| < |j|$ IMPLIES

```
  mod(i, j) =
    IF sgn(i) = sgn(j) OR i = 0
      THEN i
    ELSE i + j
    ENDIF

mod_lt_nat: LEMMA n < m IMPLIES mod(n, m) = n

mod_lt_int: LEMMA
  -m < i AND i < m IMPLIES
   mod(i, m) = IF i ≥ 0 THEN i ELSE i + m ENDIF

mod_sum_pos: LEMMA mod(i + k × m, m) = mod(i, m)

mod_gt: LEMMA
  m ≤ i AND i < 2 × m IMPLIES mod(i, m) = i - m

mod_sum: LEMMA mod(i + k × j, j) = mod(i, j)

mod_sum_nat: LEMMA
  (∀ (n1, n2: below(m)):
      mod(n1 + n2, m) =
        IF n1 + n2 < m
          THEN n1 + n2
        ELSE n1 + n2 - m
        ENDIF)

mod_it_is: LEMMA
  a = b + m × c AND b < m IMPLIES b = mod(a, m)

mod_zero: LEMMA mod(0, j) = 0

mod_one: LEMMA
  mod(1, j) =
   IF |j| = 1
     THEN 0
   ELSIF j > 0 THEN 1
   ELSE j + 1
   ENDIF

mod_of_mod: LEMMA
  mod(i + mod(k, m), m) = mod(i + k, m)
```

mod_of_mod_neg: LEMMA
  $\text{mod}(i - \text{mod}(k,\ m),\ m) = \text{mod}(i - k,\ m)$

mod_inj_plus: LEMMA
  $a < m$ AND $n < m$ AND $c < m$ AND $\text{mod}(a + n,\ m) = \text{mod}(a + c,\ m)$ IMPLIES
   $n = c$

mod_inj_minus: LEMMA
  $a < m$ AND $n < m$ AND $c < m$ AND $\text{mod}(a - n,\ m) = \text{mod}(a - c,\ m)$ IMPLIES
   $n = c$

mod_wrap_around: LEMMA
  $n < m$ AND $(c \leq m$ AND $c \geq m - n)$ IMPLIES
   $\text{mod}(n + c,\ m) = n - (m - c)$

mod_wrap2: LEMMA $c < m$ IMPLIES $\text{mod}(m + c,\ m) = c$

mod_inj1: LEMMA
  $x < m$ AND $n < m$ AND $c < m$ AND $\text{mod}(x + n,\ m) = \text{mod}(x + c,\ m)$ IMPLIES
   $n = c$

mod_inj2: LEMMA
  $x < m$ AND $n < m$ AND $c < m$ AND $\text{mod}(x - n,\ m) = \text{mod}(x - c,\ m)$ IMPLIES
   $n = c$

mod_wrap_inj: LEMMA
  $n < m$ AND
   $a < m$ AND $b < m$ AND $a > 0$ AND $\text{mod}(n + a,\ m) = \text{mod}(n - b,\ m)$
   IMPLIES $a + b = m$

mod_wrap_inj_eq: LEMMA
  $x < m$ AND $a < m$ AND $b < m$ AND $a > 0$ IMPLIES
   $(\text{mod}(x + a,\ m) = \text{mod}(x - b,\ m)) =$
    $(a + b = m)$

kk, vv: VAR $\mathbb{N}$

mod_neg_limited: LEMMA
  $0 \leq \text{kk}$ AND $\text{kk} < m$ AND $\text{vv} < m$ AND $\text{vv} - \text{kk} < 0$ IMPLIES
   $\text{mod}(\text{vv} - \text{kk},\ m) = m + \text{vv} - \text{kk}$

odd_mod: LEMMA even?(m) $\Rightarrow$ (odd?(mod(i, m)) IFF odd?(i))

even_mod: LEMMA
even?(m) $\Rightarrow$ (even?(mod(i, m)) IFF even?(i))

mj: VAR $\mathbb{N}_{>0}$

mod_mult: LEMMA mod(mod(i, mj $\times$ m), m) $=$ mod(i, m)

END mod

```
bv_arith_nat_defs[N: ℕ]: THEORY
 BEGIN

  <(bv1, bv2: bvec[N]): bool = bv2nat(bv1) < bv2nat(bv2);

  ≤(bv1, bv2: bvec[N]): bool = bv2nat(bv1) ≤ bv2nat(bv2);

  >(bv1, bv2: bvec[N]): bool = bv2nat(bv1) > bv2nat(bv2);

  ≥(bv1, bv2: bvec[N]): bool = bv2nat(bv1) ≥ bv2nat(bv2);

  bv : bvec[N] + i : ℤ:
         {bvn: bvec[N] |
                        bv2nat(bvn) =
                          mod(bv2nat(bv) + i, exp2(N))}

  bv_plus: LEMMA
    (∀ (bv: bvec[N], i: ℤ):
         bv2nat(bv + i) =
          mod(bv2nat(bv) + i, exp2(N)));

  bv : bvec[N] − i : ℤ: bvec[N] = bv + (−i);

  bv_minus: LEMMA
    (∀ (bv: bvec[N], i: ℤ):
         bv2nat(bv − i) =
          mod(bv2nat(bv) − i, exp2(N)));

  bv1 : bvec[N] + bv2 : bvec[N]:
         {bv: bvec[N] |
                        bv2nat(bv) =
                          IF bv2nat(bv1) + bv2nat(bv2) < exp2(N)
                            THEN bv2nat(bv1) + bv2nat(bv2)
                          ELSE bv2nat(bv1) + bv2nat(bv2) − exp2(N)
                          ENDIF}

  bv1 : bvec[N] × bv2 : bvec[N]:
         {bv: bvec[2 × N] |
                        bv2nat(bv) = bv2nat(bv1) × bv2nat(bv2)};

 END bv_arith_nat_defs
```

bv_int_defs$[N: \mathbb{N}_{>0}]$: THEORY
BEGIN

minint: $\mathbb{Z}$ = $-$exp2($N-1$)

maxint: $\mathbb{Z}$ = exp2($N-1$) $-$ 1

bv_maxint_to_minint: LEMMA maxint $=$ $-$minint $-$ 1

bv_minint_to_maxint: LEMMA minint $=$ $-$maxint $-$ 1

in_rng_2s_comp(i: $\mathbb{Z}$): bool = (minint $\leq$ i AND i $\leq$ maxint)

rng_2s_comp: TYPE = $\{i$: $\mathbb{Z}$ | minint $\leq$ i AND i $\leq$ maxint$\}$

bv2int(bv: bvec$[N]$): rng_2s_comp =
IF bv2nat(bv) $<$ exp2($N-1$)
THEN bv2nat(bv)
ELSE bv2nat(bv) $-$ exp2(N)
ENDIF

int2bv(iv: rng_2s_comp): {bv: bvec$[N]$ | bv2int(bv) $=$ iv}

END bv_int_defs

```
bv_arithmetic_defs[N: ℕ>0]: THEORY
 BEGIN

  bv, bv1, bv2: VAR bvec[N]

  −bv : bvec[N]:
        {bvn: bvec[N] |
                      bv2int(bvn) =
                       IF bv2int(bv) = minint[N]
                         THEN bv2int(bv)
                       ELSE −(bv2int(bv))
                       ENDIF}

  bv1 − bv2: bvec[N] = (bv1 + (−bv2))

  overflow(bv1, bv2): bool =
      (bv2int(bv1) + bv2int(bv2)) > maxint[N] OR
       (bv2int(bv1) + bv2int(bv2)) < minint[N]

  bv_slt(bv1, bv2): bool = bv2int(bv1) < bv2int(bv2)

  bv_sle(bv1, bv2): bool = bv2int(bv1) ≤ bv2int(bv2)

  bv_sgt(bv1, bv2): bool = bv2int(bv1) > bv2int(bv2)

  bv_sge(bv1, bv2): bool = bv2int(bv1) ≥ bv2int(bv2)

  bv_splus(bv1, (bv2: bvec[N] | NOT overflow(bv1, bv2))): bvec[N] =
      int2bv(bv2int(bv1) + bv2int(bv2))

  mult_overflow(bv1, bv2): bool =
      (bv2int(bv1) × bv2int(bv2)) > maxint[N] OR
       (bv2int(bv1) × bv2int(bv2)) < minint[N]

  bv_stimes(bv1, (bv2: bvec[N] | NOT mult_overflow(bv1, bv2))): bvec[N] =
      int2bv(bv2int(bv1) × bv2int(bv2))

 END bv_arithmetic_defs
```

bv_extend_defs$[N: \mathbb{N}_{>0}]$: THEORY
BEGIN

bv: VAR bvec$[N]$

k: VAR above(N)

zero_extend(k: above(N)): [bvec$[N]$ $\rightarrow$ bvec$[k]$] =
  (λ bv: fill$[k-N]$(FALSE) $\circ$ bv)

sign_extend(k: above(N)): [bvec$[N]$ $\rightarrow$ bvec$[k]$] =
  (λ bv: fill$[k-N]$(bv$^{(N-1)}$) $\circ$ bv)

zero_extend_lsend(k: above(N)): [bvec$[N]$ $\rightarrow$ bvec$[k]$] =
  (λ bv: bv $\circ$ fill$[k-N]$(FALSE))

lsb_extend(k: above(N)): [bvec$[N]$ $\rightarrow$ bvec$[k]$] =
  (λ bv: bv $\circ$ fill$[k-N]$(bv^0))

pad_left(k: above(N), b: bit)(bv): bvec$[k]$ =
  fill$[k-N]$(b) $\circ$ bv

pad_right(k: above(N), b: bit)(bv): bvec$[k]$ =
  bv $\circ$ fill$[k-N]$(b)

END bv_extend_defs

```
infinite_sets_def[T: TYPE]: THEORY
 BEGIN

  S, R: VAR set[T]

  is_infinite(S): MACRO bool = NOT is_finite(S)

  infinite_set: TYPE = {S | NOT is_finite(S)}

  Inf: VAR infinite_set

  Fin: VAR finite_set[T]

  t: VAR T

  infinite_nonempty: JUDGEMENT infinite_set SUBTYPE_OF (nonempty?[T])

  infinite_add: JUDGEMENT add(t, Inf) HAS_TYPE infinite_set

  infinite_remove: JUDGEMENT remove(t, Inf) HAS_TYPE infinite_set

  infinite_superset: THEOREM
    ∀ Inf, S: (Inf ⊆ S) ⇒ NOT is_finite(S)

  infinite_union_left: JUDGEMENT union(Inf, S) HAS_TYPE infinite_set

  infinite_union_right: JUDGEMENT union(S, Inf) HAS_TYPE infinite_set

  infinite_union: THEOREM
    ∀ S, R:
        NOT is_finite((S ∪ R)) ⇒
          NOT is_finite(S) OR NOT is_finite(R)

  infinite_intersection: THEOREM
    ∀ S, R:
        NOT is_finite((S ∩ R)) ⇒
          NOT is_finite(S) AND NOT is_finite(R)

  infinite_difference: JUDGEMENT difference(Inf, Fin) HAS_TYPE
        infinite_set

  infinite_rest: JUDGEMENT rest(Inf) HAS_TYPE infinite_set
```

infinite_fullset : THEOREM
  $(\exists\ S:$ NOT is_finite(S)) $\Rightarrow$ NOT is_finite(fullset[T])

END infinite_sets_def

finite_sets_of_sets[T: TYPE]: THEORY
BEGIN

a: VAR set[T]

powerset_natfun_rec(A: finite_set[T], n: upto(card(A)),
f: (bijective?[(A), below(card(A))]),
B: (powerset(A))): RECURSIVE
$\mathbb{N}$ =
IF $n = 0$
THEN 0
ELSE LET nval =
exp2($n-1$) $\times$ IF (inverse(f)($n-1$) $\in B$) THEN 1 ELSE 0 ENDIF
IN nval + powerset_natfun_rec(A, $n-1$, f, B)
ENDIF
MEASURE n

powerset_natfun_rec_bound: LEMMA
$\forall$ (A: finite_set[T], n: upto(card(A)),
f: (bijective?[(A), below(card(A))]), B: (powerset(A))):
powerset_natfun_rec(A, n, f, B) $<$ exp2(n)

powerset_natfun(A: finite_set[T])(B: (powerset(A))): below(exp2(card(A))) =
LET f = choose(bijective?[(A), below(card(A))]) IN
powerset_natfun_rec(A, card(A), f, B)

powerset_natfun_inj_rec: LEMMA
$\forall$ (A: finite_set[T], n: upto(card(A)),
f: (bijective?[(A), below(card(A))]), B_1, B_2: (powerset(A))):
powerset_natfun_rec(A, n, f, B_1) = powerset_natfun_rec(A, n, f, B_2) $\Leftrightarrow$
($\forall$ (m: upto(card(A))):
($m < n$) IMPLIES
((inverse(f)(m) $\in B_1$) IFF
(inverse(f)(m) $\in B_2$)))

powerset_natfun_inj: LEMMA
$\forall$ (A: finite_set[T]):
$\forall$ (B_1, B_2: (powerset(A))):
powerset_natfun(A)(B_1) = powerset_natfun(A)(B_2) IMPLIES
$B_1 = B_2$

powerset_finite: JUDGEMENT powerset(A: finite_set[T]) HAS_TYPE finite_set[set[T]]

SS: VAR setofsets[T]

Union_finite: THEOREM
∀ SS:
is_finite(⋃SS) IFF
is_finite(SS) AND every(is_finite)(SS)

finite_Union_finite: LEMMA
is_finite(SS) AND every(is_finite[T])(SS) IMPLIES
is_finite(⋃SS)

Union_infinite: COROLLARY
∀ SS:
NOT is_finite(⋃SS) IFF
NOT is_finite(SS) OR
some(λ (S: set[T]): NOT is_finite(S))(SS)

Intersection_finite: THEOREM
∀ SS:
nonempty?(SS) AND every(is_finite)(SS) ⇒
is_finite(⋂SS)

Intersection_infinite: COROLLARY
∀ SS:
NOT is_finite(⋂SS) ⇒
every(λ (S: set[T]): NOT is_finite(S))(SS)

Complement_finite: THEOREM
∀ SS: is_finite(Complement(SS)) IFF is_finite(SS)

Complement_is_finite: JUDGEMENT Complement(SS: finite_set[set[T]]) HAS_TYPE
finite_set[set[T]]

Complement_infinite: COROLLARY
∀ SS: NOT is_finite(Complement(SS)) IFF NOT is_finite(SS)

Complement_is_infinite: JUDGEMENT Complement(SS: infinite_set[set[T]]) HAS_TYPE
infinite_set[set[T]]

END finite_sets_of_sets

```
EquivalenceClosure[T: TYPE]: THEORY
  BEGIN

    R, S: VAR PRED[[T]]

    x, y: VAR T

    EquivClos(R): equivalence[T] =
          {x, y |
                ∀ (S: equivalence[T]): (R ⊆ S) IMPLIES S(x, y)}

    EquivClosSuperset: LEMMA (R ⊆ EquivClos(R))

    EquivClosMonotone: LEMMA
      (R ⊆ S) IMPLIES (EquivClos(R) ⊆ EquivClos(S))

    EquivClosLeast: LEMMA
      equivalence?(S) AND (R ⊆ S) IMPLIES (EquivClos(R) ⊆ S)

    EquivClosIdempotent: LEMMA
      EquivClos(EquivClos(R)) = EquivClos(R)

    EquivalenceCharacterization: LEMMA
      equivalence?(S) IFF (S = EquivClos(S))

  END EquivalenceClosure
```

QuotientDefinition$[T$: TYPE$]$: THEORY
BEGIN

R: VAR set$[[T]]$

S: VAR equivalence$[T]$

x, y, z: VAR T

EquivClass(R)(x): set$[T]$ = $\{z \mid R(x, z)\}$

EquivClassNonEmpty: LEMMA nonempty?$[T]$(EquivClass(S)(x))

EquivClassEq: LEMMA
EquivClass(S)(x) = EquivClass(S)(y) IFF $S(x, y)$

repEC(S)(x): T = choose(EquivClass(S)(x))

EquivClassChoose: LEMMA $S(x$, repEC(S)(x))

ChooseEquivClassChoose: LEMMA
EquivClass(S)(repEC(S)(x)) = EquivClass(S)(x)

Quotient(S): TYPE =
$\{P$: set$[T]$ $\mid \exists\ x$: P = EquivClass(S)(x)$\}$

rep(S)(P: Quotient(S)): T = choose(P)

rep_is_repEC: LEMMA
rep(S)(EquivClass(S)(x)) = repEC(S)(x)

rep_lemma: LEMMA
EquivClass(S)(x)(rep(S)(EquivClass(S)(x)))

quotient_map(S)(x): Quotient(S) = EquivClass(S)(x)

quotient_map_surjective: LEMMA surjective?(quotient_map(S))

ECQuotient(R): TYPE = Quotient(EquivClos(R))

ECquotient_map(R)(x): ECQuotient(R) =
quotient_map(EquivClos(R))(x)

END QuotientDefinition

```
KernelDefinition[X: TYPE, X1: TYPE FROM X, Y: TYPE]: THEORY
 BEGIN

  f: VAR [X1 → Y]

  R: VAR PRED[[X]]

  x1, x2: VAR X1

  EquivalenceKernel(f): equivalence[X1] =
        {x1, x2 | f(x1) = f(x2)}

  PreservesEq(R)(f): bool =
        (restrict [[X], [X1], bool](R) ⊆ EquivalenceKernel(f))

  PreservesEqClosure: LEMMA
    PreservesEq(R) =
     PreservesEq(extend[[X], [X1], bool, FALSE]
                                (EquivClos[X1]
                                        (restrict[[X], [X1], boolean]
                                                (R))))

  PreservesEq_is_preserving: LEMMA
    PreservesEq(R) =
     preserves(restrict[[X], [X1], bool](R), =[Y])

 END KernelDefinition
```

```
QuotientKernelProperties[X: TYPE, X1: TYPE FROM X]: THEORY
 BEGIN

  S: VAR equivalence[X1]

  R: VAR PRED[[X]]

  Kernel_quotient_map: LEMMA
    EquivalenceKernel[X, X1, Quotient(S)](quotient_map(S)) = S

  PreservesEq_quotient_map: LEMMA
    PreservesEq[X, X1, Quotient(S)]
          (extend[[X], [X1], bool, FALSE](S))
          (quotient_map(S))

  quotient_map_is_Quotient_EqivalenceRespecting: JUDGEMENT quotient_map(S) HAS_TYPE
        (PreservesEq[X, X1, Quotient(S)]
                (extend[[X], [X1], bool, FALSE](S)))

  Kernel_ECquotient_map: LEMMA
    EquivalenceKernel[X, X1, ECQuotient(S)](quotient_map(S)) =
     S

  PreservesEq_ECquotient_map: LEMMA
    PreservesEq[X, X1, ECQuotient(S)]
          (extend[[X], [X1], bool, FALSE](S))
          (quotient_map(S))

  quotient_map_is_ECQuotient_EqivalenceRespecting: JUDGEMENT quotient_map(S) HAS_TYPE
        (PreservesEq[X, X1, ECQuotient(S)]
                (extend[[X], [X1], bool, FALSE](S)))

 END QuotientKernelProperties
```

```
QuotientSubDefinition[X: TYPE, X1: TYPE FROM X]: THEORY
 BEGIN

  x: VAR X1

  S: VAR
        {R: equivalence[X] |
                       PreservesEq[X, X, bool](R)(X1_pred)}

  QuotientSub(S): TYPE =
        {P: set[X] | ∃ x: P = EquivClass(S)(x)}

  quotient_sub_map(S)(x): QuotientSub(S) = EquivClass(S)(x)

 END QuotientSubDefinition
```

```
QuotientExtensionProperties[X: TYPE, X1: TYPE FROM X, Y: TYPE]: THEORY
 BEGIN

  S: VAR
         {R: equivalence[X] |
                    PreservesEq[X, X, bool](R)(X1_pred)}

  lift(S)(g: (PreservesEq[X, X1, Y](S)))(P: QuotientSub[X, X1](S)): Y =
      g(rep(S)(P))

  lift_commutation: LEMMA
    ∀ S, (g: (PreservesEq[X, X1, Y](S))):
      lift(S)(g) ∘ quotient_sub_map[X, X1](S) = g

  lift_unicity: LEMMA
    ∀ ((S: {R: equivalence[X] | PreservesEq[X, X, bool](R)(X1_pred)}
                  | PreservesEq[X, X, bool](S)(X1_pred))),
       (g: (PreservesEq[X, X1, Y](S))):
      ∀ (h: [QuotientSub[X, X1](S) → Y]):
        h ∘ quotient_sub_map[X, X1](S) = g IMPLIES
         h = lift(S)(g)

 END QuotientExtensionProperties
```

```
QuotientDistributive[X, Y: TYPE]: THEORY
 BEGIN

  S: VAR equivalence[X]

  z, w: VAR [Y]

  EqualityExtension(S): set[[[Y]]] =
      {z, w | S(z`1, w`1) AND z`2 = w`2}

  EqualityExtension_is_equivalence: JUDGEMENT EqualityExtension(S) HAS_TYPE
      equivalence[[Y]]

  EqualityExtensionPreservesEq: LEMMA
    PreservesEq[[Y], [Y], [Y]]
        (EqualityExtension(S))
        (λ (x: X, y: Y): (quotient_map(S)(x), y))

  QuotientDistributive: LEMMA
    bijective?[Quotient(EqualityExtension(S)), [Y]]
        (lift[[Y], [Y], [Y]]
              (EqualityExtension(S))
              (λ (x: X, y: Y): (quotient_map(S)(x), y)))

  R: VAR equivalence[Y]

  RelExtension(S, R): equivalence[[Y]] =
      {z, w | S(z`1, w`1) AND R(z`2, w`2)}

  RelExtensionPreservesEq: LEMMA
    PreservesEq[[Y], [Y], [Quotient(R)]]
        (RelExtension(S, R))
        (λ (x: X, y: Y):
           (quotient_map(S)(x), quotient_map(R)(y)))

  RelQuotientDistributive: LEMMA
    bijective?[Quotient(RelExtension(S, R)), [Quotient(R)]]
        (lift[[Y], [Y], [Quotient(R)]]
              (RelExtension(S, R))
              (λ (x: X, y: Y):
                 (quotient_map(S)(x),
                  quotient_map(R)(y))))
```

```
F: VAR [X → equivalence[Y]]

f, g: VAR [X → Y]

FunExtension(F): equivalence[[X → Y]] =
      {f, g | ∀ (x: X): F(x)(f(x), g(x))}

FunExtensionPreservesEq: LEMMA
   PreservesEq[[X → Y], [X → Y], [x: X → Quotient(F(x))]]
         (FunExtension(F))
         (λ (f: [X → Y]):
               λ (x: X): quotient_map(F(x))(f(x)))

FunQuotientDistributive: LEMMA
   bijective?[Quotient(FunExtension(F)), [x: X → Quotient(F(x))]]
         (lift[[X → Y], [X → Y], [x: X → Quotient(F(x))]]
                  (FunExtension(F))
                  (λ (f: [X → Y]):
                        (λ (x: X):
                              (quotient_map(F(x))(f(x))))))

END QuotientDistributive
```

```
QuotientIteration[X: TYPE]: THEORY
 BEGIN

  S: VAR equivalence[X]

  x, y: VAR X

  action(S)(R: equivalence[Quotient(S)])(x, y): bool =
        R(EquivClass(S)(x), EquivClass(S)(y))

  action_equivalence_is_equivalence: JUDGEMENT action(S)(R: equivalence[Quotient(S)])
        HAS_TYPE equivalence[X]

  QuotientAction: LEMMA
    ∀ (R: equivalence[Quotient(S)]):
        bijective?[Quotient(R), Quotient(action(S)(R))]
            (lift[Quotient(S), Quotient(S), Quotient(action(S)(R))]
                  (R)
                  (lift[X, X, Quotient(action(S)(R))]
                        (S)
                        (quotient_map[X](action(S)(R)))))

 END QuotientIteration
```

```
PartialFunctionDefinitions[X, Y: TYPE]: THEORY
 BEGIN

  SubsetPartialFunction: TYPE = [#dom: PRED[X], fun: [(dom) -> Y]#]

  LiftPartialFunction: TYPE = [X -> lift[Y]]

  f: VAR LiftPartialFunction

  g: VAR SubsetPartialFunction

  h: VAR [X -> Y]

  SPartFun_appl(g): [(dom(g)) -> Y] = g`fun

  SPartFun_to_LPartFun(g): LiftPartialFunction =
        LAMBDA (x: X):
          IF dom(g)(x) THEN up(fun(g)(x)) ELSE bottom ENDIF

  LPartFun_to_SPartFun(f): SubsetPartialFunction =
        (#dom := {x: X | up?(f(x))},
          fun := LAMBDA (y: {x: X | up?(f(x))}): down(f(y))#)

  TotalFun_to_SPartFun(h): SubsetPartialFunction =
        (#dom := {x: X | TRUE}, fun := h#)

  TotalFun_to_LPartFun(h): LiftPartialFunction =
        LAMBDA (x: X): up(h(x))

  CONVERSION SPartFun_appl


  CONVERSION SPartFun_to_LPartFun


  CONVERSION LPartFun_to_SPartFun


  CONVERSION TotalFun_to_SPartFun


  CONVERSION TotalFun_to_LPartFun
```

SPartFun_to_LPartFun_to_SPartFun : LEMMA
    LPartFun_to_SPartFun(SPartFun_to_LPartFun(g)) $=$ g

LPartFun_to_SPartFun_to_LPartFun : LEMMA
    SPartFun_to_LPartFun(LPartFun_to_SPartFun(f)) $=$ f

END PartialFunctionDefinitions

```
PartialFunctionComposition[X, Y, Z: TYPE]: THEORY
 BEGIN

  f: VAR LiftPartialFunction[X, Y]

  g: VAR LiftPartialFunction[Y, Z]

  g ∘ f: LiftPartialFunction[X, Z] =
        λ (x: X):
          CASES f(x) OF bottom: bottom, up(y): g(y) ENDCASES

  h: VAR SubsetPartialFunction[X, Y]

  k: VAR SubsetPartialFunction[Y, Z]

  CompDom(k, h): PRED[X] =
        {x: X | dom(h)(x) AND dom(k)(fun(h)(x))};

  k ∘ h: SubsetPartialFunction[X, Z] =
        (#dom := CompDom(k, h),
          fun
            := λ (x: (CompDom(k, h))):
                   fun(k)(fun(h)(x))#)

  SPartFun_to_LPartFun_CompositionPreservation: LEMMA
    SPartFun_to_LPartFun(k ∘ h) =
      SPartFun_to_LPartFun(k) ∘ SPartFun_to_LPartFun(h)

  LPartFun_to_SPartFun_CompositionPreservation: LEMMA
    LPartFun_to_SPartFun(g ∘ f) =
      LPartFun_to_SPartFun(g) ∘ LPartFun_to_SPartFun(f)

 END PartialFunctionComposition
```

```
stdlang : THEORY
 BEGIN

  void : TYPE = bool

  skip : void = TRUE

  fail : void = FALSE

  try(s1, s2 : void) : MACRO void = s1 OR s2

  try(s : void) : MACRO void = s OR skip

  ifthen(b : bool, s : void) : MACRO void = IF b THEN s ELSE skip ENDIF

  ifelse(b : bool, s : void) : MACRO void = IF b THEN skip ELSE s ENDIF

  Dummy : TYPE = bool

  dummy : MACRO Dummy = FALSE

 END stdlang
```

```
stdexc[T: TYPE+]: THEORY
 BEGIN

  ExceptionTag: TYPE = string

  Exception: TYPE = [#tag: ExceptionTag, val: T#]

  make_exc(e: ExceptionTag, t: T): Exception =
      (#tag := e, val := t#)

 END stdexc
```

```
stdcatch[T1, T2: TYPE+]: THEORY
 BEGIN

  catch_lift(tag: ExceptionTag[T2], t1: [Dummy -> T1], t2: [Exception[T2] -> T1]):
        T1

  catch(tag: ExceptionTag[T2], t1: T1, t2: [Exception[T2] -> T1]): MACRO T1 =
      catch_lift(tag, LAMBDA (d: Dummy): t1, t2)

  catch_list_lift(l: list[ExceptionTag[T2]], f1: [Dummy -> T1],
                       f2: [Exception[T2] -> T1]): RECURSIVE
        T1 =
    CASES l OF
      null: f1(FALSE),
      cons(e, r): catch_lift(e, LAMBDA (d: Dummy): catch_list_lift(r, f1, f2), f2)
      ENDCASES
     MEASURE l BY <<

  catch(l: list[ExceptionTag[T2]], t1: T1, t2: [Exception[T2] -> T1]): MACRO T1 =
      catch_list_lift(l, LAMBDA (d: Dummy): t1, t2)

  throw(tag: ExceptionTag[T2], e: Exception[T2]): T1

  throw(tag: ExceptionTag[T2], val: T2): MACRO T1 =
      throw(tag, make_exc(tag, val))

 END stdcatch
```

```
stdprog[T: TYPE+]: THEORY
 BEGIN

  prog(s: void, t: T): T = t

  error(mssg: string): T

  exit: T

  catch(tag: ExceptionTag[void], t1, t2: T): MACRO T =
      catch_lift[T, void]
          (tag, λ (d: Dummy): t1, λ (e: Exception[void]): t2)

  throw(tag: ExceptionTag[void]): MACRO T =
      throw(tag, make_exc(tag, fail))

  catch(l: list[ExceptionTag[void]], t1, t2: T): MACRO T =
      catch_list_lift[T, void]
          (l, λ (d: Dummy): t1, λ (e: Exception[void]): t2)

  UndefinedMutableVariable: ExceptionTag[void] =
      "UndefinedMutableVariable"

  Mutable: TYPE+

  ref(t: T): Mutable

  new: Mutable

  undef(v: Mutable): bool

  val_lisp(v: Mutable): T

  val(v: Mutable): T =
      IF undef(v)
        THEN throw(UndefinedMutableVariable, make_exc(UndefinedMutableVariable, fail))
      ELSE val_lisp(v)
      ENDIF

  def(v: Mutable, t: T): T = t

  set(v: Mutable, t: T): void = LET nt = def(v, t) IN skip
```

CONVERSION val

Global: TYPE+ = Mutable

loop_lift(f: [Dummy $\rightarrow$ void]): T

loop(s: void): MACRO T = loop_lift(λ (d: Dummy): s)

return(t: T): void = fail

format(s: string, t: T): string

END stdprog

stdglobal$[T$: TYPE+, t: $T]$: THEORY
BEGIN

Global: TYPE+ = Mutable$[T]$

END stdglobal

```
stdpvs[T: TYPE+]: THEORY
 BEGIN

  typeof(t: T): string

  str2pvs(s: string): T

  pvs2str_lisp(t: T): string

  pvs2str(t: T): MACRO string =
      catch_lift("cant-translate", λ (d: Dummy): pvs2str_lisp(t),
                        λ (e: Exception[void]): "<?>")

  Slisp: TYPE+

  slisp(l: list[T]): Slisp

  {||}(l: list[T]): Slisp = slisp(l)

 END stdpvs
```

stdstr: THEORY
BEGIN

NotARealNumber: ExceptionTag[string] = "NotARealNumber"

NotAnInteger: ExceptionTag[string] = "NotAnInteger"

charcode(n: ℕ): string

chartable: void

emptystr: string = ""

space: string = " "

newline: string

tab: string

doublequote: string = charcode(34)

singlequote: string = "'"

backquote: string = "`"

spaces(n: ℕ): string

upcase(s: string): string

downcase(s: string): string

capitalize(s: string): string

strfind(s_1, s_2: string): ℤ

substr(s: string, i, j: ℕ): string

real2str(r: ℝ): string

bool2str(b: bool): string = IF b THEN "TRUE" ELSE "FALSE" ENDIF

tostr(r: ℝ): MACRO string = real2str(r)

tostr(b: bool): MACRO string = bool2str(b)

str2real(s: string): $\mathbb{Q}$

str2int(s: string): $\mathbb{Z}$

str2bool(s, answer: string): bool =
downcase(s) = downcase(answer)

number?(s: string): bool

int?(s: string): bool

concat(s_1, s_2: string): string = $s_1 \circ s_2$;

s_1 : string + s_2 : string: MACRO string = concat(s_1, s_2);

r : $\mathbb{R}$ + s : string: MACRO string = concat(real2str(r), s);

s : string + r : $\mathbb{R}$: MACRO string = concat(s, real2str(r));

b : bool + s : string: MACRO string =
concat(bool2str(b), s);

s : string + b : bool: MACRO string =
concat(s, bool2str(b))

pad(n: $\mathbb{N}$, s: string): RECURSIVE string =
IF $n = 0$ THEN emptystr ELSE concat(s, pad($n - 1$, s)) ENDIF
MEASURE n

strcmp(s_1, s_2: string, sensitive: bool): $\mathbb{Z}$

strcmp(s_1, s_2: string): MACRO $\mathbb{Z}$ = strcmp(s_1, s_2, TRUE)

strtrim(s_1, s_2: string): string

strtrim_left(s_1, s_2: string): string

strtrim_right(s_1, s_2: string): string trim(s : string) : string

trim_left(s : string) : string

trim_right(s : string) : string

filename(s : string) : string

directory(s : string) : string

END stdstr

```
stdio : THEORY
  BEGIN

    FileNotFound : ExceptionTag[string] = "FileNotFound"

    FileAlreadyExists : ExceptionTag[string] = "FileAlreadyExists"

    WrongInputStreamMode : ExceptionTag[string] = "WrongInputStreamMode"

    WrongOutputStreamMode : ExceptionTag[string] = "WrongOutputStreamMode"

    ClosedStream : ExceptionTag[string] = "ClosedStream"

    EndOfFile : ExceptionTag[string] = "EndOfFile"

    IOExceptionTags : list[ExceptionTag[string]] =
        (:NotARealNumber, NotAnInteger, FileNotFound, FileAlreadyExists, WrongInput-
StreamMode,
          WrongOutputStreamMode, ClosedStream, EndOfFile:)

    assert(b : bool, str : string) : void =
        b OR error[void](concat("[Assertion Failure] ", str)) & fail

    break : MACRO void = return(skip)

    while(b : bool, s : void) : MACRO void =
        loop_lift(λ (d : Dummy) :
                    IF b THEN s ELSE return(skip) ENDIF)

    for(si, b, sinc, s : void) : MACRO void =
        si &
         loop_lift(λ (d : Dummy) :
                    IF b THEN s & sinc ELSE return(skip) ENDIF)

    printstr(s : string) : void = skip

    print(s : string) : MACRO void = printstr(s)

    print(r : ℝ) : MACRO void =
        printstr(concat(real2str(r), emptystr))

    print(b : bool) : MACRO void =
```

```
    printstr(concat(bool2str(b), emptystr))

println(s: string): MACRO void = printstr(concat(s, newline))

println(r: ℝ): MACRO void =
    printstr(concat(real2str(r), newline))

println(b: bool): MACRO void =
    printstr(concat(bool2str(b), newline))

query_token(mssg, s: string): string

query_word(mssg: string): MACRO string = query_token(mssg, emptystr)

query_line(mssg: string): string

query_real(mssg: string): ℚ

query_int(mssg: string): ℤ

query_bool(mssg, answer: string): bool =
    str2bool(query_token(mssg, emptystr), answer)

read_token(s: string): MACRO string = query_token(emptystr, s)

read_word: MACRO string = query_token(emptystr, emptystr)

read_line: MACRO string = query_line(emptystr)

read_real: MACRO ℚ = query_real(emptystr)

read_int: MACRO ℤ = query_int(emptystr)

read_bool(answer: string): MACRO bool = query_bool(emptystr, answer)

Stream: TYPE+

IStream: TYPE+ FROM Stream

OStream: TYPE+ FROM Stream

fclose(f: Stream): void = skip
```

```
fexists(s: string): bool

fopen?(f: Stream): bool

strstream?(f: Stream): bool

filestream?(f: Stream): bool

sdtstream?(f: Stream): bool =
    NOT (filestream?(f) OR strstream?(f))

finput?(f: Stream): bool

foutput?(f: Stream): bool

stdin: IStream

stdout: OStream

stderr: OStream

Mode: TYPE = {input, output, create, append, overwrite, rename, str}

mode2str(m: Mode): string =
    CASES m OF
      input: "input",
      output: "output",
      create: "create",
      append: "append",
      overwrite: "overwrite",
      rename: "rename",
      str: "str"
      ENDCASES

tostr(m: Mode): MACRO string = mode2str(m)

fopenin_lisp(s: string): IStream

fopenout_lisp(s: string, n: ℕ): OStream

sopenin(s: string): IStream
```

```
sopenout(s: string): OStream

fopenin(m: Mode, s: string): IStream =
    IF m = input AND length(s) = 0
      THEN stdin
    ELSIF m = input THEN fopenin_lisp(s)
    ELSIF m = str THEN sopenin(s)
    ELSE throw(WrongInputStreamMode, make_exc(WrongInputStreamMode, mode2str(m)))
    ENDIF

fopenin(s: string): IStream = fopenin_lisp(s)

fopenout(m: Mode, s: string): OStream =
    IF m = output AND length(s) = 0
      THEN stdout
    ELSIF m = output THEN fopenout_lisp(s, 0)
    ELSIF m = create THEN fopenout_lisp(s, 1)
    ELSIF m = append THEN fopenout_lisp(s, 2)
    ELSIF m = overwrite THEN fopenout_lisp(s, 3)
    ELSIF m = rename THEN fopenout_lisp(s, 4)
    ELSIF m = str THEN sopenout(s)
    ELSE throw(WrongOutputStreamMode, make_exc(WrongOutputStreamMode, mode2str(m)))
    ENDIF

fopenout(s: string): OStream = fopenout(output, s)

fname_lisp(f: Stream): string

fgetstr_lisp(f: OStream): string

eof_lisp(f: IStream): bool

flength_lisp(f: Stream): ℕ

fgetpos_lisp(f: Stream): ℕ

fsetpos_lisp(f: Stream, n: ℕ): void

fprint_lisp(f: OStream, s: string): void = skip

fname(f: Stream): string =
```

```
    IF filestream?(f) THEN fname_lisp(f) ELSE emptystr ENDIF

fgetstr(f: OStream): string =
    IF fopen?(f)
      THEN fgetstr_lisp(f)
    ELSE throw(ClosedStream, make_exc(ClosedStream, fname(f)))
    ENDIF

eof?(f: IStream): bool =
    IF fopen?(f)
      THEN eof_lisp(f)
    ELSE throw(ClosedStream, make_exc(ClosedStream, fname(f)))
    ENDIF

flength(f: Stream): ℕ =
    IF fopen?(f)
      THEN flength_lisp(f)
    ELSE throw(ClosedStream, make_exc(ClosedStream, fname(f)))
    ENDIF

fgetpos(f: Stream): ℕ =
    IF fopen?(f)
      THEN fgetpos_lisp(f)
    ELSE throw(ClosedStream, make_exc(ClosedStream, fname(f)))
    ENDIF

fprint(f: OStream, s: string): void =
    IF fopen?(f)
      THEN fprint_lisp(f, s)
    ELSE throw(ClosedStream, make_exc(ClosedStream, fname(f)))
    ENDIF

fprint(f: OStream, r: ℝ): MACRO void =
    fprint(f, concat(real2str(r), emptystr))

fprint(f: OStream, b: bool): MACRO void =
    fprint(f, concat(bool2str(b), emptystr))

fsetpos(f: Stream, n: ℕ): void =
    IF fopen?(f)
      THEN fsetpos_lisp(f, n)
    ELSE throw(ClosedStream, make_exc(ClosedStream, fname(f)))
```

```
    ENDIF

fprintln(f: OStream, s: string): MACRO void =
    fprint(f, concat(s, newline))

fprintln(f: OStream, r: ℝ): MACRO void =
    fprint(f, concat(real2str(r), newline))

fprintln(f: OStream, b: bool): MACRO void =
    fprint(f, concat(bool2str(b), newline))

echo(f: OStream, s: string): MACRO void =
    printstr(s) & fprint(f, s)

echo(f: OStream, r: ℝ): MACRO void =
    printstr(concat(real2str(r), emptystr)) &
     fprint(f, concat(real2str(r), emptystr))

echo(f: OStream, b: bool): MACRO void =
    printstr(concat(bool2str(b), emptystr)) &
     fprint(f, concat(bool2str(b), emptystr))

echoln(f: OStream, s: string): MACRO void =
    printstr(concat(s, newline)) &
     fprint(f, concat(s, newline))

echoln(f: OStream, r: ℝ): MACRO void =
    printstr(concat(real2str(r), newline)) &
     fprint(f, concat(real2str(r), newline))

echoln(f: OStream, b: bool): MACRO void =
    printstr(concat(bool2str(b), newline)) &
     fprint(f, concat(bool2str(b), newline))

fread_token_lisp(f: IStream, s: string): string

fread_line_lisp(f: IStream): string

fread_real_lisp(f: IStream): ℚ

fread_int_lisp(f: IStream): ℤ
```

```
  fcheck(f: IStream): bool =
      (fopen?(f) OR throw(ClosedStream, make_exc(ClosedStream, fname(f)))) AND
       (NOT eof?(f) OR
           throw(EndOfFile, make_exc(EndOfFile, fname(f))))

  fread_token(f: IStream, s: string): string =
      prog(fcheck(f), fread_token_lisp(f, s))

  fread_word(f: IStream): MACRO string = fread_token(f, emptystr)

  fread_line(f: IStream): string =
      prog(fcheck(f), fread_line_lisp(f))

  fread_real(f: IStream): ℚ =
      prog(fcheck(f), fread_real_lisp(f))

  fread_int(f: IStream): ℤ =
      prog(fcheck(f), fread_int_lisp(f))

  fread_bool(f: IStream, answer: string): MACRO bool =
      str2bool(fread_token(f, emptystr), answer)

END stdio
```

```
stdmath: THEORY
  BEGIN

    MathExceptions: list[ExceptionTag[string]] =
        (:NotARealNumber, NotAnInteger:)

    Π: ℝ>0

    SIN(x: ℝ): {x: ℝ | −1 ≤ x AND x ≤ 1}

    COS(y: ℝ): {x: ℝ | −1 ≤ x AND x ≤ 1}

    EXP(x: ℝ): ℝ>0

    RANDOM: {y: ℝ≥0 | 0 ≤ y AND y ≤ 1}

    NRANDOM(n: ℕ>0): {y: ℕ | 0 ≤ y AND y < n}

    sqrt_lisp(x: ℝ≥0): ℝ≥0

    log_lisp(x: ℝ>0): ℝ

    atan_lisp(x, y: ℝ): ℝ

    asin_lisp(x: ℝ): ℝ

    acos_lisp(x: ℝ): ℝ

    SQRT(x: ℝ): ℝ≥0 =
        IF x < 0
          THEN throw(NotARealNumber,
                    make_exc(NotARealNumber, concat(concat("SQRT(", real2str(x)), ")")))
        ELSE sqrt_lisp(x)
        ENDIF

    LOG(x: ℝ): ℝ =
        IF x ≤ 0
          THEN throw(NotARealNumber,
                    make_exc(NotARealNumber, concat(concat("LOG(", real2str(x)), ")")))
        ELSE log_lisp(x)
        ENDIF
```

```
TAN(x: ℝ): ℝ =
    LET d = COS(x) IN
      IF d = 0
        THEN throw(NotARealNumber,
                   make_exc(NotARealNumber,
                            concat(concat("TAN(", real2str(x)), "0)")))
      ELSE SIN(x)/COS(x)
      ENDIF

ATAN(y, x: ℝ): ℝ =
    IF x = 0 AND y = 0
      THEN throw(NotARealNumber, make_exc(NotARealNumber, "ATAN(0,0)"))
    ELSE atan_lisp(y, x)
    ENDIF

ASIN(x: ℝ): ℝ =
    IF x < −1 OR x > 1
      THEN throw(NotARealNumber,
                 make_exc(NotARealNumber, concat(concat("ASIN(", real2str(x)), ")")))
    ELSE asin_lisp(x)
    ENDIF

ACOS(x: ℝ): ℝ =
    IF x < −1 OR x > 1
      THEN throw(NotARealNumber,
                 make_exc(NotARealNumber, concat(concat("ACOS(", real2str(x)), ")")))
    ELSE acos_lisp(x)
    ENDIF

BRANDOM: bool = (NRANDOM(2) = 0)

END stdmath
```

```
stdfmap[T: TYPE+]: THEORY
 BEGIN

  fmap(f: IStream, fread: [IStream → string], t: T, st: [[T] → T],
        n: ℕ): RECURSIVE
        T =
    IF fopen?(f)
      THEN IF n = 0 OR eof?(f)
             THEN t
           ELSE LET s = fread(f) IN
                  LET nt = st(s, t) IN fmap(f, fread, nt, st, n − 1)
           ENDIF
    ELSE throw(ClosedStream, make_exc(ClosedStream, fname(f)))
    ENDIF
     MEASURE n

  fmap(f: IStream, fread: [IStream → string], t: T, st: [[T] → T]): T =
      LET l = flength(f) IN
        LET nt = fmap(f, fread, t, st, l) IN prog(fclose(f), nt)

  fmap_line(f: IStream, t: T, st: [[T] → T]): T =
      LET l = flength(f) IN
        LET nt = fmap(f, fread_line, t, st, l) IN
          prog(fclose(f), nt)

  printf(s: string, t: T): MACRO void = printstr(format(s, t))

  fprintf(f: OStream, s: string, t: T): MACRO void =
      fprint(f, format(s, t))

 END stdfmap
```

```
stdindent : THEORY
 BEGIN

  Indent : TYPE+

  create_indent(n : ℕ, s : string) : Indent

  push_indent(i : Indent, n : ℕ) : void

  pop_indent(i : Indent) : void

  top_indent(i : Indent) : ℕ

  get_indent(i : Indent) : ℕ

  set_indent(i : Indent, n : ℕ) : void

  get_prefix(i : Indent) : string

  set_prefix(i : Indent, s : string) : void

  create_indent(n : ℕ) : MACRO Indent = create_indent(n, emptystr)

  open_block(i : Indent, n : ℕ) : MACRO void = push_indent(i, n)

  open_block(i : Indent) : MACRO void =
      push_indent(i, get_indent(i))

  close_block(i : Indent) : MACRO void = pop_indent(i)

  indent(i : Indent) : string =
      concat(get_prefix(i), spaces(top_indent(i)))

  indent(i : Indent, s : string) : string = concat(indent(i), s)

  prindent(i : Indent, s : string) : void = printstr(indent(i, s))

  prindentln(i : Indent, s : string) : void =
      printstr(concat(indent(i, s), newline))

  fprindent(f : OStream, i : Indent, s : string) : void =
      fprint(f, indent(i, s))
```

```
  fprindentln(f: OStream, i: Indent, s: string): void =
      fprint(f, concat(indent(i, s), newline))

  center(col: ℕ, s: string): string =
      format(concat(concat("~", real2str(col)), ":@<~a~>"), s)

  flushleft(col: ℕ, s: string): string =
      format(concat(concat("~", real2str(col)), "a"), s)

  flushright(col: ℕ, s: string): string =
      format(concat(concat("~", real2str(col)), "@a"), s)

END stdindent
```

```
stdtokenizer: THEORY
 BEGIN

  NoError: ℕ = 0

  FileNotFound: ℕ = 1

  EndOfTokenizer: ℕ = 2

  InvalidToken: ℕ = 3

  ExpectingWord: ℕ = 4

  ExpectingTestWord: ℕ = 5

  ExpectingInt: ℕ = 6

  ExpectingTestInt: ℕ = 7

  ExpectingReal: ℕ = 8

  ExpectingTestReal: ℕ = 9

  Tokenizer: TYPE =
          [#stream: [ℕ → string],
             lines: [ℕ → ℕ],
             casesen: boolean,
             separ: string,
             error: ℤ,
             val_int: ℤ,
             val_real: ℝ,
             length: ℕ,
             pos: upto(length)#]

  init_tokenizer(casesen: bool, separ: string): Tokenizer =
      (#stream := λ (x: ℕ): emptystr,
         lines := λ (x: ℕ): 0,
         casesen := casesen,
         separ := separ,
         error := NoError,
         val_int := 0,
         val_real := 0,
```

```
      length := 0,
      pos := 0#)

empty_tokenizer: Tokenizer =
    (#stream := λ (x: ℕ): emptystr,
      lines := λ (x: ℕ): 0,
      casesen := TRUE,
      separ := emptystr,
      error := NoError,
      val_int := 0,
      val_real := 0,
      length := 0,
      pos := 0#)

TokenizerOfLength(l: ℤ): TYPE = {t: Tokenizer | t`length = l}

set_casesen(t: Tokenizer, c: bool): Tokenizer =
    t WITH [`casesen := c]

error?(t: Tokenizer): MACRO bool = t`error ≠ NoError

set_error(t: Tokenizer, code: ℤ): TokenizerOfLength(t`length) =
    t WITH [`error := code]

last_token(t: Tokenizer): string =
    IF t`pos = 0
      THEN emptystr
    ELSE t`stream(t`pos − 1)
    ENDIF

peek(t: Tokenizer, n: ℕ>0): MACRO string =
    t`stream(t`pos + n − 1)

next_token(t: Tokenizer): MACRO string =
    t`stream(t`pos + 1 − 1)

tostr(t: Tokenizer, i: upto(t`length)): RECURSIVE string =
  IF i = t`length
    THEN emptystr
  ELSIF i = t`pos
    THEN concat(concat(concat("[", t`stream(i)), "] "), tostr(t, i + 1))
  ELSE concat(concat(t`stream(i), space), tostr(t, i + 1))
```

```
  ENDIF
   MEASURE  t`length − i

add_token(s: string, t: Tokenizer, l: ℕ): MACRO Tokenizer =
    LET n = t`length IN
      t
        WITH [`stream(n) := s,
              `lines(n) := l,
              `length := n + 1]

read_token(t: Tokenizer)(f: IStream): string =
    fread_token(f, t`separ)

line_tokenizer(s: string, tl: [ℕ]): [ℕ] =
    LET (t, l) = tl IN
      LET g = fopenin(str, s) IN
        LET f =
              (λ (mys: string, myt: Tokenizer):
                  LET n = myt`length IN
                    myt
                      WITH [`stream(n) := mys,
                            `lines(n) := l,
                            `length := n + 1])
          IN
          LET nt = fmap(g, read_token(t), t, f, length(s)) IN
            prog(fclose(g), (nt, l + 1))

file2tokenizer(s: string, t: Tokenizer): Tokenizer =
    IF fexists(s)
      THEN LET (nt, l) = fmap_line(fopenin(s), (t, 1), line_tokenizer) IN nt
    ELSE LET filename = s IN t WITH [`stream(0) := filename, `error := FileNotFound]
    ENDIF

file2tokenizer(s: string): Tokenizer =
    file2tokenizer(s, empty_tokenizer)

str2tokenizer(s: string, t: Tokenizer): Tokenizer =
    LET f = fopenin(str, s) IN
      LET (nt, l) = fmap_line(f, (t, 1), line_tokenizer) IN nt

str2tokenizer(s: string): Tokenizer =
    str2tokenizer(s, empty_tokenizer)
```

```
eot?(t: Tokenizer): bool = t`pos = t`length

get_line(t: Tokenizer): MACRO ℕ = t`lines(t`pos)

consume(t: Tokenizer, n: ℕ>0): TokenizerOfLength(t`length) =
    IF t`error ≠ NoError
      THEN t
    ELSIF eot?(t) OR t`pos + n > t`length THEN t WITH [`error := EndOfTokenizer]
    ELSE t WITH [`pos := t`pos + n]
    ENDIF

go_next(t: Tokenizer): MACRO TokenizerOfLength(t`length) =
    consume(t, 1)

go_back(t: Tokenizer): TokenizerOfLength(t`length) =
    IF t`pos = 0
      THEN t
    ELSE t WITH [`pos := t`pos − 1]
    ENDIF

pos_go_next: LEMMA
  ∀ (t1: Tokenizer):
    LET t2 = consume(t1, 1) IN
      NOT t2`error ≠ NoError IMPLIES t2`pos = t1`pos + 1

accept_word(t: Tokenizer, test: [string → bool]): TokenizerOfLength(t`length) =
    IF t`error ≠ NoError
      THEN t
    ELSIF eot?(t) THEN t WITH [`error := EndOfTokenizer]
    ELSIF number?(t`stream(t`pos)) THEN t WITH [`error := ExpectingWord]
    ELSIF test(t`stream(t`pos)) THEN t WITH [`pos := t`pos + 1]
    ELSE t WITH [`error := ExpectingTestWord]
    ENDIF

pos_accept_word: LEMMA
  ∀ (t1: Tokenizer, test: [string → bool]):
    LET t2 = accept_word(t1, test) IN
      NOT t2`error ≠ NoError IMPLIES t2`pos = t1`pos + 1

the_word(s: string)(token: string): bool = str2bool(s, token)
```

```
accept_word(t: Tokenizer, s: string): MACRO TokenizerOfLength(t`length) =
    accept_word(t, the_word(s))

any_word(token: string): bool = TRUE

accept_word(t: Tokenizer): MACRO TokenizerOfLength(t`length) =
    accept_word(t, any_word)

accept_int(t: Tokenizer, test: [ℤ → bool]): TokenizerOfLength(t`length) =
    IF t`error ≠ NoError
      THEN t
    ELSIF eot?(t) THEN t WITH [`error := EndOfTokenizer]
    ELSIF NOT int?(t`stream(t`pos)) THEN t WITH [`error := ExpectingInt]
    ELSE LET i = str2int(t`stream(t`pos)) IN
           IF test(i)
             THEN t WITH [`pos := t`pos + 1, `val_int := i, `val_real := i]
           ELSE t WITH [`error := ExpectingTestInt]
           ENDIF
    ENDIF

pos_accept_int: LEMMA
  ∀ (t1: Tokenizer, test: [ℤ → bool]):
    LET t2 = accept_int(t1, test) IN
      NOT t2`error ≠ NoError IMPLIES t2`pos = t1`pos + 1

any_int(i: ℤ): bool = TRUE

accept_int(t: Tokenizer): MACRO TokenizerOfLength(t`length) =
    accept_int(t, any_int)

accept_real(t: Tokenizer, test: [ℝ → bool]): TokenizerOfLength(t`length) =
    IF t`error ≠ NoError
      THEN t
    ELSIF eot?(t) THEN t WITH [`error := EndOfTokenizer]
    ELSIF NOT number?(t`stream(t`pos)) THEN t WITH [`error := ExpectingReal]
    ELSE LET r = str2real(t`stream(t`pos)) IN
           IF test(r)
             THEN t WITH [`pos := t`pos + 1, `val_real := r]
           ELSE t WITH [`error := ExpectingTestReal]
           ENDIF
    ENDIF
```

```
pos_accept_real: LEMMA
  ∀ (t1: Tokenizer, test: [ℝ → bool]):
    LET t2 = accept_real(t1, test) IN
      NOT t2`error ≠ NoError IMPLIES t2`pos = t1`pos + 1

any_real(r: ℝ): bool = TRUE

accept_real(t: Tokenizer): MACRO TokenizerOfLength(t`length) =
    accept_real(t, any_real)

Messenger: TYPE = [ℤ → string]

std_mssg(t: Tokenizer)(code: ℤ): string =
    IF code ≤ 0
      THEN emptystr
    ELSIF code = FileNotFound
      THEN concat(concat(concat(concat("File not found: ", doublequote),
                                       t`stream(t`pos + 1 − 1)),
                                doublequote),
                         ".")
    ELSIF code = EndOfTokenizer THEN "Found EOT."
    ELSIF code = InvalidToken
      THEN concat(concat(concat(concat("Invalid Token: ", doublequote),
                                       t`stream(t`pos + 1 − 1)),
                                doublequote),
                         ".")
    ELSIF code = ExpectingWord
      THEN concat(concat(concat(concat("Expecting a word. Found: ", double-
quote),
                                       t`stream(t`pos + 1 − 1)),
                                doublequote),
                         ".")
    ELSIF code = ExpectingTestWord
      THEN concat(concat(concat(concat("Expecting word that satisfies test. Found:
                                                doublequote),
                                       t`stream(t`pos + 1 − 1)),
                                doublequote),
                         ".")
    ELSIF code = ExpectingInt
      THEN concat(concat(concat(concat("Expecting an integer. Found: ", dou-
blequote),
                                       t`stream(t`pos + 1 − 1)),
```

```
                                        doublequote),
                                "."")
      ELSIF code = ExpectingTestInt
        THEN concat(concat(concat(concat("Expecting integer that satisfies test. Foun
                                                          doublequote),
                                                t`stream(t`pos + 1 − 1)),
                                        doublequote),
                                ".")
      ELSIF code = ExpectingReal
        THEN concat(concat(concat(concat("Expecting a real. Found: ", double-
quote),
                                                t`stream(t`pos + 1 − 1)),
                                        doublequote),
                                ".")
      ELSIF code = ExpectingTestReal
        THEN concat(concat(concat(concat("Expecting real that satisfies test. Found:
                                                          doublequote),
                                                t`stream(t`pos + 1 − 1)),
                                        doublequote),
                                ".")
      ELSE emptystr
      ENDIF

  print_error(t: Tokenizer, m: Messenger): MACRO void =
      IF t`error ≠ NoError
        THEN printstr(concat(concat(concat(concat(concat("Syntax Error. ", "Line "),
                                                                  real2str(t`lines(t`pos))),
                                                          ": "),
                                                  m(error(t))),
                                        newline))
                & fail
      ELSE skip
      ENDIF

  print_error(t: Tokenizer): MACRO void =
      IF t`error ≠ NoError
        THEN printstr(concat(concat(concat(concat(concat("Syntax Error. ", "Line "),
                                                                  real2str(t`lines(t`pos))),
                                                          ": "),
                                                  std_mssg(t)(error(t))),
                                        newline))
                & fail
```

```
      ELSE skip
      ENDIF;

  m :  Messenger × cs :  [string]:  Messenger  =
      LET  (code,  s)  =  cs  IN  m  WITH  [`code  :=  s]

  tokenizer2str(t:  Tokenizer):  string  =
      concat(concat(tostr(t,  0),  newline),
                  std_mssg(t)(t`error))

  CONVERSION  tokenizer2str


  tostr(t:  Tokenizer):  MACRO  string  =  tokenizer2str(t)

END  stdtokenizer
```

stdpvsio : THEORY
BEGIN

help_pvs_attachment(s : string) : void

help_pvs_theory_attachments(s : string) : void

pvsio_version : string

set_promptin(s : string) : void

set_promptout(s : string) : void

END stdpvsio

```
stdsys: THEORY
 BEGIN

  Time: TYPE =
         [#second: below(60),
           minute: below(60),
           hour: below(24),
           day: subrange(1, 31),
           month: subrange(1, 12),
           year: ℕ,
           dow: below(7),
           dst: bool,
           tz: {x: ℚ | −24 ≤ x AND x ≤ 24}#]

  tinybang: Time =
      (#day := 1,
        dow := 0,
        dst := FALSE,
        hour := 0,
        minute := 0,
        month := 1,
        second := 0,
        tz := 0,
        year := 0#)

  days_of_week(dow: below(7)): string =
      IF dow = 0
        THEN "Monday"
      ELSIF dow = 1 THEN "Tuesday"
      ELSIF dow = 2 THEN "Wednesday"
      ELSIF dow = 3 THEN "Thursday"
      ELSIF dow = 4 THEN "Friday"
      ELSIF dow = 5 THEN "Saturday"
      ELSE "Sunday"
      ENDIF

  months(month: subrange(1, 12)): string =
      IF month = 1
        THEN "January"
      ELSIF month = 2 THEN "February"
      ELSIF month = 3 THEN "March"
      ELSIF month = 4 THEN "April"
```

```
      ELSIF month = 5 THEN "May"
      ELSIF month = 6 THEN "June"
      ELSIF month = 7 THEN "July"
      ELSIF month = 8 THEN "August"
      ELSIF month = 9 THEN "September"
      ELSIF month = 10 THEN "October"
      ELSIF month = 11 THEN "November"
      ELSE "December"
      ENDIF

tostr(t: Time): string =
      concat(concat(concat(concat(concat(concat(days_of_week(dow(t)), " "),
                                                                  months(month(t))),
                                                      format(" ~2,'0d ", day(t))),
                                         real2str(year(t))),
                            format(", ~2,'0d:~2,'0d:~2,'0d ",
                                         (hour(t), minute(t), second(t)))),
                 format("(GMT~@d)", -tz(t)))

get_time: Time

today: string =
      LET t = get_time IN
         format("~d/~2,'0d/~d", (month(t), day(t), year(t)))

date: string = LET t = get_time IN tostr(get_time)

sleep(n: ℕ): void

get_env(name, default: string): string

get_env(name: string): MACRO string = get_env(name, emptystr)

printf(s: string): MACRO void = printstr(format(s, ""))

fprintf(f: OStream, s: string): MACRO void =
      fprint(f, format(s, ""))

Blisp: TYPE+

blisp(b: bool): Blisp
```

{||}(b: bool): Blisp = blisp(b)

END stdsys