Miscellanea

Roger Bishop Jones

Abstract

This document contains things used by my other theories which do not particularly belong in them. Definitions or theorems which arguably belong in a theory already produced by someone else.

Date Created 2004/07/15

Last Changed Date: 2013/01/03 17:12:44

http://www.rbjones.com/rbjpub/pp/doc/t006.pdf

Id: t006.doc,v 1.44 2013/01/03 17:12:44 rbj Exp

Contents

References

[1] Alonzo Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–, 1940.

[2] Roger Bishop Jones. Introduction to Work in Progress. *RBJones.com*, 2010. `http://www.rbjones.com/rbjpub/pp/doc/t000.pdf`.

[3] B. Russell. Mathematical Logic as based on the Theory of Types. *American Journal of Mathematics*, 30:222–262, 1908.

[4] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1910-1913.

1 The Theory rbjmisc

For context and motivation see [2].

All the theorems are placed in the one temporary location, which therefore must have as ancestors all the theories which are being extended.

There is one section in this document, following this one, for each theory to which an addition is being made so new parents may be added in those sections, or new proof contexts used.

SML

```
open PreConsisProof;
```

SML

```
open_theory "cache'rbjhol";
force_new_theory "rbjmisc";
val _ =     let open ReaderWriterSupport.PrettyNames;
    in add_new_symbols [ (["sqsubseteq2"], Value "⊑", Simple) ]
        end
handle _ => ();
new_parent "analysis";
new_parent "equiv_rel";
force_new_pc "'rbjmisc";
merge_pcs ["'prove_∃_⇒_conv", "'savedthm_cs_∃_proof"] "'rbjmisc";
set_merge_pcs ["basic_hol1", "'sets_alg", "'ℝ", "'rbjmisc"];
open UnifyForwardChain; open RbjTactics1;
```

2 Combinators

HOL Constant

```
$CombC: ('a → 'b → 'c) → ('b → 'a → 'c)
```

```
∀f• CombC f = λx y• f y x
```

```
combc_thm = ⊢ ∀ f x y• CombC f x y = f y x
```

HOL Constant

```
BinComp : ('a → 'b → 'c) → ('d → 'a) → ('e → 'b) → ('d → 'e → 'c)
```

```
∀ f g h• BinComp f g h = λx y• f (g x) (h y)
```

```
combc_thm = ⊢ ∀ f x y• CombC f x y = f y x
```

3 Predicate Calculus

There is probably a better way of doing this (or a better thing to be doing).

In some circumstances ∀_ ∧ _*out_lemma* can be used to avoid or postpone a case split.

```
∀_η_lemma =
    ⊢ ∀ p• $∀ p ⇔ (∀ x• p x)

∀_∧_out_lemma =
    ⊢ ∀ p q• $∀ p ∧ $∀ q ⇔ (∀ x• p x ∧ q x)
```

3.0.1 ManyOne

The relations used in replacement must be "ManyOne" relations, otherwise the image may be larger than the domain, and Russell's paradox would reappear.

HOL Constant

$$\boldsymbol{ManyOne} : ('a \to 'b \to BOOL) \to BOOL$$

$$\forall r\bullet\ ManyOne\ r \Leftrightarrow \forall x\ y\ z\bullet\ r\ x\ y \wedge r\ x\ z \Rightarrow y = z$$

4 Type Definition Lemmas

$$\boldsymbol{type\_lemmas\_thm2} =$$
$$\vdash \forall\ pred$$
$$\bullet\ (\exists\ f\bullet\ TypeDefn\ pred\ f)$$
$$\Rightarrow (\exists\ abs\ rep$$
$$\bullet\ (\forall\ a\bullet\ abs\ (rep\ a) = a)$$
$$\wedge\ (\forall\ r\bullet\ pred\ r \Leftrightarrow rep\ (abs\ r) = r)$$
$$\wedge\ OneOne\ rep)$$

$$\boldsymbol{type\_defn\_lemma1} =$$
$$\vdash \forall\ f\ g\bullet\ (\forall\ x\bullet\ f\ (g\ x) = x) \Rightarrow (\forall\ x\ y\bullet\ g\ x = g\ y \Rightarrow x = y)$$

$$\boldsymbol{type\_defn\_lemma2} =$$
$$\vdash \forall\ p\ f\ g$$
$$\bullet\ (\forall\ x\bullet\ p\ x \Rightarrow f\ (g\ x) = x) \Rightarrow (\forall\ x\ y\bullet\ p\ x \wedge p\ y \Rightarrow g\ x = g\ y \Rightarrow x = y)$$

$$\boldsymbol{type\_defn\_lemma3} =$$
$$\vdash (\exists\ f\bullet\ TypeDefn\ (\lambda\ x\bullet\ T)\ f)$$
$$\Rightarrow (\exists\ abs\ rep\bullet\ (\forall\ a\bullet\ abs\ (rep\ a) = a) \wedge (\forall\ r\bullet\ rep\ (abs\ r) = r))$$

$$\boldsymbol{type\_defn\_lemma4} =$$
$$\vdash \forall\ pred$$
$$\bullet\ (\exists\ f\bullet\ TypeDefn\ pred\ f)$$
$$\Rightarrow (\exists\ abs\ rep$$
$$\bullet\ (\forall\ a\bullet\ abs\ (rep\ a) = a)$$
$$\wedge\ (\forall\ r\bullet\ pred\ r \Leftrightarrow rep\ (abs\ r) = r)$$
$$\wedge\ OneOne\ rep$$
$$\wedge\ (\forall\ a\bullet\ pred\ (rep\ a)))$$

$$\boldsymbol{oneone\_contrapos\_lemma} =$$
$$\vdash OneOne\ f \Rightarrow (\forall\ x\ y\bullet\ \neg\ x = y \Rightarrow \neg\ f\ x = f\ y)$$

5 Sets

5.1 Pairwise Disjointness

Here is a definition of "Pairwise disjoint".

HOL Constant

$\$\boldsymbol{PDisj}: 'a\ \ SET\ SET \to BOOL$

$\forall ss\bullet\ PDisj\ ss \Leftrightarrow \neg\ \exists t\ u\bullet \{t; u\} \subseteq ss \wedge \neg\ t = u \wedge \neg\ t \cap u = \{\}$

5.2 Transitivity of Inclusion

$\boldsymbol{\subseteq\_trans\_thm} = \vdash \forall\ A\ B\ C\bullet\ A \subseteq B \wedge B \subseteq C \Rightarrow A \subseteq C$

5.3 Singleton Subsets

$\boldsymbol{singleton\_subset\_lemma} =$
$\quad \vdash \forall\ x\ v\bullet \{x\} \subseteq V \Leftrightarrow x \in V$

5.4 Image of a Set under a Function

HOL Constant

$\boldsymbol{FunImage}: ('a \to 'b) \to 'a\ SET \to 'b\ SET$

$\forall f\ A\bullet\ FunImage\ f\ A = \{b \mid \exists a\bullet\ a \in A \wedge f\ a = b\}$

$\boldsymbol{FunImage\_o\_thm} =$
$\quad \vdash \forall\ A\ f\ g\bullet\ FunImage\ (f\ o\ g)\ A = FunImage\ f\ (FunImage\ g\ A)$

$\boldsymbol{FunImage\_mono\_thm} =$
$\quad \vdash \forall\ A\ B\ f\bullet\ A \subseteq B \Rightarrow FunImage\ f\ A \subseteq FunImage\ f\ B$

5.5 Set Displays

The following are introduced to facilitate reasoning about sets of truth values below.

$\boldsymbol{insert\_com\_thm} =$
$\quad \vdash \forall\ x\ y\ z\bullet\ Insert\ x\ (Insert\ y\ z) = Insert\ y\ (Insert\ x\ z)$

$\boldsymbol{insert\_twice\_thm} =$
$\quad \vdash \forall\ x\ y\bullet\ Insert\ x\ (Insert\ x\ y) = Insert\ x\ y$

$\in\_disp\_\Rightarrow\_thm =$
$\quad \vdash \forall p\ d\ s\bullet\ (\forall\ e\bullet\ e \in Insert\ d\ s \Rightarrow p\ e) \Leftrightarrow p\ d \wedge \forall\ e\bullet\ e \in s \Rightarrow p\ e$

A conversion to apply ∈_disp_⇒_thm (because we don't have higher order rewriting).

SML

```
val ∈_disp_⇒_conv:CONV = fn t =>
    let val (e, body) = dest_∀ t;
        val (lh, rh) = dest_⇒ body;
        val (_, [vare, ins]) = strip_app lh;
        val (_, [d, s]) = strip_app ins
        val p = mk_λ (vare, rh);
        val equiv = conv_rule (LEFT_C(SIMPLE_BINDER_C(RIGHT_C β_conv))) (list_∀_elim [p, d,
    in equiv
    end handle _ => fail_conv t;

val ∈_disp_⇒_tac = conv_tac (MAP_C ∈_disp_⇒_conv);
```

5.6 NESET - A Type of Non-Empty Sets

SML

```
new_type_defn (["NESET"], "NESET", ["'a"],
    tac_proof (([], ⌜∃x:'a ℙ• (λy• ∃z• z ∈ y) x⌝),
    ∃_tac ⌜{ϵx:'a• T}⌝ THEN rewrite_tac [] THEN ∃_tac ⌜ϵx:'a• T⌝ THEN rewrite_tac[]) );
```

HOL Constant

```
NeSet : 'a ℙ → 'a NESET;
PeSet : 'a NESET → 'a ℙ
├──────────────────────────────────────
    (∀x• ∃y• y ∈ PeSet x)
  ∧ (∀x y• x = y ⇔ ∀z• z ∈ PeSet x ⇔ z ∈ PeSet y)
  ∧ (∀x y• x ∈ y ⇒ PeSet (NeSet y) = y)
  ∧ (∀y• NeSet (PeSet y) = y)

NeSet_ne_thm =
    ⊢ ∀ x• ∃ y• y ∈ PeSet x
NeSet_ext_thm =
    ⊢ ∀ x y• x = y ⇔ (∀ z• z ∈ PeSet x ⇔ z ∈ PeSet y)
NeSet_fc_thm =
    ⊢ ∀ x y• x ∈ y ⇒ PeSet (NeSet y) = y
NeSet_PeSet_thm =
    ⊢ ∀ y• NeSet (PeSet y) = y

PeSet_Insert_thm =
    ⊢ ∀ x y• PeSet (NeSet (Insert x y)) = Insert x y
```

HOL Constant

$$\boldsymbol{MemOf} : 'a\ NESET \to 'a$$

$$\forall x \bullet\ MemOf\ x = \epsilon y \bullet\ y \in PeSet\ x$$

$$\boldsymbol{MemOf\_memof\_thm} =$$
$$\vdash\ \forall\ x \bullet\ MemOf\ x \in PeSet\ x$$
$$\boldsymbol{MemOf\_NeSet\_unit\_thm} =$$
$$\vdash\ \forall\ x \bullet\ MemOf\ (NeSet\ \{x\}) = x$$

SML

```
add_pc_thms1 "'rbjmisc" [NeSet_ne_thm];
add_pc_thms "'rbjmisc" [NeSet_PeSet_thm, MemOf_memof_thm, PeSet_Insert_thm, MemOf_NeSet_un
```

5.7 Cantor's Theorem

Presumably there is a proof of this somewhere but here is another.

The following is a record of a proof session with ProofPower in which the Cantor's theorem is proven. The logic is very close to that of Principia Mathematica[4], being based on Church's formulation[1] of the Simple Theory of Types (which is more or less equivalent to Russell's Theory of Types [3] once the ramifications have been neutralised by the axiom of reducibility).

The interactive proof tool effectively checks a formal proof which it has constructed behind the scenes following instructions from the user of the system, but does not display the full details of the proof (complete automation is the ideal, which we approach from afar). The user gives his instructions in a language called 'Standard ML' (in the passages headed 'SML') in which the 'ML' stands for 'Meta-Language', the sections marked 'ProofPower output' show the output from the proof tool. The proof is conducted using a 'goal package' (a bit of software written in SML) which supports a backward proof idiom in which the proof begins with the conjecture to be proven, and simplifies this conjecture by reverse application of rules until we reach axiomatic premises. Behind the scenes the proof tool verifies the existence of a forward proof from axioms to the desired theorem.

The proof is begun by stating the goal (conjecture) to be proven, which is that there does not exist a function from a type ⌜:'a⌝ onto the type ⌜:'a SET⌝ (in which $'a$ is a type variable).

SML

```
set_goal([], ⌜¬ ∃f:'a → 'a SET• ∀s:'a SET• ∃e:'a• f e = s⌝);
```

ProofPower output

```
(* *** Goal "" *** *)

(* ?⊢ *)  ⌜¬ (∃ f• ∀ s• ∃ e• f e = s)⌝
```

By the most routine transformations:

SML

```
a (REPEAT strip_tac);
```

we get (showing some intermediate steps):

ProofPower output

$$\vdash (* \ ?\vdash \ *) \ \ \ulcorner \forall \ f\bullet \ \neg \ (\forall \ s\bullet \ \exists \ e\bullet \ f \ \ e \ = \ s)\urcorner$$

$$\vdash (* \ ?\vdash \ *) \ \ \ulcorner \neg \ (\forall \ s\bullet \ \exists \ e\bullet \ f \ \ e \ = \ s)\urcorner$$

$$\vdash (* \ ?\vdash \ *) \ \ \ulcorner \exists \ s\bullet \ \neg \ (\exists \ e\bullet \ f \ \ e \ = \ s)\urcorner$$

Which is asking for a counterexample to the supposition that f is a surjection.

The required counterexample is supplied thus:

SML

```
a (∃_tac ⌜{x | ¬ x ∈ f x }⌝);
```

and we then have to prove that it is indeed a counterexample:

ProofPower output

$$\vdash (* \ ?\vdash \ *) \ \ \ulcorner \neg \ (\exists \ e\bullet \ f \ \ e \ = \ \{x | \neg \ x \in f \ \ x\})\urcorner$$

which again involves first some routine inferences:

SML

```
a (REPEAT strip_tac);
```

which goes:

ProofPower output

$$\vdash (* \ ?\vdash \ *) \ \ \ulcorner \forall \ e\bullet \ \neg \ f \ \ e \ = \ \{x | \neg \ x \in f \ \ x\}\urcorner$$

$$\vdash (* \ ?\vdash \ *) \ \ \ulcorner \neg \ f \ \ e \ = \ \{x | \neg \ x \in f \ \ x\}\urcorner$$

To progress this proof we now must use extensionality of sets transforming the equation into an equivalence by rewriting.

SML

```
a (rewrite_tac [sets_ext_clauses]);
```

Which gives:

ProofPower output

$$\vdash (* \ ?\vdash \ *) \ \ \ulcorner \neg \ (\forall \ x\bullet \ x \in f \ \ e \Leftrightarrow \neg \ x \in f \ \ x)\urcorner$$

which is too obvious for us to care how it is discharged!

SML

```
a (prove_tac []);
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

We now have a theorem, which we can save in our theory:

SML

```
val cantors_thm = save_pop_thm "cantors_thm";
```

ProofPower output

$$val \ \ cantors\_thm \ = \ \vdash \ \neg \ (\exists \ f\bullet \ \forall \ s\bullet \ \exists \ e\bullet \ f \ \ e \ = \ s) \ : \ THM$$

6 Type OPT

SML

```
set_merge_pcs ["hol1", "'rbjmisc"];

new_type_defn (["OPT"], "OPT", ["'a"],
    tac_proof (([], ⌜∃x:'a+ONE• (λy• T) x⌝), ∃_tac ⌜InR One⌝ THEN rewrite_tac []) );
```

To make use of the type abbreviation 'OPT' more readable the following constants are introduced:

HOL Constant

```
Value : 'a → 'a OPT;
Undefined : 'a OPT
```

```
OneOne Value
    ∧ (∀x• ¬ Value x = Undefined)
    ∧ (∀y• y = Undefined ∨ (∃z• y = Value z))

opt_cases_thm =
  ⊢ ∀ x• x = Undefined ∨ (∃ y• x = Value y)

value_not_undefined_lemma =
  ⊢ ∀ x• ¬ Value x = Undefined ∧ ¬ Undefined = Value x
```

HOL Constant

```
ValueOf : 'a OPT → 'a
```

```
∀v• ValueOf (Value v) = v
```

HOL Constant

```
IsDefined : 'a OPT → BOOL
```

```
∀v• IsDefined v ⇔ ¬ v = Undefined
```

6.1 Proof Contexts

SML

```
add_pc_thms "'rbjmisc" (map get_spec [⌜IsDefined⌝, ⌜ValueOf⌝] @
    [value_not_undefined_lemma, value_oneone_lemma]);
set_merge_pcs ["basic_hol1", "'sets_alg", "'ℝ", "'rbjmisc"];
```

7 Lists

7.1 List Membership

SML

```
declare_infix(300, "∈L");
```

HOL Constant

$\$\in_{\boldsymbol{L}}$: $'a \to 'a\ \ LIST \to BOOL$

$\forall\ a\ b\ al\bullet\ (a \in_L [] \Leftrightarrow F)$
$\quad \wedge\ (a \in_L (Cons\ b\ al) \Leftrightarrow a = b \vee a \in_L al)$

7.2 Quantification

HOL Constant

$\forall_{\boldsymbol{L}}$: $BOOL\ LIST \to BOOL$

$\forall\ bl\bullet\ \forall_L\ bl = Fold\ \$\wedge\ bl\ T$

HOL Constant

$\exists_{\boldsymbol{L}}$: $BOOL\ LIST \to BOOL$

$\forall\ bl\bullet\ \exists_L\ bl = Fold\ \$\vee\ bl\ F$

$\forall_{\boldsymbol{L}}\textbf{\_thm} =$
$\quad \vdash (\forall_L [] \Leftrightarrow T) \wedge (\forall h\ list\bullet\ \forall_L (Cons\ h\ list) \Leftrightarrow h \wedge \forall_L list)$

$\exists_{\boldsymbol{L}}\textbf{\_thm} =$
$\quad \vdash (\exists_L [] \Leftrightarrow F) \wedge (\forall\ h\ list\bullet\ \exists_L (Cons\ h\ list) \Leftrightarrow h \vee \exists_L list)$

$\forall_{\boldsymbol{L}}\textbf{\_clauses} =$
$\quad \vdash \forall_L [] \Leftrightarrow T \wedge (\forall\ t\bullet\ \forall_L (Cons\ T\ t) \Leftrightarrow \forall_L t) \wedge (\forall\ t\bullet\ \forall_L (Cons\ F\ t) \Leftrightarrow F)$

$\exists_{\boldsymbol{L}}\textbf{\_clauses} =$
$\quad \vdash \exists_L [] \Leftrightarrow F \wedge (\forall\ t\bullet\ \exists_L (Cons\ F\ t) \Leftrightarrow \exists_L t) \wedge (\forall\ t\bullet\ \exists_L (Cons\ T\ t) \Leftrightarrow T)$

$\forall_{\boldsymbol{L}}\textbf{\_append\_thm} =$
$\quad \vdash \forall\ l\ m\bullet\ \forall_L (l \frown m) \Leftrightarrow \forall_L l \wedge \forall_L m$

$\exists_{\boldsymbol{L}}\textbf{\_append\_thm} =$
$\quad \vdash \forall\ l\ m\bullet\ \exists_L (l \frown m) \Leftrightarrow \exists_L l \vee \exists_L m$

7.3 Proof Contexts

SML

```
add_pc_thms "'rbjmisc" (map get_spec [⌜$∈_L⌝] @
    [∀_L_clauses, ∃_L_clauses, ∀_L_thm, ∃_L_thm, ∀_L_append_thm, ∃_L_append_thm]);
set_merge_pcs ["basic_hol1", "'sets_alg", "'ℝ", "'rbjmisc"];
```

7.4 Mapping Constructors

The idea here is to facilitate the construction of a list of objects of some kind (typically a HOL labelled product), given a curried constructor and lists of the operands. We will need a different one for each arity of constructor, so I will use a numeric suffix.

HOL Constant

$\boldsymbol{MapCf_2}$: $('a \to 'b \to 'c) \to 'a\ LIST \to 'b\ LIST \to 'c\ LIST$

$\forall\ cf\ al\ bl \bullet\ \ MapCf_2\ cf\ al\ bl = Map\ (Uncurry\ cf)\ (Combine\ al\ bl)$

HOL Constant

$\boldsymbol{MapCf_3}$: $('a \to 'b \to 'c \to 'd) \to 'a\ LIST \to 'b\ LIST \to 'c\ LIST \to 'd\ LIST$

$\forall\ cf\ al\ bl\ cl \bullet\ \ MapCf_3\ cf\ al\ bl\ cl =$
$\quad Map\ (Uncurry\ (Uncurry\ cf))\ (Combine\ (Combine\ al\ bl)\ cl)$

7.5 Liberal Combine

This is a combine function which "works" with lists of different lengths.

HOL Constant

$\boldsymbol{Combine2}$: $'a\ LIST \to 'b\ LIST \to ('a \times 'b)\ LIST$

$(\forall b \bullet\ \ Combine2\ []\ b = [])$
$\wedge\ (\forall a \bullet\ \ Combine2\ a\ [] = [])$
$\wedge\ (\forall ha\ ta\ hb\ tb \bullet\ Combine2\ (Cons\ ha\ ta)\ (Cons\ hb\ tb) = Cons\ (ha, hb)\ (Combine2\ ta\ tb))$

7.6 Lists of Sets

HOL Constant

$\boldsymbol{List2Set}$: $'a\ LIST \to 'a\ SET$

$\forall l \bullet\ List2Set\ l = \{e \mid e \in_L l\}$

HOL Constant

$\boldsymbol{ListUnion}$: $'a\ SET\ LIST \to 'a\ SET$

$\forall l \bullet\ ListUnion\ l = \bigcup\ (List2Set\ l)$

HOL Constant

$\boldsymbol{ListFunUnion}$: $('a\ SET \to 'a\ SET)\ LIST \to ('a\ SET \to 'a\ SET)$

$\forall l\ as \bullet\ ListFunUnion\ l\ as = ListUnion\ (Map\ (\lambda f \bullet\ f\ as)\ l)$

$\boldsymbol{ListUnion\_thm} =$
$\quad \vdash ListUnion\ [] = \{\} \wedge (\forall\ h\ t \bullet\ ListUnion\ (Cons\ h\ t) = h \cup ListUnion\ t)$

7.7 Lists of Natural Numbers

A function for making a list of ascending natural numbers.

SML

$$declare\_infix\ (300,\ "..{}_L");$$

HOL Constant

$$\$..{}_L : \mathbb{N} \to \mathbb{N} \to \mathbb{N}\ LIST$$

$$\forall\ x\ y \bullet\ x\ ..{}_L\ 0 = []$$
$$\wedge\ x\ ..{}_L\ (y + 1) = if\ x \leqslant y\ then\ (x\ ..{}_L\ y)\ @\ [y{+}1]\ else\ []$$

8 Natural Numbers and Arithmetic

8.1 Primitive Recursion

$$\boldsymbol{prim\_rec\_thm2} = \vdash \forall\ z\ s \bullet \exists\ f \bullet\ f\ 0 = z \wedge (\forall\ n \bullet\ f\ (n + 1) = s\ (f\ n)\ n)$$

9 Real Numbers and Analysis

9.1 Products

$$\mathbb{R}\_prod\_sign\_iff\_clauses$$
$$\vdash (\forall\ x\ y \bullet\ \mathbb{NR}\ 0 <R\ x *_R y \quad \Leftrightarrow \mathbb{NR}\ 0 <_R x \wedge \mathbb{NR}\ 0 <_R y \vee x <_R \mathbb{NR}\ 0 \wedge y <_R \mathbb{NR}\ 0)$$
$$\wedge (\forall\ x\ y \bullet\ x *_R y <_R \mathbb{NR}\ 0 \quad \Leftrightarrow \mathbb{NR}\ 0 <_R x \wedge y <_R \mathbb{NR}\ 0 \vee x <_R \mathbb{NR}\ 0 \wedge \mathbb{NR}\ 0 <_R y)$$
$$\wedge (\forall\ x\ y \bullet\ \mathbb{NR}\ 0 \leqslant_R x *_R y \quad \Leftrightarrow \mathbb{NR}\ 0 \leqslant_R x \wedge \mathbb{NR}\ 0 \leqslant_R y \vee x \leqslant_R \mathbb{NR}\ 0 \wedge y \leqslant_R \mathbb{NR}\ 0)$$
$$\wedge (\forall\ x\ y \bullet\ x *_R y \leqslant_R \mathbb{NR}\ 0 \quad \Leftrightarrow \mathbb{NR}\ 0 \leqslant_R x \wedge y \leqslant_R \mathbb{NR}\ 0 \vee x \leqslant_R \mathbb{NR}\ 0 \wedge \mathbb{NR}\ 0 \leqslant_R y)$$
$$\wedge (\forall\ x\ y \bullet\ x *_R y = \mathbb{NR}\ 0 \quad \Leftrightarrow x = \mathbb{NR}\ 0 \vee y = \mathbb{NR}\ 0)$$
$$\wedge (\forall\ x\ y \bullet\ \mathbb{NR}\ 0 = x *_R y \quad \Leftrightarrow x = \mathbb{NR}\ 0 \vee y = \mathbb{NR}\ 0)$$

9.2 Squares

9.3 Sums

9.4 Abs

The following arithmetic results are obtained for reasoning about norms on real vector spaces, in particular to prove that *Abs* is a norm over the reals and that the defined product operation over norms yields a norm.

$$\mathbb{R}\_Abs\_Norm\_clauses$$
$$\vdash (\forall\ v \bullet\ \mathbb{NR}\ 0 \leqslant_R Abs_R\ v)$$
$$\wedge (\forall\ v \bullet\ (Abs_R\ v = \mathbb{NR}\ 0) \Leftrightarrow v = \mathbb{NR}\ 0)$$
$$\wedge (\forall\ x\ v \bullet\ Abs_R\ (x *_R v) = Abs_R\ x *_R Abs_R\ v)$$
$$\wedge (\forall\ v\ w \bullet\ Abs_R\ (v +_R w) \leqslant_R Abs_R\ v +_R Abs_R\ w)$$

9.5 Square Root

HOL Constant

$$\boldsymbol{SqrtA} : \mathbb{R} \to \mathbb{R}$$

$$\forall\ x \bullet\ \mathbb{NR}\ 0 \leqslant SqrtA\ x$$
$$\wedge\ (SqrtA\ x)\hat{}2 = Abs\ x$$

9.6 Sums of Countable Collections of Reals

In evaluating the cosmological consequences of Newton's Laws it is desirable to formulate them as cosmological theories in ways which do not prejudge such questions as whether the cardinality of the universe is finite. To do this it is necessary to be able, where possible, to form the sum of an infinite set of reals, possibly even an uncountably infinite set of real numbers.

For the most general formulations it seems possible that the use of non-standard reals might be needed. We are concerned here with what can be done with standard reals, i.e. with formalising the notion that some collection of real numbers has a finite sum.

The following definition gives the sum of a possibly finite or countable collection of real numbers.

SML

```
declare_infix (300, "⇻>");
```

HOL Constant

$$\$⇻> : ('a \to \mathbb{R} + ONE) \to \mathbb{R} \to BOOL$$

$$\forall c\ r \bullet\ c ⇻> r \Leftrightarrow$$
$$\exists s \bullet\ (\forall\ a\ n\ m \bullet\ IsL\ (c\ a) \Rightarrow s\ n = s\ m \Rightarrow n = m)$$
$$\Rightarrow (Series\ (\lambda n \bullet\ if\ IsR\ (c\ (s\ n))\ then\ 0_R\ else\ OutL\ (c\ (s\ n)))) -> r$$

10 Cartesian and Dependent Products

10.1 Cartesian Products

$$\boldsymbol{cp\_eq\_thm1} =$$
$$\vdash \forall\ x\ y\ v\ w\ p\ q \bullet\ p \in x \wedge q \in y \Rightarrow (x \times y) = (v \times w) \Rightarrow x = v \wedge y = w$$

$$\boldsymbol{cp\_eq\_thm2} =$$
$$\vdash \forall\ x\ y\ v\ w\ p \bullet\ p \in (x \times y) \Rightarrow (x \times y) = (v \times w) \Rightarrow x = v \wedge y = w$$

$$\boldsymbol{cp\_l\_part\_thm} = \vdash \forall\ l\ r\ x \bullet\ x \in r \Rightarrow l = \{m | \exists\ p \bullet\ p \in (l \times r) \wedge m = Fst\ p\}$$

$$\boldsymbol{cp\_r\_part\_thm} = \vdash \forall\ l\ r\ x \bullet\ x \in l \Rightarrow r = \{m | \exists\ p \bullet\ p \in (l \times r) \wedge m = Snd\ p\}$$

$$\boldsymbol{cp\_part\_thm} =$$
$$\vdash \forall\ l\ r\ x \bullet\ x \in (l \times r)$$
$$\Rightarrow l = \{m | \exists\ p \bullet\ p \in (l \times r) \wedge m = Fst\ p\}$$
$$\wedge\ r = \{m | \exists\ p \bullet\ p \in (l \times r) \wedge m = Snd\ p\}$$

10.2 Distributed Cartesian Product

The "distributed cartesian product" is what you need to get the equivalance classes generated by two relations to the equivalence classes generated by the product of the equivalence relations (see section **??**).

SML

$$declare\_infix\ (340,\ "\times_D");$$

HOL Constant

$$\$\times_{\boldsymbol{D}} : ('a\ SET\ SET) \to ('b\ SET\ SET) \to\ (('a \times 'b)\ SET\ SET)$$

$$\forall l\ r \bullet\ l \times_D r = \{cp \mid \exists\ leq\ req \bullet\ leq \in l \wedge req \in r \wedge cp = (leq \times req)\}$$

$$\times_{\boldsymbol{D}}\textbf{\textit{-ext\_thm}} =$$
$$\vdash \forall\ l\ r\ cp \bullet\ cp \in l \times_D r \Leftrightarrow (\exists\ leq\ req \bullet\ leq \in l \wedge req \in r \wedge cp = (leq \times req))$$

10.3 Dependent Function Spaces ??

Our generic treatment of algebraic operators represents operators as functions over indexed sets of arugments. In order to take a quotient of an algebra involving such operators we need to lift the equivalence relation over the domain of the algebra to one over indexed sets of domain values. This is taking an arbitrary power of the equivalence relation.

These indexed sets are actually functions, and the equivalence classes are like dependent function spaces, whose type corresponds to an indexed set of the equivalence classes over the domain, i.e. a function from the indexes into the set of equivalence classes.

It is therefore useful to define this dependent function space operation which may also be though of as a dependent cartesian power.

HOL Constant

$$\boldsymbol{\Pi_f} : 'b\ SET \to ('b \to 'a\ SET) \to ('b \to 'a)\ SET$$

$$\forall\ is\ f \bullet\ \Pi_f\ is\ f = \{g \mid \forall i \bullet\ i \in is \Rightarrow g\ i \in f\ i\}$$

Now we take a set of (probably equivalence) classes and map the dependent function space over all the indexed sets taken from this set.

HOL Constant

$$\times_{\boldsymbol{\Pi}} : 'b\ SET \to ('a\ SET\ SET) \to ('b \to 'a)\ SET\ SET$$

$$\forall\ is\ ss \bullet\ \times_\Pi\ is\ ss = \{g \mid \exists h \bullet\ (\forall i \bullet\ i \in is \Rightarrow h\ i \in ss) \wedge g = \Pi_f\ is\ h\}$$

11 Relation Products

The kind of relation which we consider here is the kind which is used in the theory *equiv_rel*, structures represented by an ordered pair of which the first element is the domain of the relation and the second is a relation as a curried function of two operands into type *BOOL*.

This section provides some small additions to the theory "equiv_rel" related to lifting functions over quotients.

SML

$$declare\_infix(210,\ "RelProd");$$
$$declare\_infix(230,\ "\leqslant_l");$$
$$declare\_infix(230,\ "\leqslant_r");$$

HOL Constant

$$\$RelProd : ('a\ SET \times ('a \to 'a \to BOOL))$$
$$\to ('b\ SET \times ('b \to 'b \to BOOL))$$
$$\to (('a \times 'b)\ SET \times ('a \times 'b \to 'a \times 'b \to BOOL))$$

$$\forall\ \$\leqslant_l\ \$\leqslant_r\ L\ R\bullet\ ((L,\ \$\leqslant_l)\ RelProd\ (R,\ \$\leqslant_r)) =$$
$$((L \times R),$$
$$\lambda(l1,\ r1)\ (l2,\ r2)\bullet\ l1 \in L \wedge l2 \in L \wedge r1 \in R \wedge r2 \in R$$
$$\wedge\ l1 \leqslant_l l2 \wedge r1 \leqslant_r r2)$$

$$RelProd\_projections\_thm =$$
$$\vdash \forall\ (L,\ \$\leqslant_l)\ (R,\ \$\leqslant_r)\bullet$$
$$Fst\ ((L,\ \$\leqslant_l)\ RelProd\ (R,\ \$\leqslant_r)) = (L \times R)$$
$$\wedge\ Snd\ ((L,\ \$\leqslant_l)\ RelProd\ (R,\ \$\leqslant_r)) = (\lambda\ (l1,\ r1)\ (l2,\ r2)$$
$$\bullet\ l1 \in L \wedge l2 \in L \wedge r1 \in R \wedge r2 \in R \wedge l1 \leqslant_l l2 \wedge r1 \leqslant_r r2)$$

The product construction preserves various properties of relations.

$$Trans\_RelProd\_thm =$$
$$\vdash \forall\ (L,\ \$\widehat{=}_l)\ (R,\ \$\widehat{=}_r)\bullet$$
$$Trans\ (L,\ \$\widehat{=}_l) \wedge Trans\ (R,\ \$\widehat{=}_r) \Rightarrow Trans\ ((L,\ \$\widehat{=}_l)\ RelProd\ (R,\ \$\widehat{=}_r))$$

$$Sym\_RelProd\_thm =$$
$$\vdash \forall\ (L,\ \$\widehat{=}_l)\ (R,\ \$\widehat{=}_r)\bullet$$
$$Sym\ (L,\ \$\widehat{=}_l) \wedge Sym\ (R,\ \$\widehat{=}_r) \Rightarrow Sym\ ((L,\ \$\widehat{=}_l)\ RelProd\ (R,\ \$\widehat{=}_r))$$

$$Refl\_RelProd\_thm =$$
$$\vdash \forall\ (L,\ \$\widehat{=}_l)\ (R,\ \$\widehat{=}_r)\bullet$$
$$Refl\ (L,\ \$\widehat{=}_l) \wedge Refl\ (R,\ \$\widehat{=}_r) \Rightarrow Refl\ ((L,\ \$\widehat{=}_l)\ RelProd\ (R,\ \$\widehat{=}_r))$$

12 Powers of Relations

Universal algebra (below) involves operators of arbitrary arity, and reasoning generally about algebras involving such operators involves raising equivalence relations to arbitrary powers.

We define such operations first for relations in general.

The operators are represented by functions over indexed sets of arguments, the indexed sets being represented by total functions over a type of indexes, together with a subset of the type indicating the

range of significance of the functions in the domain of the operators. The operators are themselves also significant only for arguments all of which fall in some domain (the domain of a structure), but this need not concern us here.

Our concern here is not with the operators, but with quotients of the domain of the operators, and with the resulting equivalence relations over the domain of the operators. So we want to take an equivalence relation over some type, together with a set of indices, and obtain an equivalence relation over the total functions from indexes to values in the domain. We define this for arbitrary relations, and then prove (later) that when the relation is an equivalence the result will be an equivalence.

HOL Constant

$$\$\boldsymbol{RelPower} : ('a\ SET \times ('a \to 'a \to BOOL)) \to 'b\ SET \to (('b \to 'a)\ SET \times (('b \to 'a) \to ('b \to 'a) \to BOOL))$$

$$\forall\ D\ r\ is \bullet\ RelPower\ (D,\ r)\ is = (\{f \mid \forall i \bullet\ i \in is \Rightarrow f\ i \in D\},\ \lambda f\ g \bullet\ \forall i \bullet\ i \in is \Rightarrow r\ (f\ i)\ (g\ i))$$

We now show that this construction preserves various properties of the relation.

RelPower_Trans_thm =

$$\vdash \forall\ (D,\ r)\ is \bullet\ Trans\ (D,\ r) \Rightarrow Trans\ (RelPower\ (D,\ r)\ is)$$

RelPower_Sym_thm =

$$\vdash \forall\ (D,\ r)\ is \bullet\ Sym\ (D,\ r) \Rightarrow Sym\ (RelPower\ (D,\ r)\ is)$$

RelPower_Refl_thm =

$$\vdash \forall\ (D,\ r)\ is \bullet\ Refl\ (D,\ r) \Rightarrow Refl\ (RelPower\ (D,\ r)\ is)$$

13 Group Theory

SML

```
new_parent "group_egs";
get_alias_info "Append";
set_merge_pcs ["basic_hol1", "'sets_alg", "'ℝ", "'rbjmisc"];
```

13.1 Group Products

HOL Constant

$$\boldsymbol{GroupProduct} : 'a\ GROUP \to 'b\ GROUP \to ('a \times 'b)\ GROUP$$

$$\begin{aligned}&\forall\ G\ H \bullet\ GroupProduct\ G\ H =\\ &\quad let\ car = (Car\ G \times Car\ H)\\ &\quad and\ prod\ \ (la,\ lb)\ (ra,\ rb) = ((la.ra)\ G,\ (lb.rb)\ H)\\ &\quad and\ unit = (Unit\ G,\ Unit\ H)\\ &\quad and\ inv\ (a,\ b) = ((a\ ^{\sim})\ G,\ (b\ ^{\sim})\ H)\\ &\quad in\ MkGROUP\ car\ prod\ unit\ inv\end{aligned}$$

SML

```
declare_alias ("*", ⌜GroupProduct⌝);
```

```
group_product_thm = ⊢ ∀g:'a GROUP; h:'b GROUP•
    g ∈ Group ∧ h ∈ Group ⇒ g * h ∈ Group
```

13.2 Abelian Groups

HOL Constant

```
AbelianGroup : 'a GROUP SET
```

```
∀ G• G ∈ AbelianGroup ⇔ G ∈ Group
    ∧ ∀ u v:'a• u ∈ Car G ∧ v ∈ Car G
            ⇒ (u.v) G = (v.u) G
```

```
abelian_group_product_thm = ⊢ ∀g:'a GROUP; h:'b GROUP•
    g ∈ AbelianGroup ∧ h ∈ AbelianGroup ⇒ (g * h) ∈ AbelianGroup
```

```
ℝ_plus_abelian_thm = ⊢ ℝ+ ∈ AbelianGroup
```

14 Topology

SML

```
new_parent "topology";
get_alias_info "Append";
```

14.1 Bases etc.

The following definitions belong properly in the theory "topology".

First we define the relationship between a *base* and the topology of which it is a base.

SML

```
declare_infix (300, "BaseOf");
```

HOL Constant

```
$BaseOf : 'a SET SET → 'a SET SET → BOOL
```

```
∀ base topology• base BaseOf topology ⇔
    ∀s• s ∈ topology ⇒ ∃ ss• ss ⊆ base ∧ s = ⋃ ss
```

However, what we really need here is the construction of a topology from an arbitrary set of sets, which is done as follows:

HOL Constant

```
$TopologyFrom : 'a SET SET → 'a SET SET
```

```
∀ sets• TopologyFrom sets =
    ⋂ {topology | topology ∈ Topology ∧ sets ⊆ topology}
```

15 Disjoint Unions (Sum)

Two ways of constructing functions over disjoint unions.

HOL Constant

$$\boldsymbol{Fun_+}: ('a \to 't) \to ('b \to 'u) \to ('a + 'b \to 't + 'u)$$

$$\forall f{:}'a \to 't;\ g{:}'b \to 'u;\ ab{:}'a + 'b \bullet$$
$$Fun_+\ f\ g\ ab = if\ IsL\ ab\ then\ InL\ (f\ (OutL\ ab))\ else\ InR\ (g\ (OutR\ ab))$$

HOL Constant

$$\boldsymbol{FunSum}: ('a \to 't) \to ('b \to 't) \to ('a + 'b \to 't)$$

$$\forall f{:}'a \to 't;\ g{:}'b \to 't;\ ab{:}'a + 'b \bullet$$
$$FunSum\ f\ g\ ab = if\ IsL\ ab\ then\ f\ (OutL\ ab)\ else\ g\ (OutR\ ab)$$

$$\boldsymbol{IsL\_IsR\_lemma} =$$
$$\vdash\ \forall\ x \bullet\ IsR\ x \Leftrightarrow \neg\ IsL\ x$$

16 Indexed Sets

SML

$$declare\_type\_abbrev(\texttt{"}\boldsymbol{IX}\texttt{"},\ [\texttt{"}'a\texttt{"},\ \texttt{"}'b\texttt{"}],\ \ulcorner{:}'a \to 'b\ OPT\urcorner);$$

HOL Constant

$$\boldsymbol{IxVal} : ('b, 'a)\ IX \to 'b \to 'a$$

$$\forall is\ g \bullet\ IxVal\ is\ g = ValueOf\ (is\ g)$$

HOL Constant

$$\boldsymbol{IxRan} : ('b, 'a)\ IX \to 'a\ SET$$

$$\forall is \bullet\ IxRan\ is = \{v \mid \exists \alpha \bullet\ Value\ v = is\ \alpha\}$$

HOL Constant

$$\boldsymbol{IxDom} : ('b, 'a)\ IX \to 'b\ SET$$

$$\forall is \bullet\ IxDom\ is = \{i \mid IsDefined\ (is\ i)\}$$

$$\boldsymbol{ix\_domran\_lemma} =$$
$$\vdash\ \forall\ x\ y \bullet\ x \in IxDom\ y \Rightarrow IxVal\ y\ x \in IxRan\ y$$

$$\boldsymbol{ix\_valueof\_ran\_lemma} =$$
$$\vdash\ \forall\ x\ y \bullet\ \neg\ x\ y = Undefined \Rightarrow ValueOf\ (x\ y) \in IxRan\ x$$

HOL Constant

$$\boldsymbol{IxDomRes} : 'a\ SET \to ('a, 'b)\ IX \to ('a, 'b)\ IX$$

$$\forall ns\ is \bullet\ IxDomRes\ ns\ is = \lambda x \bullet\ if\ x \in ns\ then\ is\ x\ else\ Undefined$$

HOL Constant

$$\boldsymbol{IxRanRes} : ('a, 'b)\ IX \to 'b\ SET \to ('a, 'b)\ IX$$

$$\forall is\ ns \bullet\ IxRanRes\ is\ ns = \lambda x \bullet\ if\ x \in IxDom\ is \wedge ValueOf\ (is\ x) \in ns\ then\ is\ x\ else\ Undefined$$

SML

```
declare_alias ("⊲", ⌜IxDomRes⌝);
declare_alias ("⊳", ⌜IxRanRes⌝);
```

HOL Constant

$$\boldsymbol{IxUd} : ('a, 'b)\ IX \to 'a\ SET$$

$$\forall is \bullet\ IxUd\ is = \{i \mid is\ i = Undefined\}$$

HOL Constant

$$\boldsymbol{IxOverRide} : ('a, 'b)\ IX \to ('a, 'b)\ IX \to ('a, 'b)\ IX$$

$$\forall is1\ is2 \bullet\ IxOverRide\ is1\ is2 = \lambda i \bullet\ if\ \neg\ i \in IxUd\ is2\ then\ is2\ i\ else\ is1\ i$$

$$\boldsymbol{ixud\_eq\_iff\_ixdom\_eq\_lemma} = \vdash \forall\ x\ y \bullet\ IxUd\ x = IxUd\ y \Leftrightarrow IxDom\ x = IxDom\ y$$

$$\boldsymbol{ixoverride\_ixdom\_lemma} = \vdash \forall\ x\ y \bullet\ IxDom\ (IxOverRide\ x\ y) = IxDom\ x \cup IxDom\ y$$

$$\boldsymbol{ixoverride\_ixud\_lemma} = \vdash \forall\ x\ y \bullet\ IxUd\ (IxOverRide\ x\ y) = IxUd\ x \setminus IxDom\ y$$

$$\boldsymbol{ixoverride\_ixran\_lemma} = \vdash \forall\ x\ y \bullet\ IxRan\ (IxOverRide\ x\ y) \subseteq IxRan\ x \cup IxRan\ y$$

HOL Constant

$$\boldsymbol{IxComp} : ('a, 'b)\ IX \to ('b \to 'c) \to ('a, 'c)\ IX$$

$$\forall ix\ j \bullet\ IxComp\ ix\ j = \lambda x \bullet\ if\ IsDefined\ (ix\ x)\ then\ Value\ (j\ (ValueOf\ (ix\ x)))\ else\ Undefined$$

IxDom_IxComp_thm =

$\vdash \forall\ is\ f\bullet\ IxDom\ (IxComp\ is\ f) = IxDom\ is$

HOL Constant

IxCompIx : $('a, 'b)\ IX \to ('b, 'c)\ IX \to ('a, 'c)\ IX$

$\forall ix\ jx\bullet\ IxCompIx\ ix\ jx = \lambda x\bullet$
$\quad if\ IsDefined\ (ix\ x)\ then\ jx\ (ValueOf\ (ix\ x))\ else\ Undefined$

IxDom_IxCompIx_thm =

$\vdash \forall\ is1\ is2\bullet\ IxDom\ (IxCompIx\ is1\ is2) \subseteq IxDom\ is1$

HOL Constant

IxInc : $('a, 'b)\ IX \to ('a, 'b)\ IX \to BOOL$

$\forall i\ j\bullet\ IxInc\ i\ j \Leftrightarrow \forall x\bullet\ IsDefined\ (i\ x) \Rightarrow j\ x = i\ x$

SML

```
declare_alias("⊑", ⌜IxInc⌝);
declare_infix(200, "⊑");
```

IxInc_trans_thm =

$\vdash \forall\ A\ B\ C\bullet\ A \sqsubseteq B \wedge B \sqsubseteq C \Rightarrow A \sqsubseteq C$

IxDom_⊑_thm =

$\vdash \forall\ A\ B\ s\bullet\ s \in IxDom\ A \wedge A \sqsubseteq B \Rightarrow s \in IxDom\ B$

⊑_IxVal_thm =

$\vdash \forall\ A\ B\ s\bullet\ s \in IxDom\ A \wedge A \sqsubseteq B \Rightarrow IxVal\ B\ s = IxVal\ A\ s$

HOL Constant

IxPack : $('a \times 'b)LIST \to ('a, 'b)IX$

$\quad IxPack\ [] = (\lambda is\bullet\ Undefined)$
$\wedge\ \forall h\ t\bullet\ IxPack\ (Cons\ h\ t) = \lambda is\bullet$
$\quad if\ Fst\ h = is\ then\ Value\ (Snd\ h)\ else\ IxPack\ t\ is$

IxDom_IxPack_disp_thm =

$\vdash$ *IxDom* (*IxPack* []) = {}
$\wedge$ ($\forall$ *x y z*• *IxDom* (*IxPack* (*Cons* (*x*, *y*) *z*)) = *Insert x* (*IxDom* (*IxPack z*)))

IxPack_lemma1 =

$\vdash$ $\forall$ *x y z*• *IxPack* (*Cons* (*x*, *y*) *z*) *x* = *Value y*

IxPack_lemma2 =

$\vdash$ $\forall$ *w x y z*• (*x*=*w* $\Leftrightarrow$ *F*) $\Rightarrow$ *IxPack* (*Cons* (*x*, *y*) *z*) *w* = *IxPack z w*

IxComp_IxPack_lemma =

$\vdash$ $\forall$ *f h t*• *IxComp* (*IxPack* (*Cons h t*)) *f*
= (λ *z*• *if z* = *Fst h then Value* (*f* (*Snd h*)) *else IxComp* (*IxPack t*) *f z*)

IxComp_IxPack_thm =

$\vdash$ $\forall$ *f l*• *IxComp* (*IxPack l*) *f* = *IxPack* (*Map* (λ (*x*, *y*)• (*x*, *f y*)) *l*)

SML

```
fun IxPack_conv t =
    let val (_, [c, w]) = strip_app t;
            val (_, [xy, z]) = strip_app c;
        val (x, y) = dest_pair xy;
        val thm =  list_∀_elim [w,x,y,z] IxPack_lemma2;
        val eq = mk_eq (x,w);
        val prem = string_eq_conv eq
    in  ⇒_elim thm prem
    end handle _ => fail_conv t;
```

SML

```
set_rw_eqn_cxt ([(⌜IxPack (Cons (x,y) z) w⌝, IxPack_conv)]) "'rbjmisc";
add_rw_thms (map get_spec [] @ [singleton_subset_lemma, insert_twice_thm]) "'rbjmisc";
add_rw_thms [NeSet_ne_thm] "'rbjmisc";
add_rw_thms [NeSet_PeSet_thm, MemOf_memof_thm, PeSet_Insert_thm, MemOf_NeSet_unit_thm] "'r
add_rw_thms (map get_spec [⌜IsDefined⌝, ⌜ValueOf⌝] @ [value_not_undefined_lemma, value_oneone_lem
add_rw_thms (map get_spec [⌜$∈_L⌝] @ [∀_L_clauses, ∃_L_clauses, ∀_L_thm, ∃_L_thm, ∀_L_append_thm, ∃_L_ap
add_rw_thms (map get_spec [] @ [RelProd_projections_thm]) "'rbjmisc";
add_rw_thms (map get_spec [] @ [IsL_IsR_lemma]) "'rbjmisc";
add_rw_thms (map get_spec [] @ [ixud_eq_iff_ixdom_eq_lemma, ixoverride_ixdom_lemma, ixoverride_ixu
add_rw_thms (map get_spec [] @ [IxDom_IxPack_disp_thm, IxPack_lemma1, IxDom_IxComp_thm]) "'rb
add_rw_thms (map get_spec [] @ [combc_thm, bincomp_thm]) "'rbjmisc";

add_pc_thms "'rbjmisc" (map get_spec [] @ [IxDom_IxPack_disp_thm, IxPack_lemma1]);
set_merge_pcs ["hol", "'rbjmisc"];

commit_pc "'rbjmisc";
```

```
|
|
|force_new_pc "rbjmisc";
|merge_pcs ["hol", "'rbjmisc"] "rbjmisc";
|commit_pc "rbjmisc";
|
|force_new_pc "rbjmisc1";
|merge_pcs ["'rbjmisc", "hol1"] "rbjmisc1";
|commit_pc "rbjmisc1";
```

17 The Theory rbjmisc

17.1 Parents

topology *group_egs* *equiv_rel* *analysis* *cache'rbjhol*

17.2 Constants

CombC	$('a \to 'b \to 'c) \to 'b \to 'a \to 'c$
BinComp	$('a \to 'b \to 'c) \to ('d \to 'a) \to ('e \to 'b) \to 'd \to 'e \to 'c$
ManyOne	$('a \to 'b \to BOOL) \to BOOL$
PDisj	$'a\ \mathbb{P}\ \mathbb{P} \to BOOL$
FunImage	$('a \to 'b) \to 'a\ \mathbb{P} \to 'b\ \mathbb{P}$
PeSet	$'a\ NESET \to 'a\ \mathbb{P}$
NeSet	$'a\ \mathbb{P} \to 'a\ NESET$
MemOf	$'a\ NESET \to 'a$
Undefined	$'a\ OPT$
Value	$('a, 'a)\ IX$
ValueOf	$'a\ OPT \to 'a$
IsDefined	$'a\ OPT \to BOOL$
$\$\in_{\boldsymbol{L}}$	$'a \to 'a\ LIST \to BOOL$
$\forall_{\boldsymbol{L}}$	$BOOL\ LIST \to BOOL$
$\exists_{\boldsymbol{L}}$	$BOOL\ LIST \to BOOL$
***MapCf*$_2$**	$('a \to 'b \to 'c) \to 'a\ LIST \to 'b\ LIST \to 'c\ LIST$
***MapCf*$_3$**	$('a \to 'b \to 'c \to 'd)$ $\to 'a\ LIST$ $\to 'b\ LIST$ $\to 'c\ LIST$ $\to 'd\ LIST$
Combine2	$'a\ LIST \to 'b\ LIST \to ('a \times 'b)\ LIST$
List2Set	$'a\ LIST \to 'a\ \mathbb{P}$
ListUnion	$'a\ \mathbb{P}\ LIST \to 'a\ \mathbb{P}$
ListFunUnion	$('a\ \mathbb{P} \to 'a\ \mathbb{P})\ LIST \to 'a\ \mathbb{P} \to 'a\ \mathbb{P}$
$\$..{}_{\boldsymbol{L}}$	$\mathbb{N} \to \mathbb{N} \to \mathbb{N}\ LIST$
SqrtA	$\mathbb{R} \to \mathbb{R}$
$\$\rightarrowtail>$	$('a \to \mathbb{R} + ONE) \to \mathbb{R} \to BOOL$
$\$\times_{\boldsymbol{D}}$	$'a\ \mathbb{P}\ \mathbb{P} \to 'b\ \mathbb{P}\ \mathbb{P} \to ('a \leftrightarrow 'b)\ \mathbb{P}$
$\mathbf{\Pi}_{\boldsymbol{f}}$	$'b\ \mathbb{P} \to ('b \to 'a\ \mathbb{P}) \to ('b \to 'a)\ \mathbb{P}$
$\times_{\mathbf{\Pi}}$	$'b\ \mathbb{P} \to 'a\ \mathbb{P}\ \mathbb{P} \to ('b \to 'a)\ \mathbb{P}\ \mathbb{P}$
$\$$***RelProd***	$'a\ \mathbb{P} \times ('a \to 'a \to BOOL)$ $\to 'b\ \mathbb{P} \times ('b \to 'b \to BOOL)$ $\to 'a \leftrightarrow 'b \times ('a \times 'b \to 'a \times 'b \to BOOL)$
RelPower	$'a\ \mathbb{P} \times ('a \to 'a \to BOOL)$ $\to 'b\ \mathbb{P}$ $\to ('b \to 'a)\ \mathbb{P} \times (('b \to 'a) \to ('b \to 'a) \to BOOL)$
GroupProduct	$'a\ GROUP \to 'b\ GROUP \to ('a \times 'b)\ GROUP$
AbelianGroup	$'a\ GROUP\ \mathbb{P}$
$\$$***BaseOf***	$'a\ \mathbb{P}\ \mathbb{P} \to 'a\ \mathbb{P}\ \mathbb{P} \to BOOL$
TopologyFrom	$'a\ \mathbb{P}\ \mathbb{P} \to 'a\ \mathbb{P}\ \mathbb{P}$
***Fun*$_+$**	$('a \to 't) \to ('b \to 'u) \to 'a + 'b \to 't + 'u$
FunSum	$('a \to 't) \to ('b \to 't) \to 'a + 'b \to 't$

$\boldsymbol{IxVal}$	$('b, 'a)\ IX \to 'b \to 'a$
$\boldsymbol{IxRan}$	$('b, 'a)\ IX \to 'a\ \mathbb{P}$
$\boldsymbol{IxDom}$	$('b, 'a)\ IX \to 'b\ \mathbb{P}$
$\boldsymbol{IxDomRes}$	$'a\ \mathbb{P} \to ('a, 'b)\ IX \to ('a, 'b)\ IX$
$\boldsymbol{IxRanRes}$	$('a, 'b)\ IX \to 'b\ \mathbb{P} \to ('a, 'b)\ IX$
$\boldsymbol{IxUd}$	$('a, 'b)\ IX \to 'a\ \mathbb{P}$
$\boldsymbol{IxOverRide}$	$('a, 'b)\ IX \to ('a, 'b)\ IX \to ('a, 'b)\ IX$
$\boldsymbol{IxComp}$	$('a, 'b)\ IX \to ('b \to 'c) \to ('a, 'c)\ IX$
$\boldsymbol{IxCompIx}$	$('a, 'b)\ IX \to ('b, 'c)\ IX \to ('a, 'c)\ IX$
$\boldsymbol{IxInc}$	$('a, 'b)\ IX \to ('a, 'b)\ IX \to BOOL$
$\boldsymbol{IxPack}$	$('a \times 'b)\ LIST \to ('a, 'b)\ IX$

17.3 Aliases

$*$	$GroupProduct : 'a\ GROUP \to 'b\ GROUP \to ('a \times 'b)\ GROUP$
$\lhd$	$IxDomRes : 'a\ \mathbb{P} \to ('a, 'b)\ IX \to ('a, 'b)\ IX$
$\rhd$	$IxRanRes : ('a, 'b)\ IX \to 'b\ \mathbb{P} \to ('a, 'b)\ IX$
$\sqsubseteq$	$IxInc : ('a, 'b)\ IX \to ('a, 'b)\ IX \to BOOL$

17.4 Types

$'1\ \boldsymbol{NESET}$
$'1\ \boldsymbol{OPT}$

17.5 Type Abbreviations

$('a, 'b)\ \boldsymbol{IX}$	$('a, 'b)\ IX$

17.6 Fixity

Right Infix 200:
$\sqsubseteq$
Right Infix 210:
$\boldsymbol{RelProd}$
Right Infix 230:
$\leqslant_{\boldsymbol{l}}\ \leqslant_{\boldsymbol{r}}$
Right Infix 300:
$\boldsymbol{BaseOf}\ \ ..{}_{\boldsymbol{L}}\ \in_{\boldsymbol{L}}\ \rightarrowtail\!\!\!\!\!\rightarrow >$
Right Infix 340:
$\times_{\boldsymbol{D}}$

17.7 Definitions

CombC $\vdash \forall\, f \bullet\ CombC\ f = (\lambda\, x\ y \bullet\ f\ y\ x)$

BinComp $\vdash \forall\, f\ g\ h \bullet\ BinComp\ f\ g\ h = (\lambda\, x\ y \bullet\ f\ (g\ x)\ (h\ y))$

ManyOne $\vdash \forall\, r \bullet\ ManyOne\ r \Leftrightarrow (\forall\, x\ y\ z \bullet\ r\ x\ y \wedge r\ x\ z \Rightarrow y = z)$

PDisj $\vdash \forall\, ss$
$\bullet\ PDisj\ ss$
$\Leftrightarrow \neg\ (\exists\, t\ u \bullet\ \{t; u\} \subseteq ss \wedge \neg\ t = u \wedge \neg\ t \cap u = \{\})$

FunImage $\vdash \forall\, f\ A \bullet\ FunImage\ f\ A = \{b | \exists\, a \bullet\ a \in A \wedge f\ a = b\}$

NESET $\vdash \exists\, f \bullet\ TypeDefn\ (\lambda\, y \bullet\ \exists\, z \bullet\ z \in y)\ f$

NeSet

PeSet $\vdash (\forall\, x \bullet\ \exists\, y \bullet\ y \in PeSet\ x)$
$\wedge\ (\forall\, x\ y \bullet\ x = y \Leftrightarrow (\forall\, z \bullet\ z \in PeSet\ x \Leftrightarrow z \in PeSet\ y))$
$\wedge\ (\forall\, x\ y \bullet\ x \in y \Rightarrow PeSet\ (NeSet\ y) = y)$
$\wedge\ (\forall\, y \bullet\ NeSet\ (PeSet\ y) = y)$

MemOf $\vdash \forall\, x \bullet\ MemOf\ x = (\epsilon\, y \bullet\ y \in PeSet\ x)$

OPT $\vdash \exists\, f \bullet\ TypeDefn\ (\lambda\, y \bullet\ T)\ f$

Value

Undefined $\vdash\ OneOne\ Value$
$\wedge\ (\forall\, x \bullet\ \neg\ Value\ x = Undefined)$
$\wedge\ (\forall\, y \bullet\ y = Undefined \vee (\exists\, z \bullet\ y = Value\ z))$

ValueOf $\vdash \forall\, v \bullet\ ValueOf\ (Value\ v) = v$

IsDefined $\vdash \forall\, v \bullet\ IsDefined\ v \Leftrightarrow \neg\ v = Undefined$

$\in_L$ $\vdash \forall\, a\ b\ al$
$\bullet\ (a \in_L [] \Leftrightarrow F)$
$\wedge\ (a \in_L Cons\ b\ al \Leftrightarrow a = b \vee a \in_L al)$

$\forall_L$ $\vdash \forall\, bl \bullet\ \forall_L\ bl \Leftrightarrow Fold\ \$\wedge\ bl\ T$

$\exists_L$ $\vdash \forall\, bl \bullet\ \exists_L\ bl \Leftrightarrow Fold\ \$\vee\ bl\ F$

MapCf$_2$ $\vdash \forall\, cf\ al\ bl$
$\bullet\ MapCf_2\ cf\ al\ bl = Map\ (Uncurry\ cf)\ (Combine\ al\ bl)$

MapCf$_3$ $\vdash \forall\, cf\ al\ bl\ cl$
$\bullet\ MapCf_3\ cf\ al\ bl\ cl$
$= Map$
$(Uncurry\ (Uncurry\ cf))$
$(Combine\ (Combine\ al\ bl)\ cl)$

Combine2 $\vdash\ ConstSpec$
$(\lambda\, Combine2'$
$\bullet\ (\forall\, b \bullet\ Combine2'\ []\ b = [])$
$\wedge\ (\forall\, a \bullet\ Combine2'\ a\ [] = [])$
$\wedge\ (\forall\, ha\ ta\ hb\ tb$
$\bullet\ Combine2'\ (Cons\ ha\ ta)\ (Cons\ hb\ tb)$
$= Cons\ (ha, hb)\ (Combine2'\ ta\ tb)))$
$Combine2$

List2Set $\vdash \forall\, l \bullet\ List2Set\ l = \{e | e \in_L l\}$

ListUnion $\vdash \forall\, l \bullet\ ListUnion\ l = \bigcup\ (List2Set\ l)$

ListFunUnion $\vdash \forall\, l\ as$
$\bullet\ ListFunUnion\ l\ as = ListUnion\ (Map\ (\lambda\, f \bullet\ f\ as)\ l)$

..$_L$ $\vdash \forall\, x\ y$
$\bullet\ x\ .._L\ 0 = []$
$\wedge\ x\ .._L\ y + 1$
$= (if\ x \leqslant y\ then\ (x\ .._L\ y)\ @\ [y + 1]\ else\ [])$

SqrtA $\vdash \forall\, x \bullet\ 0. \leqslant SqrtA\ x \wedge SqrtA\ x \hat{}\ 2 = Abs\ x$

$\rightarrowtail\!\!>>$ $\vdash \forall\ c\ r$
• $c \rightarrowtail\!\!>> r$
$\Leftrightarrow (\exists\ s$
• $(\forall\ a\ n\ m \bullet\ IsL\ (c\ a) \Rightarrow s\ n = s\ m \Rightarrow n = m)$
$\Rightarrow Series$
$(\lambda\ n$
• $if\ IsR\ (c\ (s\ n))$
$then\ 0_R$
$else\ OutL\ (c\ (s\ n)))$
$-> r)$

$\times_{\boldsymbol{D}}$ $\vdash \forall\ l\ r$
• $l \times_D r$
$= \{cp$
$|\exists\ leq\ req \bullet\ leq \in l \wedge req \in r \wedge cp = (leq \times req)\}$

$\boldsymbol{\Pi_f}$ $\vdash \forall\ is\ f \bullet\ \Pi_f\ is\ f = \{g | \forall\ i \bullet\ i \in is \Rightarrow g\ i \in f\ i\}$

$\times_{\boldsymbol{\Pi}}$ $\vdash \forall\ is\ ss$
• $\times_\Pi\ is\ ss$
$= \{g$
$|\exists\ h \bullet\ (\forall\ i \bullet\ i \in is \Rightarrow h\ i \in ss) \wedge g = \Pi_f\ is\ h\}$

RelProd $\vdash \forall\ \$\leqslant_l\ \$\leqslant_r\ L\ R$
• $(L, \$\leqslant_l)\ RelProd\ (R, \$\leqslant_r)$
$= (L \times R,$
$(\lambda\ (l1, r1)\ (l2, r2)$
• $l1 \in L$
$\wedge\ l2 \in L$
$\wedge\ r1 \in R$
$\wedge\ r2 \in R$
$\wedge\ l1 \leqslant_l l2$
$\wedge\ r1 \leqslant_r r2))$

RelPower $\vdash \forall\ D\ r\ is$
• $RelPower\ (D, r)\ is$
$= (\{f | \forall\ i \bullet\ i \in is \Rightarrow f\ i \in D\},$
$(\lambda\ f\ g \bullet\ \forall\ i \bullet\ i \in is \Rightarrow r\ (f\ i)\ (g\ i)))$

GroupProduct $\vdash \forall\ G\ H$
• $G * H$
$= (let\ car = (Car\ G \times Car\ H)$
$and\ prod\ (la, lb)\ (ra, rb)$
$= ((la\ .\ ra)\ G, (lb\ .\ rb)\ H)$
$and\ unit = (Unit\ G, Unit\ H)$
$and\ inv\ (a, b) = ((a\ \sim)\ G, (b\ \sim)\ H)$
$in\ MkGROUP\ car\ prod\ unit\ inv)$

AbelianGroup $\vdash \forall\ G$
• $G \in AbelianGroup$
$\Leftrightarrow G \in Group$
$\wedge\ (\forall\ u\ v$
• $u \in Car\ G \wedge v \in Car\ G$
$\Rightarrow (u\ .\ v)\ G = (v\ .\ u)\ G)$

BaseOf $\vdash \forall\ base\ topology$
• $base\ BaseOf\ topology$
$\Leftrightarrow (\forall\ s$
• $s \in topology \Rightarrow (\exists\ ss \bullet\ ss \subseteq base \wedge s = \bigcup ss))$

TopologyFrom $\vdash \forall\ sets$
$\bullet\ TopologyFrom\ sets$
$= \bigcap$
$\{topology$
$\mid topology \in Topology \wedge sets \subseteq topology\}$

Fun$_+$ $\vdash \forall\ f\ g\ ab$
$\bullet\ Fun_+\ f\ g\ ab$
$= (if\ IsL\ ab$
$then\ InL\ (f\ (OutL\ ab))$
$else\ InR\ (g\ (OutR\ ab)))$

FunSum $\vdash \forall\ f\ g\ ab$
$\bullet\ FunSum\ f\ g\ ab$
$= (if\ IsL\ ab\ then\ f\ (OutL\ ab)\ else\ g\ (OutR\ ab))$

IxVal $\vdash \forall\ is\ g\bullet\ IxVal\ is\ g = ValueOf\ (is\ g)$

IxRan $\vdash \forall\ is\bullet\ IxRan\ is = \{v \mid \exists\ \alpha\bullet\ Value\ v = is\ \alpha\}$

IxDom $\vdash \forall\ is\bullet\ IxDom\ is = \{i \mid IsDefined\ (is\ i)\}$

IxDomRes $\vdash \forall\ ns\ is$
$\bullet\ ns \lhd is = (\lambda\ x\bullet\ if\ x \in ns\ then\ is\ x\ else\ Undefined)$

IxRanRes $\vdash \forall\ is\ ns$
$\bullet\ is \rhd ns$
$= (\lambda\ x$
$\bullet\ if\ x \in IxDom\ is \wedge ValueOf\ (is\ x) \in ns$
$then\ is\ x$
$else\ Undefined)$

IxUd $\vdash \forall\ is\bullet\ IxUd\ is = \{i \mid is\ i = Undefined\}$

IxOverRide $\vdash \forall\ is1\ is2$
$\bullet\ IxOverRide\ is1\ is2$
$= (\lambda\ i\bullet\ if\ \neg\ i \in IxUd\ is2\ then\ is2\ i\ else\ is1\ i)$

IxComp $\vdash \forall\ ix\ j$
$\bullet\ IxComp\ ix\ j$
$= (\lambda\ x$
$\bullet\ if\ IsDefined\ (ix\ x)$
$then\ Value\ (j\ (ValueOf\ (ix\ x)))$
$else\ Undefined)$

IxCompIx $\vdash \forall\ ix\ jx$
$\bullet\ IxCompIx\ ix\ jx$
$= (\lambda\ x$
$\bullet\ if\ IsDefined\ (ix\ x)$
$then\ jx\ (ValueOf\ (ix\ x))$
$else\ Undefined)$

IxInc $\vdash \forall\ i\ j\bullet\ i \sqsubseteq j \Leftrightarrow (\forall\ x\bullet\ IsDefined\ (i\ x) \Rightarrow j\ x = i\ x)$

IxPack $\vdash IxPack\ [] = (\lambda\ is\bullet\ Undefined)$
$\wedge\ (\forall\ h\ t$
$\bullet\ IxPack\ (Cons\ h\ t)$
$= (\lambda\ is$
$\bullet\ if\ Fst\ h = is$
$then\ Value\ (Snd\ h)$
$else\ IxPack\ t\ is))$

17.8 Theorems

combc_thm $\vdash \forall\ f\ x\ y\bullet\ CombC\ f\ x\ y = f\ y\ x$

bincomp_thm $\vdash \forall\ f\ g\ h\ x\ y \bullet\ BinComp\ f\ g\ h\ x\ y = f\ (g\ x)\ (h\ y)$

$\forall\_\eta\_lemma$ $\vdash \forall\ p \bullet\ \$\forall\ p \Leftrightarrow (\forall\ x \bullet\ p\ x)$

$\forall\_\wedge\_out\_lemma$

$\vdash \forall\ p\ q \bullet\ \$\forall\ p\ \wedge\ \$\forall\ q \Leftrightarrow (\forall\ x \bullet\ p\ x\ \wedge\ q\ x)$

type_lemmas_thm2

$\vdash \forall\ pred$
$\bullet\ (\exists\ f \bullet\ TypeDefn\ pred\ f)$
$\Rightarrow (\exists\ abs\ rep$
$\bullet\ (\forall\ a \bullet\ abs\ (rep\ a) = a)$
$\wedge\ (\forall\ r \bullet\ pred\ r \Leftrightarrow rep\ (abs\ r) = r)$
$\wedge\ OneOne\ rep)$

type_defn_lemma1

$\vdash \forall\ f\ g$
$\bullet\ (\forall\ x \bullet\ f\ (g\ x) = x) \Rightarrow (\forall\ x\ y \bullet\ g\ x = g\ y \Rightarrow x = y)$

type_defn_lemma2

$\vdash \forall\ p\ f\ g$
$\bullet\ (\forall\ x \bullet\ p\ x \Rightarrow f\ (g\ x) = x)$
$\Rightarrow (\forall\ x\ y \bullet\ p\ x\ \wedge\ p\ y \Rightarrow g\ x = g\ y \Rightarrow x = y)$

type_defn_lemma3

$\vdash (\exists\ f \bullet\ TypeDefn\ (\lambda\ x \bullet\ T)\ f)$
$\Rightarrow (\exists\ abs\ rep$
$\bullet\ (\forall\ a \bullet\ abs\ (rep\ a) = a)\ \wedge\ (\forall\ r \bullet\ rep\ (abs\ r) = r))$

type_defn_lemma4

$\vdash \forall\ pred$
$\bullet\ (\exists\ f \bullet\ TypeDefn\ pred\ f)$
$\Rightarrow (\exists\ abs\ rep$
$\bullet\ (\forall\ a \bullet\ abs\ (rep\ a) = a)$
$\wedge\ (\forall\ r \bullet\ pred\ r \Leftrightarrow rep\ (abs\ r) = r)$
$\wedge\ OneOne\ rep$
$\wedge\ (\forall\ a \bullet\ pred\ (rep\ a)))$

oneone_contrapos_lemma

$\vdash \forall\ f \bullet\ OneOne\ f \Rightarrow (\forall\ x\ y \bullet\ \neg\ x = y \Rightarrow \neg\ f\ x = f\ y)$

$\subseteq\_trans\_thm$ $\vdash \forall\ A\ B\ C \bullet\ A \subseteq B\ \wedge\ B \subseteq C \Rightarrow A \subseteq C$

singleton_subset_lemma

$\vdash \forall\ V\ x \bullet\ \{x\} \subseteq V \Leftrightarrow x \in V$

FunImage_o_thm

$\vdash \forall\ A\ f\ g$
$\bullet\ FunImage\ (f\ o\ g)\ A = FunImage\ f\ (FunImage\ g\ A)$

FunImage_mono_thm

$\vdash \forall\ A\ B\ f \bullet\ A \subseteq B \Rightarrow FunImage\ f\ A \subseteq FunImage\ f\ B$

insert_com_thm

$\vdash \forall\ x\ y\ z$
$\bullet\ Insert\ x\ (Insert\ y\ z) = Insert\ y\ (Insert\ x\ z)$

insert_twice_thm

$\vdash \forall\ x\ y \bullet\ Insert\ x\ (Insert\ x\ y) = Insert\ x\ y$

$\in\_disp\_\Rightarrow\_thm$ $\vdash \forall\ p\ d\ s$
$\bullet\ (\forall\ e \bullet\ e \in Insert\ d\ s \Rightarrow p\ e)$
$\Leftrightarrow p\ d\ \wedge\ (\forall\ e \bullet\ e \in s \Rightarrow p\ e)$

PeSet_Insert_thm

$\vdash \forall\ x\ y \bullet\ PeSet\ (NeSet\ (Insert\ x\ y)) = Insert\ x\ y$

MemOf_memof_thm

$\vdash \forall\ x \bullet\ MemOf\ x \in PeSet\ x$

MemOf_NeSet_unit_thm

$\vdash \forall\ x \bullet\ MemOf\ (NeSet\ \{x\}) = x$

cantors_thm $\vdash \neg\ (\exists\ f \bullet \forall\ s \bullet \exists\ e \bullet\ f\ e = s)$

opt_cases_thm

$\vdash \forall\ x \bullet\ x = Undefined \vee (\exists\ y \bullet\ x = Value\ y)$

value_not_undefined_lemma

$\vdash \forall\ x \bullet \neg\ Value\ x = Undefined \wedge \neg\ Undefined = Value\ x$

value_oneone_lemma

$\vdash \forall\ x\ y \bullet\ Value\ x = Value\ y \Leftrightarrow x = y$

$\forall_L$***-thm*** $\vdash (\forall_L\ [] \Leftrightarrow T)$
$\wedge\ (\forall\ h\ list \bullet\ \forall_L\ (Cons\ h\ list) \Leftrightarrow h \wedge \forall_L\ list)$

$\exists_L$***-thm*** $\vdash (\exists_L\ [] \Leftrightarrow F)$
$\wedge\ (\forall\ h\ list \bullet\ \exists_L\ (Cons\ h\ list) \Leftrightarrow h \vee \exists_L\ list)$

$\forall_L$***-clauses*** $\vdash (\forall_L\ [] \Leftrightarrow T)$
$\wedge\ (\forall\ t \bullet\ \forall_L\ (Cons\ T\ t) \Leftrightarrow \forall_L\ t)$
$\wedge\ (\forall\ t \bullet\ \forall_L\ (Cons\ F\ t) \Leftrightarrow F)$
$\wedge\ (\forall\ h \bullet\ \forall_L\ [h] \Leftrightarrow h)$

$\exists_L$***-clauses*** $\vdash (\exists_L\ [] \Leftrightarrow F)$
$\wedge\ (\forall\ t \bullet\ \exists_L\ (Cons\ F\ t) \Leftrightarrow \exists_L\ t)$
$\wedge\ (\forall\ t \bullet\ \exists_L\ (Cons\ T\ t) \Leftrightarrow T)$
$\wedge\ (\forall\ h \bullet\ \exists_L\ [h] \Leftrightarrow h)$
$\wedge\ (\forall\ h \bullet\ \exists_L\ [h] \Leftrightarrow h)$

$\forall_L$***-append_thm***

$\vdash \forall\ l\ m \bullet\ \forall_L\ (l\ @\ m) \Leftrightarrow \forall_L\ l \wedge \forall_L\ m$

$\exists_L$***-append_thm***

$\vdash \forall\ l\ m \bullet\ \exists_L\ (l\ @\ m) \Leftrightarrow \exists_L\ l \vee \exists_L\ m$

ListUnion $\vdash ListUnion\ [] = \{\}$
$\wedge\ (\forall\ h\ t \bullet\ ListUnion\ (Cons\ h\ t) = h \cup ListUnion\ t)$

prim_rec_thm2

$\vdash \forall\ z\ s \bullet \exists\ f \bullet\ f\ 0 = z \wedge (\forall\ n \bullet\ f\ (n + 1) = s\ (f\ n)\ n)$

$\mathbb{R}$***_prod_sign_iff_clauses***

$\vdash (\forall\ x\ y$
$\bullet\ 0. < x * y \Leftrightarrow 0. < x \wedge 0. < y \vee x < 0. \wedge y < 0.)$
$\wedge\ (\forall\ x\ y$
$\bullet\ x * y < 0. \Leftrightarrow 0. < x \wedge y < 0. \vee x < 0. \wedge 0. < y)$
$\wedge\ (\forall\ x\ y$
$\bullet\ 0. \leqslant x * y \Leftrightarrow 0. \leqslant x \wedge 0. \leqslant y \vee x \leqslant 0. \wedge y \leqslant 0.)$
$\wedge\ (\forall\ x\ y$
$\bullet\ x * y \leqslant 0. \Leftrightarrow 0. \leqslant x \wedge y \leqslant 0. \vee x \leqslant 0. \wedge 0. \leqslant y)$
$\wedge\ (\forall\ x\ y \bullet\ x * y = 0. \Leftrightarrow x = 0. \vee y = 0.)$
$\wedge\ (\forall\ x\ y \bullet\ 0. = x * y \Leftrightarrow x = 0. \vee y = 0.)$

$\mathbb{R}$***_times_mono_thm1***

$\vdash \forall\ x\ y\ z \bullet\ 0. < x \wedge y < z \Rightarrow y * x < z * x$

$\mathbb{R}$***_times_mono_thm2***

$\vdash \forall\ x\ y\ z \bullet\ 0. \leqslant x \wedge y \leqslant z \Rightarrow x * y \leqslant x * z$

$\mathbb{R}$***_times_mono_thm3***

$\vdash \forall\ x\ y\ z \bullet\ 0. \leqslant x \wedge y \leqslant z \Rightarrow y * x \leqslant z * x$

$\mathbb{R}$***_times_mono_thm4***

$\vdash \forall\ w\ x\ y\ z$
$\bullet\ 0. < w \wedge 0. < y \wedge w < x \wedge y < z \Rightarrow w * y < x * z$

$\mathbb{R}$_**times_mono_thm5**
$$\vdash \forall\ w\ x\ y\ z \bullet\ 0. \leqslant w \wedge 0. \leqslant y \wedge w \leqslant x \wedge y \leqslant z \Rightarrow w * y \leqslant x * z$$
$\mathbb{R}$_**square_mono_thm**
$$\vdash \forall\ x\ y \bullet\ 0. < x \wedge x < y \Rightarrow x * x < y * y$$
$\mathbb{R}$_**square_mono_thm1**
$$\vdash \forall\ x\ y \bullet\ 0. \leqslant x \wedge x \leqslant y \Rightarrow x * x \leqslant y * y$$
$\mathbb{R}$_**square_less_less_thm**
$$\vdash \forall\ x\ y\ z \bullet\ 0. < x \wedge 0. < y \wedge x * x < y * y \Rightarrow x < y$$
$\mathbb{R}$_**square_eq_thm1**
$$\vdash \forall\ x\ y \bullet\ 0. < x \wedge 0. < y \wedge y * y = x * x \Rightarrow x = y$$
$\mathbb{R}$_**square_eq_thm2**
$$\vdash \forall\ x\ y \bullet\ 0. \leqslant x \wedge 0. \leqslant y \wedge y * y = x * x \Rightarrow x = y$$
$\mathbb{R}$_**square_eq_eq_thm**
$$\vdash \forall\ x\ y \bullet\ 0. < x \wedge 0. < y \Rightarrow (x * x = y * y \Leftrightarrow x = y)$$
$\mathbb{R}$_**square_eq_eq_thm2**
$$\vdash \forall\ x\ y \bullet\ 0. \leqslant x \wedge 0. \leqslant y \Rightarrow (x * x = y * y \Leftrightarrow x = y)$$
$\mathbb{R}$_**square_**$\leqslant$**_**$\leqslant$**_thm**
$$\vdash \forall\ x\ y \bullet\ 0. \leqslant x \wedge 0. \leqslant y \wedge x * x \leqslant y * y \Rightarrow x \leqslant y$$
$\mathbb{R}$_**square_**$\leqslant$**_iff_**$\leqslant$**_thm**
$$\vdash \forall\ x\ y \bullet\ 0. \leqslant x \wedge 0. \leqslant y \Rightarrow (x * x \leqslant y * y \Leftrightarrow x \leqslant y)$$
$\mathbb{R}$_**square_eq_iff_abs_eq_thm**
$$\vdash \forall\ x\ y \bullet\ x * x = y * y \Leftrightarrow Abs\ x = Abs\ y$$
$\mathbb{R}$_**square_pos_thm**
$$\vdash \forall\ x \bullet\ 0. \leqslant x \hat{}\ 2$$
$\mathbb{R}$_**sum_pos_thm**
$$\vdash \forall\ x\ y \bullet\ 0. \leqslant x \wedge 0. \leqslant y \Rightarrow 0. \leqslant x + y$$
$\mathbb{R}$_**sum_square_pos_thm**
$$\vdash \forall\ x\ y \bullet\ 0. \leqslant x \hat{}\ 2 + y \hat{}\ 2$$
$\mathbb{R}$_**sum_square_zero_thm**
$$\vdash \forall\ x\ y \bullet\ x \hat{}\ 2 + y \hat{}\ 2 = 0. \Leftrightarrow x = 0. \wedge y = 0.$$
$\mathbb{R}$_**sum_zero_thm**
$$\vdash \forall\ x\ y \bullet\ 0. \leqslant x \wedge 0. \leqslant y \Rightarrow (x + y = 0. \Leftrightarrow x = 0. \wedge y = 0.)$$
$\mathbb{R}$_**abs_sum_square_thm**
$$\vdash \forall\ x\ y \bullet\ Abs\ (x \hat{}\ 2 + y \hat{}\ 2) = x \hat{}\ 2 + y \hat{}\ 2$$
$\mathbb{R}$_**plus_mono_thm**
$$\vdash \forall\ u\ v\ x\ y \bullet\ u \leqslant v \wedge x \leqslant y \Rightarrow u + x \leqslant v + y$$
$\mathbb{R}$_**Abs_Norm_clauses**
$$\begin{aligned}\vdash\ & (\forall\ v \bullet\ 0. \leqslant Abs\ v)\\ & \wedge (\forall\ v \bullet\ Abs\ v = 0. \Leftrightarrow v = 0.)\\ & \wedge (\forall\ x\ v \bullet\ Abs\ (x * v) = Abs\ x * Abs\ v)\\ & \wedge (\forall\ v\ w \bullet\ Abs\ (v + w) \leqslant Abs\ v + Abs\ w)\end{aligned}$$
$\mathbb{R}$_**$\leqslant$_abs_thm** $\vdash \forall\ x \bullet\ x \leqslant Abs\ x$
$\mathbb{R}$_**abs_pos_id_thm**
$$\vdash \forall\ x \bullet\ 0. \leqslant x \Rightarrow Abs\ x = x$$
$\mathbb{R}$_**abs_mono_thm**
$$\vdash \forall\ x\ y \bullet\ 0. \leqslant x \wedge x \leqslant y \Rightarrow Abs\ x \leqslant Abs\ y$$
$\mathbb{R}$_**square_eq_abs_thm**
$$\vdash \forall\ x\ y \bullet\ x \hat{}\ 2 = y \hat{}\ 2 \Leftrightarrow Abs\ x = Abs\ y$$
$\mathbb{R}$_**abs_square_thm1**

$\vdash \forall\ x\bullet\ Abs\ (x * x) = x * x$

$\mathbb{R}$***_square_⩽_abs_⩽_thm***

$\vdash \forall\ x\ y\bullet\ x * x \leqslant y * y \Rightarrow Abs\ x \leqslant Abs\ y$

$\mathbb{R}$***_abs_⩽_square_⩽_thm***

$\vdash \forall\ x\ y\bullet\ Abs\ x \leqslant Abs\ y \Rightarrow x * x \leqslant y * y$

$\mathbb{R}$***_square_⩽_iff_abs_⩽_thm***

$\vdash \forall\ x\ y\bullet\ x * x \leqslant y * y \Leftrightarrow Abs\ x \leqslant Abs\ y$

$\mathbb{R}$***_square_less_iff_abs_less_thm***

$\vdash \forall\ x\ y\bullet\ x * x < y * y \Leftrightarrow Abs\ x < Abs\ y$

$\mathbb{R}$***_abs_square_thm2***

$\vdash \forall\ x\bullet\ Abs\ (x \hat{}\ 2) = x \hat{}\ 2$

$\mathbb{R}$***_abs_prod_thm***

$\vdash \forall\ x\ y\bullet\ Abs\ (x * y) = Abs\ x * Abs\ y$

sqrt_thm1 $\vdash \forall\ x\bullet\ SqrtA\ x = 0. \Leftrightarrow x = 0.$

sqrt_square_thm

$\vdash \forall\ x\ y\bullet\ SqrtA\ (x \hat{}\ 2 + y \hat{}\ 2) = 0. \Leftrightarrow x = 0. \wedge y = 0.$

$\mathbb{R}$***_sqrt_minus_thm***

$\vdash \forall\ x\bullet\ SqrtA\ (\sim x) = SqrtA\ x$

$\mathbb{R}$***_abs_clauses***

$\vdash \forall\ x$
- $SqrtA\ (Abs\ x) = SqrtA\ x$
  $\wedge\ Abs\ (SqrtA\ x) = SqrtA\ x$
  $\wedge\ Abs\ (Abs\ x) = Abs\ x$
  $\wedge\ Abs\ 0. = 0.$
  $\wedge\ Abs\ (\sim x) = Abs\ x$

$\mathbb{R}$***_sqrt_abs_thm***

$\vdash \forall\ x\ y\bullet\ SqrtA\ x = SqrtA\ y \Leftrightarrow Abs\ x = Abs\ y$

$\mathbb{R}$***_sqrt_mono_thm***

$\vdash \forall\ x\ y\bullet\ 0. \leqslant x \wedge x \leqslant y \Rightarrow SqrtA\ x \leqslant SqrtA\ y$

$\mathbb{R}$***_sqrt_square_thm1***

$\vdash \forall\ x\bullet\ SqrtA\ (x * x) = Abs\ x$

$\mathbb{R}$***_sqrt_square_thm2***

$\vdash \forall\ x\bullet\ SqrtA\ (x \hat{}\ 2) = Abs\ x$

$\mathbb{R}$***_sqrt_prod_thm***

$\vdash \forall\ x\ y\bullet\ SqrtA\ (x * y) = SqrtA\ x * SqrtA\ y$

sqrt_plus_thm

$\vdash \forall\ x\ y\bullet\ SqrtA\ (x + y) \leqslant SqrtA\ x + SqrtA\ y$

cp_eq_thm1 $\vdash \forall\ x\ y\ v\ w\ p\ q$
- $p \in x \wedge q \in y \Rightarrow (x \times y) = (v \times w) \Rightarrow x = v \wedge y = w$

cp_eq_thm2 $\vdash \forall\ x\ y\ v\ w\ p$
- $p \in (x \times y) \Rightarrow (x \times y) = (v \times w) \Rightarrow x = v \wedge y = w$

cp_l_part_thm

$\vdash \forall\ l\ r\ x\bullet\ x \in r \Rightarrow l = \{m | \exists\ p\bullet\ p \in (l \times r) \wedge m = Fst\ p\}$

cp_r_part_thm

$\vdash \forall\ l\ r\ x\bullet\ x \in l \Rightarrow r = \{m | \exists\ p\bullet\ p \in (l \times r) \wedge m = Snd\ p\}$

cp_part_thm $\vdash \forall\ l\ r\ x$
- $x \in (l \times r)$
  $\Rightarrow l = \{m | \exists\ p\bullet\ p \in (l \times r) \wedge m = Fst\ p\}$
  $\wedge\ r = \{m | \exists\ p\bullet\ p \in (l \times r) \wedge m = Snd\ p\}$

$\times_{D}$***_ext_thm*** $\vdash \forall\ l\ r\ cp$
- $cp \in l \times_D r$

$\Leftrightarrow (\exists\ leq\ req$
$\bullet\ leq \in l \wedge req \in r \wedge cp = (leq \times req))$

RelProd_projections_thm

$\vdash \forall\ (L, \$\leqslant_l)\ (R, \$\leqslant_r)$
$\bullet\ Fst\ ((L, \$\leqslant_l)\ RelProd\ (R, \$\leqslant_r)) = (L \times R)$
$\wedge\ Snd\ ((L, \$\leqslant_l)\ RelProd\ (R, \$\leqslant_r))$
$= (\lambda\ (l1, r1)\ (l2, r2)$
$\bullet\ l1 \in L$
$\wedge\ l2 \in L$
$\wedge\ r1 \in R$
$\wedge\ r2 \in R$
$\wedge\ l1 \leqslant_l l2$
$\wedge\ r1 \leqslant_r r2)$

Trans_RelProd_thm

$\vdash \forall\ (L, \widehat{=}_l)\ (R, \widehat{=}_r)$
$\bullet\ Trans\ (L, \widehat{=}_l) \wedge Trans\ (R, \widehat{=}_r)$
$\Rightarrow Trans\ ((L, \widehat{=}_l)\ RelProd\ (R, \widehat{=}_r))$

Sym_RelProd_thm

$\vdash \forall\ (L, \widehat{=}_l)\ (R, \widehat{=}_r)$
$\bullet\ Sym\ (L, \widehat{=}_l) \wedge Sym\ (R, \widehat{=}_r)$
$\Rightarrow Sym\ ((L, \widehat{=}_l)\ RelProd\ (R, \widehat{=}_r))$

Refl_RelProd_thm

$\vdash \forall\ (L, \widehat{=}_l)\ (R, \widehat{=}_r)$
$\bullet\ Refl\ (L, \widehat{=}_l) \wedge Refl\ (R, \widehat{=}_r)$
$\Rightarrow Refl\ ((L, \widehat{=}_l)\ RelProd\ (R, \widehat{=}_r))$

RelPower_Trans_thm

$\vdash \forall\ (D, r)\ is$
$\bullet\ Trans\ (D, r) \Rightarrow Trans\ (RelPower\ (D, r)\ is)$

RelPower_Sym_thm

$\vdash \forall\ (D, r)\ is \bullet\ Sym\ (D, r) \Rightarrow Sym\ (RelPower\ (D, r)\ is)$

RelPower_Refl_thm

$\vdash \forall\ (D, r)\ is \bullet\ Refl\ (D, r) \Rightarrow Refl\ (RelPower\ (D, r)\ is)$

group_product_thm

$\vdash \forall\ g\ h \bullet\ g \in Group \wedge h \in Group \Rightarrow g * h \in Group$

abelian_group_product_lemma

$\vdash \forall\ g\ h$
$\bullet\ g \in AbelianGroup$
$\wedge\ h \in AbelianGroup$
$\wedge\ Car\ g = Universe$
$\wedge\ Car\ h = Universe$
$\Rightarrow Car\ (g * h) = Universe$

abelian_group_product_thm

$\vdash \forall\ g\ h$
$\bullet\ g \in AbelianGroup \wedge h \in AbelianGroup$
$\Rightarrow g * h \in AbelianGroup$

group_prod_prod_thm

$\vdash \forall\ G\ H\ x\ y\ v\ w$
$\bullet\ ((x, v)\ .\ (y, w))\ (G * H) = ((x\ .\ y)\ G, (v\ .\ w)\ H)$

group_prod_prod_thm1

$\vdash \forall\ G\ H\ x\ y$
$\bullet\ (x\ .\ y)\ (G * H)$

$= ((Fst\ x\ .\ Fst\ y)\ G,\ (Snd\ x\ .\ Snd\ y)\ H)$

$\mathbb{R}$_plus_abelian_thm

$\vdash \mathbb{R}_+ \in AbelianGroup$

IsL_IsR_lemma

$\vdash \forall\ x \bullet\ IsR\ x \Leftrightarrow \neg\ IsL\ x$

ix_domran_lemma

$\vdash \forall\ x\ y \bullet\ x \in IxDom\ y \Rightarrow IxVal\ y\ x \in IxRan\ y$

ix_valueof_ran_lemma

$\vdash \forall\ x\ y \bullet\ \neg\ x\ y = Undefined \Rightarrow ValueOf\ (x\ y) \in IxRan\ x$

ixud_eq_iff_ixdom_eq_lemma

$\vdash \forall\ x\ y \bullet\ IxUd\ x = IxUd\ y \Leftrightarrow IxDom\ x = IxDom\ y$

ixoverride_ixdom_lemma

$\vdash \forall\ x\ y \bullet\ IxDom\ (IxOverRide\ x\ y) = IxDom\ x \cup IxDom\ y$

ixoverride_ixran_lemma

$\vdash \forall\ x\ y \bullet\ IxRan\ (IxOverRide\ x\ y) \subseteq IxRan\ x \cup IxRan\ y$

ixoverride_ixud_lemma

$\vdash \forall\ x\ y \bullet\ IxUd\ (IxOverRide\ x\ y) = IxUd\ x \setminus IxDom\ y$

IxDom_IxComp_thm

$\vdash \forall\ is\ f \bullet\ IxDom\ (IxComp\ is\ f) = IxDom\ is$

IxDom_IxCompIx_thm

$\vdash \forall\ is1\ is2 \bullet\ IxDom\ (IxCompIx\ is1\ is2) \subseteq IxDom\ is1$

IxInc_trans_thm

$\vdash \forall\ A\ B\ C \bullet\ A \sqsubseteq B \wedge B \sqsubseteq C \Rightarrow A \sqsubseteq C$

IxDom_⊑_thm $\vdash \forall\ A\ B\ s \bullet\ s \in IxDom\ A \wedge A \sqsubseteq B \Rightarrow s \in IxDom\ B$

⊑_IxVal_thm $\vdash \forall\ A\ B\ s \bullet\ s \in IxDom\ A \wedge A \sqsubseteq B \Rightarrow IxVal\ B\ s = IxVal\ A\ s$

IxPack_lemma1

$\vdash \forall\ x\ y\ z \bullet\ IxPack\ (Cons\ (x,\ y)\ z)\ x = Value\ y$

IxPack_lemma2

$\vdash \forall\ w\ x\ y\ z$
$\quad \bullet\ (x = w \Leftrightarrow F) \Rightarrow IxPack\ (Cons\ (x,\ y)\ z)\ w = IxPack\ z\ w$

IxComp_IxPack_lemma

$\vdash \forall\ f\ h\ t$
$\quad \bullet\ IxComp\ (IxPack\ (Cons\ h\ t))\ f$
$\qquad = (\lambda\ z$
$\qquad \bullet\ if\ z = Fst\ h$
$\qquad\ then\ Value\ (f\ (Snd\ h))$
$\qquad\ else\ IxComp\ (IxPack\ t)\ f\ z)$

IxComp_IxPack_thm

$\vdash \forall\ f\ l$
$\quad \bullet\ IxComp\ (IxPack\ l)\ f$
$\qquad = IxPack\ (Map\ (\lambda\ (x,\ y) \bullet\ (x,\ f\ y))\ l)$

18 INDEX