HOL Formalised:
Formal Design of the Logical Kernel

R.D. Arthan
ICL,
Eskdale Road,
Winnersh,
Berks.
RG11 5TT
rda@win.icl.co.uk

25 October 1993
Revised 29 October 2002

Abstract

This document contains a formal specification, in HOL, of the design of the logical kernel of the ProofPower system.

This design document is essentially a sequel to a suite of documents entitle “HOL Formalised” which define the syntax, semantics and deductive system of HOL and provide formal criteria for assessing a tool that purports to be a theorem-proving system for HOL. This document defines a design for such a theorem-proving system which is believed likely to meet these criteria (although that has not been formally proved).

Although fairly abstract, the design does address realistic architectural issues such as how a large body of HOL theories may be physically distributed for use by several users and how the system can support deletion of definitions and axioms without compromising its logical integrity.

An index to the formal material is provided at the end of the document.
1 DOCUMENT CONTROL

1.1 Contents list

1 DOCUMENT CONTROL .............................................................. 1
1.1 Contents list ........................................................................ 1
1.2 Document cross references .................................................... 3
1.3 Changes history .................................................................... 3
1.4 Changes forecast .................................................................. 3

2 GENERAL ................................................................................ 4
2.1 Scope .................................................................................. 4
2.2 Introduction .......................................................................... 4
  2.2.1 Background and Requirements ........................................ 4
  2.2.2 Dependencies .................................................................. 4
  2.2.3 Notation .......................................................................... 4
  2.2.4 Deficiencies ...................................................................... 4
  2.2.5 Possible Enhancements .................................................... 4

3 DISCUSSION ............................................................................ 5
3.1 Basic Concepts for Ensuring Integrity ....................................... 5
3.2 Implementation Strategies ..................................................... 6
3.3 Overview of Model ................................................................ 6
3.4 System Construction ............................................................. 7

4 PRELIMINARIES ....................................................................... 7
5 Preamble .................................................................................. 7
  5.1 Dictionaries ......................................................................... 7
  5.2 Stores ................................................................................ 9

6 THE SYSTEM STATE .................................................................. 10
6.1 User-Defined Data ............................................................... 10
6.2 System Inputs ...................................................................... 10
6.3 Concrete Theories ................................................................ 10
6.4 Concrete Theory Hierarchies ................................................ 11
6.5 Concrete Theorems .............................................................. 12
6.6 The System State ................................................................. 12
  6.6.1 Definition and Initialisation .............................................. 12
  6.6.2 Interpretation Mapping ..................................................... 14
  6.6.3 Well-Formedness ............................................................ 15

7 OPERATIONS ........................................................................... 16
7.1 Discussion ............................................................................ 16
7.2 Utility Functions ................................................................... 16
7.3 Operations on Hierarchies ..................................................... 21
  7.3.1 freeze_hierarchy ............................................................. 21
  7.3.2 new_hierarchy ............................................................... 21
  7.3.3 load_hierarchy ............................................................... 22
7.4 Operations on Theory Attributes .......................................... 22
  7.4.1 open_theory ................................................................. 22
  7.4.2 delete_theory ............................................................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.3</td>
<td>new_theory</td>
</tr>
<tr>
<td>7.4.4</td>
<td>new_parent</td>
</tr>
<tr>
<td>7.4.5</td>
<td>duplicate_theory</td>
</tr>
<tr>
<td>7.4.6</td>
<td>lock_theory</td>
</tr>
<tr>
<td>7.4.7</td>
<td>unlock_theory</td>
</tr>
<tr>
<td>7.5</td>
<td>Operations on Theory Contents</td>
</tr>
<tr>
<td>7.5.1</td>
<td>save_thm</td>
</tr>
<tr>
<td>7.5.2</td>
<td>delete_extension</td>
</tr>
<tr>
<td>7.5.3</td>
<td>delete_thm</td>
</tr>
<tr>
<td>7.5.4</td>
<td>pds_new_axiom</td>
</tr>
<tr>
<td>7.5.5</td>
<td>General Definitional Mechanisms</td>
</tr>
<tr>
<td>7.5.6</td>
<td>pds_new_type</td>
</tr>
<tr>
<td>7.5.7</td>
<td>pds_new_constant</td>
</tr>
<tr>
<td>7.6</td>
<td>Inference</td>
</tr>
</tbody>
</table>

8 SYSTEM CONSTRUCTION

8.1 Auxiliary Definitions
8.2 The System Construction
8.3 Subsystem Critical Properties

9 THEORY LISTING

10 THE THEORY spc005

10.1 Parents
10.2 Constants
10.3 Types
10.4 Type Abbreviations
10.5 Fixity
10.6 Definitions

11 INDEX
1.2 Document cross references


1.3 Changes history

**Issue 1.2 (26 February 1991)** First draft for initial comment on approach.

**Issue 1.6 (16 July 1991)** Revision and corrections in light of comments.

**Issue 1.7 (29 October 2002)** Revision after inspection ID0014.

1.4 Changes forecast

The state well-formedness predicate is yet to be formalised. A choice of which formulation of the critical requirements to use is yet to be made.
2 GENERAL

2.1 Scope

This document gives a formal specification, at an abstract level of parts of the HOL proof development system. The document is called for in [4] and is intended to help meet the requirements concerning integrity and the route to high assurance stated in [3].

2.2 Introduction

2.2.1 Background and Requirements

The high level design document, [4], discusses informally the abstract data type used to represent theorems, which it refers to as the Logical Kernel of the ProofPower proof development system.

This document gives a formal model of the main data structures used in the logical kernel and of the operations on them. We stress that the definitions are intended to describe a simplified model of the actual system, and, for reasons of efficiency or practicality, the actual details of the internal datatypes will be implementation dependent.

2.2.2 Dependencies

This document depends on the suite of documents formalising HOL, overviewed in [5].

2.2.3 Notation

In addition to the specification facilities mentioned in [5], we use the Z-like schema boxes to introduce labelled record types.

2.2.4 Deficiencies

The formalisation of the well-formedness condition on states has yet to be included.

2.2.5 Possible Enhancements

It has been suggested that there may be some merit in giving system extenders more control over collections of theories. For example, it might be useful in some circumstances to arrange that no one theory in a given collection of related theories could be opened without the others also being opened. A neat scheme for doing this within the framework of the present document could be adopted, if such a scheme is discovered. As things stand, such a feature must be implemented in non-critical code.
3 DISCUSSION

3.1 Basic Concepts for Ensuring Integrity

The design of the logical kernel is a development of the LCF paradigm which has been used in earlier
implementations of the HOL logic, most notably the Cambridge implementation described in [9]. The
original reference for LCF is [1]. The ideas depend upon the use of a strongly typed metalanguage
supporting the abstract data type facility, whereby a data type may be declared together with one
or more so-called constructors, in such a way that the type discipline ensures that all values of the
data type are created using the constructors. For the ProofPower implementation of the HOL logic
the metalanguage is Standard ML, see [2], in which the abstype construct provides the necessary
feature.

In crude outline, integrity is ensured in an LCF-style proof development system by representing
theorems as elements of an abstract data type, which we will call THM. The representation type for
THM is the set of sentences of the logic — sequents in the case of HOL as formulated in [5]. The
constructors of the abstract data type are the axioms and inference rules of the logic, so that values
of type THM arise from computations which directly represent formal proofs. The great merit of
this approach is that it concentrates all the code which is critical to the integrity of the system in the
abstract data type and the small amount of code which supports it. Facilities such as user-interfaces
and proof procedures may be implemented using the primitive operations of the abstract data type
and are not themselves critical since they cannot compromise the integrity of the system.

This elegant paradigm has, however, to be adapted to meet practical requirements. The following
paragraphs summarise the key issues and solutions for ProofPower:

1. We must allow the user to make definitions and other extensions, both conservative and axiom-
omatic. Thus proofs (i.e. computations of theorems) are carried out with respect to a context
determined by a collection of such extensions, i.e. a theory in the sense of [5]. Since we wish to
let the user navigate at will around the various theories which have been constructed, we must
mark each theorem with an indicator (actually a store address) which uniquely identifies the
theory to which the theorem belongs. The inference rules use these indicators to ensure that a
theorem is valid in the context in which it is being used.

2. We wish to store the collection of theories belonging to one or more users in a reasonably
efficient fashion. Two measures facilitate this. Firstly, we make the concrete representation
of a theory hold only those extensions which are specific to it. The theories are organised as
a directed acyclic graph given by a parenthood relation defined by the user. The context (or
abstract theory) determined by such a concrete theory comprises the union of the extensions
contained in it and its ancestors with respect to this relation and is represented by a set of
theory addresses. Secondly, we organise the theories themselves into a tree of theory hierarchies.
A theory hierarchy is intended to represent the set of theories constructed by a particular user.
A theory hierarchy may conveniently be implemented as a physical store or database in which
we hold a set of theories together with a pointer to a parent theory hierarchy. This parenthood
relation on theory hierarchies allows a collection of theories to be shared amongst many users
without undue replication in the physical store. A theory hierarchy determines a set of theory
addresses from which the user may construct contexts in which to carry out proof.

3. We wish to allow the user to edit the contents of a theory by deleting extensions, and then,
perhaps, making new extensions which are logically incompatible with the ones which have
been deleted. This implies that a theorem must be marked with an indicator identifying the
set of extensions on which it may depend. For reasons of efficiency, this indicator comprises a
so-called level number. Each extension to a theory or deletion of an extension from a theory
causes the level number to be incremented. When an extension is deleted, the corresponding level number is added to a set of invalid levels maintained in the data structure representing the theory. The inference rules both check that any theorem presented to them has a valid level number and also generate theorems which have a level number corresponding to the most recent extension to the context.

3.2 Implementation Strategies

The design given here can be implemented using several different strategies. The one currently used in ProofPower uses an implementation of Standard ML, namely, Abstract Hardware Ltd.’s Poly/ML, which provides a persistent object store structured as a tree of physical files called databases. The root of this tree as realised for ProofPower contains the ML code of the compiler itself together with the code and data which implements the ProofPower system.

When an interactive or batch session with the Poly/ML compiler is started, the user indicates a database which, if it is to be updated, must be a leaf in the tree. As and when desired the user can save the results of his work in the database. New databases are created using a command script supplied as part of ProofPower which protects the user from most of the intricacies of working with hierarchies as described in section 7.3. The `freeze_hierarchy` operation, for example, is carried out automatically on the parent database when a child is created, and the `load_hierarchy` operation is invoked automatically at the beginning of each session.

An alternative implementation strategy would be to store representations of theory hierarchies in files using metalanguage I/O operations. This has the disadvantage of disassociating the theory hierarchies from any associated metalanguage variable bindings. This disadvantage could doubtless be ameliorated in various ways, largely determined by the capabilities of the metalanguage compiler and associated tools.

3.3 Overview of Model

In the sequel we define a model of a proof development system for HOL. This is a more concrete model than the abstract one used in [8]. Where confusion might otherwise arise we use the terms concrete and abstract to distinguish notions in the present model from related notions in the more abstract treatment. However, the model is still quite abstract in a number of ways. For example, there is no commitment here as to whether the theory hierarchy is held entirely in main store or whether it is a main store data structure used to access the contents of a theory in backing store. Nor do we define a number of mechanisms which will be necessary in the interests of efficiency, e.g. the use of a symbol table to give fast access to the context.

The main features of the implementation which we are modelling are as follows:

1. the representation of the theory hierarchy within the store of a machine;

2. the mechanisms whereby use of a theorem is restricted to contexts which include the context in which it was proved;

3. the commands which manipulate the theory hierarchy or modify the context in which proof is carried out;

4. a reversible facility for the user to prevent a theory from further modification\(^1\).

\(^1\)This locking and unlocking facility is offered as a more general substitute for the ability, in earlier implementations of HOL, to load a theory for read-only access (or the ability to use operating system facilities to prevent a filestore representation of a theory from being modified).
3.4 System Construction

In order to focus attention on the features identified in the previous section we specify the model as a function $pds$ which constructs the system from three subsystems:

1. A **DEFINER**, which stands for the operations which perform theory extensions;
2. An **INFERRER**, standing for the inference rules;
3. An **INTERPRETER**, which corresponds, approximately, to the metalanguage compiler, and is actually a function from DEFINERs and INTERPRETERs to state transition functions.

This construction is purely for conceptual purposes, it is not intended to imply the use of any particular implementation technique only that the implementation be capable of being viewed in this way under an appropriate interpretation function.

Note that the above view on what it means to be an implementation of the design implies that the names used in the implementation may differ from the names used in the design.

It is intended that implementations will include definition schemata, and, perhaps, other built-in theorem schemata, for numeric and other literals. This technique demands either (a) that the proof development system as seen by the user always has suitable definitions of appropriate types and constants in scope or (b) that the implementation of each schema checks that it is operating in a context containing appropriate definitions. Approach (b) is unlikely to be attractive for performance reasons, and approach (a) may lead to boot-strapping problems (since it appears to imply that the proof development system code cannot be used to assist in making the necessary definitions). One approach might be to work on the assumption that the implementation of such schemata actually checked that the right definitions were available and then demonstrate that the checks are actually unnecessary in a particular implementation, in which steps are taken to ensure that the theories containing the definitions are always in scope.

4 PRELIMINARIES

5 Preamble

The theory “spc005” which is defined in this document is introduced as follows. Its parent is the theory “spc004” which defines the critical properties of an abstract model of a HOL proof development system.

```sml
open theory"spc004";
new theory"spc005";
new parent"cache'play" handle Fail _ => ();
```

5.1 Dictionaries

Axioms, definitions and the like are held in the implementation in tables indexed by names. We refer to such tables as dictionaries.
We model dictionaries as sets of pairs representing partial functions in the usual set-theoretic manner. In the implementation these will typically be finite partial functions represented by a concrete data structure such as a list of pairs. However, the implementation will also contain some definitions or theorem schemata (e.g. the rules which define numbers or strings), these may be thought of as (parts of) infinite dictionaries in the appropriate theories.

SML

```sml
declare_type_abbrev("DICT", ["'X"], "STRING -> 'X");
```

Dictionaries are formed starting with an initial, empty, dictionary:

```haskell
HOL Constant
initial_dict : ('X)DICT
```

```
initial_dict = {}
```

Entries may be added to a dictionary using the function `enter`:

```haskell
HOL Constant
enter : STRING -> 'X -> ('X)DICT -> ('X)DICT
```

```
\forall key item dict. enter key item dict = dict \oplus \{(key, item)\}
```

We look things up in a dictionary using `lookup` defined below. Note that the use we will make of `lookup` is such that it may actually be implemented as a partial function, i.e. we will never associate more than one value with a given key.

```haskell
HOL Constant
lookup : STRING -> ('X)DICT -> 'X -> BOOL
```

```
\forall key dict item. lookup key dict item \equiv (key, item) \in dict
```

We may delete things by key from a dictionary using `delete`:

```haskell
HOL Constant
delete : STRING -> ('X)DICT -> ('X)DICT
```

```
\forall key dict. delete key dict = {key} \varsubsetneq dict
```

We may delete entries whose values lie in some set from a dictionary using `block_delete`:

```haskell
HOL Constant
block_delete : ('X SET) -> ('X)DICT -> ('X)DICT
```

```
\forall a dict. block_delete a dict = dict \varsubsetneq a
```

`keys` gives the set of key values in use in a dictionary:

```haskell
HOL Constant
keys : ('X)DICT -> STRING SET
```

```
keys = Dom
```
5.2 Stores

The state of the proof development system will be held in assignable metalanguage variables of various types. To model these we use a polymorphic notion of a store.

The addresses for our stores come from the following countably infinite type, ADDR. It gives a useful cross-check on the present specification for this type to have a parameter which identifies the type of object addressed. To achieve this we represent a (′X)ADDR as a pair ((ex′X•T), n) where n is a natural number. The result is a polymorphic type all of whose instances are isomorphic to the natural numbers.

SML
val ADDR_DEF = new_type_defn(["ADDR_DEF"], "ADDR", ["'X"]
        (tac_proof([[], "\exists a':X \times N(\lambda x.Fst x = ex':X•T) a"],
          \exists_tac((ex':X•T), (n:N))\ THEN
          rewrite_tac([])));

A store is a partial function from addresses to values, represented as a set of pairs:

SML
declare_type_abbrev("STORE", ["'X"], \;(′X)ADDR ↔ ′X);

The operations on stores are assignment, dereferencing and allocation.

< − is the assignment operation, note that it is not defined to create new storage locations, but only to modify existing ones.

SML
declare_infix(300, "<−");

HOL Constant

\$<− : (′X)ADDR → ′X → (′X)STORE → (′X)STORE

\(\forall addr\ value\ st\bullet\)

\(addr \in Dom st \Rightarrow\)

\(addr <− value\ st = st \oplus \{(addr, value)\}\)

fetch is the dereferencing operation:

HOL Constant

fetch : (′X)ADDR → (′X)STORE → ′X → BOOL

\(\forall addr\ st\ value\bullet fetch\ addr\ st\ value \Leftrightarrow (addr, value) \in st\)

new is the allocation operation:

HOL Constant

new : ′X → (′X)STORE → ((′X)STORE × (′X)ADDR) → BOOL

\(\forall value\ st1\ st2\ addr\bullet\)

\(new\ value\ st1\ (st2, addr) \Leftrightarrow\)

\(\neg addr \in Dom st1\)

\(\land\ st2 = st1 \oplus \{(addr, value)\}\)
Stores are constructed using \textit{new} from an initial empty store:

\begin{verbatim}
HOL Constant

\begin{verbatim}
initial_store : ('X)STORE
\end{verbatim}

\begin{verbatim}
initial_store = {}
\end{verbatim}

\end{verbatim}

6 \textbf{THE SYSTEM STATE}

6.1 User-Defined Data

Theories will be record types containing a field in which essentially arbitrary user-defined data can be stored. This will be used to support the concrete syntax of HOL, e.g. by allowing the syntactic properties of identifiers to be stored in a theory, and may be used for similar purposes for other languages. The presence of this field is not critical to the integrity of the system. Our model will be polymorphic over the type of this information, for which we will systematically use the type variable \textit{'UD}.

6.2 System Inputs

We will systematically use the type variable \textit{'IP} for the components of the input to the system which we do not wish to specify in detail here. The actual inputs to the abstract data type would be represented in the model by instantiating \textit{'IP} to some disjoint union type allowing for the various possibilities (e.g. the template term which is a parameter to the rule of substitution defined in [7]).

6.3 Concrete Theories

It is useful to have a representation for the contents of a theory. This serves for the internal representation in our simplified model (and an analogous type might be available in an implementation for general use, e.g by the theory lister).

\begin{verbatim}
HOL Labelled Product

\_THEORY\_CONTENTS
\begin{verbatim}
tc_name : STRING;
tc_ty_env : (N × N) DICT;
tc_con_env : (TYPE × N) DICT;
tc_parents : STRING LIST;
tc_axiom_dict : (SEQ × N) DICT;
tc_definition_dict : (SEQ × N) DICT;
tc_theorem_dict : (SEQ × N) DICT;
tc_current_level : N;
tc_deleted_levels : N SET;
tc_user_data : 'UD
\end{verbatim}
\end{verbatim}

Here the fields have the following significance:
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>This gives the name of the theory.</td>
</tr>
<tr>
<td>ty_env</td>
<td>This represents a type environment assigning arities and level numbers to type operator names. It corresponds to the $TY_ENV$ component of a theory as specified in [5]. The level number gives the level number which was current when the type was introduced.</td>
</tr>
<tr>
<td>con_env</td>
<td>This represents a constant environment assigning types and level numbers to constant names. It corresponds to the $CON_ENV$ component of a theory as specified in [5]. The level number gives the level number which was current when the constant was introduced.</td>
</tr>
<tr>
<td>parents</td>
<td>This is the set of names of parents of this theory.</td>
</tr>
<tr>
<td>axiom_dict</td>
<td>This contains the non-definition axioms of the theory. Each axiom is marked with the level number which was current when the axiom was introduced.</td>
</tr>
<tr>
<td>definition_dict</td>
<td>This contains the definitional axioms of the theory. Like the axioms, these are marked with the level number current when the definitional axiom was introduced.</td>
</tr>
<tr>
<td>theorem_dict</td>
<td>This contains the theorems which have been saved on the theory. These are marked with the level number which was current when the theorem was proved (or 0 if the theorem belongs to an ancestor of the current theory).</td>
</tr>
<tr>
<td>user_data</td>
<td>This contains the user-defined data stored in the theory.</td>
</tr>
<tr>
<td>current_level</td>
<td>This is the current level number. It is 0 when a theory is first created. It is incremented whenever an extension to the theory is introduced or deleted.</td>
</tr>
<tr>
<td>deleted_levels</td>
<td>This is the set of level numbers corresponding to extensions which have been deleted.</td>
</tr>
</tbody>
</table>

Note that a theory can be used without modifying any of the above information. Moreover this information does not depend on the hierarchy containing the theory.

### 6.4 Concrete Theory Hierarchies

A theory hierarchy is essentially a finite set of records each comprising a theory contents together with information about the theory which is local to the hierarchy.

The local information comprises a status attribute (which indicates a fairly permanent property of the theory) and a scope attribute which is set true when the theory in question is the current theory or one of its ancestors. The scope attribute is discussed in more detail in section 7.4.1 below.

We recognise the following four values for the status attribute.

**SML**

```
| declare_type_abbrev("STATUS", [], "\{ONE + ONE + ONE + ONE\}");
```

**HOL Constant**

```
| TSNormal    : STATUS;
| TSLocked    : STATUS;
| TSAncestor  : STATUS;
| TSDelated    : STATUS;
```
The signiﬁcance of the theory status values is as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSNormal</td>
<td>A theory which can be modiﬁed while this theory hierarchy is current;</td>
</tr>
<tr>
<td>TSLocked</td>
<td>A theory which cannot be modiﬁed while this theory hierarchy is current because the user has asked for it to be locked (see section 7.4.6 for more information);</td>
</tr>
<tr>
<td>TSAncestor</td>
<td>A theory which cannot be modiﬁed while this theory hierarchy is current since it belongs to an ancestor of some hierarchy (see section 6.6.3 below for more information);</td>
</tr>
<tr>
<td>TSDeleted</td>
<td>A theory which has been deleted.</td>
</tr>
</tbody>
</table>

The information about a theory held in a theory hierarchy then has the following type:

\[
\text{THEORY\_INFO}
\]

- \(ti\_status\) : STATUS;
- \(ti\_inscope\) : BOOL;
- \(ti\_contents\) : ((‘UD)THEORY\_CONTENTS)ADDR

Here the address will reference a store of theory contents held in the state.

A theory hierarchy will comprise a list of THEORY\_INFOS:

\[
\text{SML}
\]

\[
declare\_type\_abbrev("HIERARCHY", ["‘UD"], "\langle\langle‘UD)THEORY\_INFO\rangle\LIST\rangle");
\]

### 6.5 Concrete Theorems

A theorem is represented by the following data type:

\[
PDS\_THM
\]

- \(pt\_theory\) : ((‘UD)THEORY\_CONTENTS)ADDR;
- \(pt\_level\) : N;
- \(pt\_sequent\) : SEQ

The \(pt\_theory\) component here gives the address of the (contents of the) theory to which the theorem belongs (with respect to a store of theory contents held in the state of the system). The level number is that which was current when the theorem was proved.

### 6.6 The System State

#### 6.6.1 Definition and Initialisation

The state of our model of the proof development system has the following type:
Here the current theory (resp. hierarchy) is the theory (resp. hierarchy) in which modifications to the state of the system are currently being made, these modifications often constituting updates to the three stores.

The theorem store component is different in intention from the other two stores in that it will not, in practice, correspond to a metalanguage variable inside the abstract data type. It represents the locations in which theorems computed by the abstract data type have been stored.

The initial state of the system is parameterised by the initial user-defined data. To define it we first define the initial theory (we apologise for the fact that the initial theory is \textsc{MIN} not \textsc{INIT}) The initial theory information comes supplied with a store containing the contents of a suitable initial theory:

\begin{verbatim}
HOL Constant
initial_theory : 'UD \rightarrow 
  (  (('UD)THEORY_CONTENTS)STORE) \times (('UD)THEORY_INFO
\end{verbatim}

\begin{verbatim}
\forall ud\bullet initial_theory ud = 
  let contents = MkTHEORY_CONTENTS
    "MIN"
    initial_dict initial_dict
    []
    initial_dict initial_dict initial_dict
    0 []
    ud
  in let (st, addr) = \epsilon(st, addr)\bullet new contents initial_store (st, addr)
  in (st, MkTHEORY_INFO TSNormal T addr)
\end{verbatim}

The initial state is then as follows:

\begin{verbatim}
HOL Constant
initial_state : 'UD \rightarrow ('UD)PDS_STATE
\end{verbatim}

\begin{verbatim}
\forall ud\bullet initial_state ud = 
  let (thy_st, thy_info) = initial_theory ud
  in let (hier_st, hier_addr) = \epsilon(st, addr)\bullet new [thy_info] initial_store (st, addr)
  in MkPDS_STATE (ti_contents thy_info) hier_addr thy_st hier_st initial_store
\end{verbatim}
6.6.2 Interpretation Mapping

In this section we define an interpretation function from PDS\_STATEs to the more abstract notion of a theory hierarchy defined in [8]. To do this requires a number of auxiliary definitions:

\textit{theory\_contents} returns the theory contents associated with a theory name in a state. It is a partial function which we represent as a relation.

\[
\text{theory\_contents} : (\text{PDS\_STATE} \rightarrow \text{STRING}) \rightarrow (\text{PDS\_STATE} \rightarrow \text{THEORY\_CONTENTS}) \rightarrow \text{BOOL}
\]

\[
\forall \text{name thy\_c} \cdot \text{theory\_contents state name thy\_c} \leftrightarrow
\begin{align*}
\text{let thy\_st = ps\_theory\_store state} \\
\text{in let hier\_st = ps\_hierarchy\_store state} \\
\text{in let cur\_hier = ps\_current\_hierarchy state} \\
\text{in let infos = ex\_fetch cur\_hier hier\_st x} \\
\text{in let thys = Map((\lambda \text{addr \_x} \cdot \text{fetch addr thy\_st x}) \circ \text{ti\_contents}) infos} \\
\text{in \exists \text{thy} \cdot \text{thy Elems thys \land tc\_name thy = name}}
\end{align*}
\]

The following function returns the names of the theories in a state:

\[
\text{theory\_names} : (\text{PDS\_STATE} \rightarrow \text{STRING}) \rightarrow \text{STRING SET}
\]

\[
\forall \text{name} \cdot \text{name \in \text{theory\_names state}} \leftrightarrow \exists \text{thy\_c} \cdot \text{theory\_contents state name thy\_c}
\]

\textit{theory\_ancestors} returns the names of the ancestors of a given theory (which we take to include the theory itself, if it is in the state):

\[
\text{theory\_ancestors} : (\text{PDS\_STATE} \rightarrow \text{STRING}) \rightarrow \text{STRING SET}
\]

\[
\forall \text{name} \cdot \text{theory\_ancestors state name = \bigcap\{P:STRING SET | (name \in \text{theory\_names state} \Rightarrow name \in P) \land (\forall \text{anc1 thy\_c anc2} \cdot \text{anc1} \in P \land \text{theory\_contents state anc1 thy\_c} \land \text{anc2} \in \text{Elems (tc\_parents thy\_c)} \Rightarrow \text{anc2} \in P\}\}
\]

Given a set of theory contents, \textit{interpret\_theory\_contents} constructs a \textit{THEORY} in the sense of [5], together with the sets of definitional axioms and saved theorems which are used in the definition of the abstract notion of theory hierarchy in [8].
Our interpretation mapping for a state is now easy to define (note that the definition results in abstract theory hierarchies which map undefined theory names to the theory all of whose components are empty).

(Note that the interpretation of a state does not depend on the theorem store. This is because the theorem store will in general contain theorems which were proved in theories which have been deleted or which depend on definitions or axioms which have been deleted.)

6.6.3 Well-Formedness

As is apparent from the construction of the interpretation mapping, we require the state to satisfy an invariant which ensures that:

1. no hierarchy in the hierarchy store contains two distinct THEORY_INFOs whose contents fields address theory contents with the same name. Thus a theory name uniquely identifies the address of the corresponding theory within a hierarchy;

2. there are no dangling addresses; more accurately the current theory (resp. hierarchy) should be a valid address for the theory (resp. hierarchy) store and the list of addresses addressed by the current hierarchy should all be valid addresses for the theory store;
3. the ancestral of the parenthood relation is a rooted DAG (with root the initial theory);

4. the set of type names defined in a theory is disjoint from the type names in its ancestors (and similarly for constant names);

5. no entry in any dictionary in any theory contains a level number which is in the set of deleted levels for that theory.

Note that condition 4 above implies that that the type (or constant) names in a theory must be disjoint from those in its descendants. This implies that we must not introduce new type or constant names into a hierarchy which is the ancestor of some hierarchy. This is the significance of the TSAncestor status value.

The formalisation of these conditions has been deferred.

7 Operations

7.1 Discussion

We can now define the operations on states which are of concern to us. We consider the operations under four headings:

Operations on Hierarchies These are the operations concerned with creating and loading theory hierarchies;

Operations on Theory Attributes These are the operations which affect the status and scope attributes for one or more theories;

Operations on Theory Contents These are the operations which affect the contents of a theory;

Inference Rules These are the inference rules (viewed as functions on states returning theorems).

The operations are described in the following sections under the above headings. Except for the Inference Rules, the operations are (functions returning) functions from states to states, and, we are essentially doing imperative programming in HOL. It will be an implicit precondition of all of these operations that the stores in the state are not full. Since each operation only allocates a finite number of new addresses in the stores, this precondition will always be met by states constructed by finite iteration of these operations starting from the initial state. We specify the operations so that they always succeed if there is room enough in the stores (by making them do the identity state change, if what might otherwise be a precondition does not hold).

The operation new_parent affects both the contents of the current theory and the scope attributes of the new ancestors. We will classify it, arbitrarily, as an operation on theory attributes.

7.2 Utility Functions

It is convenient to have a single function giving the components of a state (the only inconvenience is having to write out its signature!):
HOL Constant

\textbf{dest\_state} : ('UD)\texttt{PDS\_STATE} \to \\
\hspace{1em} ( ('UD)\texttt{THEORY\_CONTENTS ADDR} \\
\hspace{2em} \times \ ('UD)\texttt{HIERARCHY ADDR} \\
\hspace{2em} \times \ ('UD) \texttt{THEORY\_CONTENTS STORE} \\
\hspace{2em} \times \ ('UD)\texttt{HIERARCHY STORE} \\
\hspace{2em} \times \ ('UD)\texttt{PDS\_THM STORE} )

\forall \text{state} \bullet \text{dest\_state state} = ( \\
\hspace{2em} \text{ps\_current\_theory state}, \\
\hspace{2em} \text{ps\_current\_hierarchy state}, \\
\hspace{2em} \text{ps\_theory\_store state}, \\
\hspace{2em} \text{ps\_hierarchy\_store state}, \\
\hspace{2em} \text{ps\_theorem\_store state})

Similarly, the following destructor function for theory contents is useful

HOL Constant

\textbf{dest\_theory\_contents} : ('UD)\texttt{THEORY\_CONTENTS} \to \\
\hspace{1em} ( STRING \\
\hspace{2em} \times \ ('N \times 'N) \texttt{DICT} \\
\hspace{2em} \times \ (\texttt{TYPE} \times 'N) \texttt{DICT} \\
\hspace{2em} \times \ \texttt{STRING LIST} \\
\hspace{2em} \times \ (\texttt{SEQ} \times 'N) \texttt{DICT} \\
\hspace{2em} \times \ (\texttt{SEQ} \times 'N) \texttt{DICT} \\
\hspace{2em} \times \ ('N \times 'N) \texttt{DICT} \\
\hspace{2em} \times \ 'N \\
\hspace{2em} \times \ 'N \texttt{SET} \\
\hspace{2em} \times \ 'UD )

\forall \text{tc} \bullet \text{dest\_theory\_contents tc} = ( \\
\hspace{2em} \text{tc\_name tc}, \\
\hspace{2em} \text{tc\_ty\_env tc}, \\
\hspace{2em} \text{tc\_con\_env tc}, \\
\hspace{2em} \text{tc\_parents tc}, \\
\hspace{2em} \text{tc\_axiom\_dict tc}, \\
\hspace{2em} \text{tc\_definition\_dict tc}, \\
\hspace{2em} \text{tc\_theorem\_dict tc}, \\
\hspace{2em} \text{tc\_current\_level tc}, \\
\hspace{2em} \text{tc\_deleted\_levels tc}, \\
\hspace{2em} \text{tc\_user\_data tc} )

current\_theory\_contents returns the contents of the current theory, (here and elsewhere we use variable names of the form \_1, \_2 etc. for variables which are required by the syntax but whose value we are not concerned with).
HOL Constant

\[ \text{current\_theory\_contents} : \mathcal{PDS\_STATE} \to \mathcal{PDS\_THEORY\_CONTENTS} \]

\[
\forall \text{state} \cdot \text{current\_theory\_contents} \text{ state } = \\
\text{let } (\text{cur\_thy}, \_1, \text{thy\_st}, \_2, \_3) = \text{dest\_state} \text{ state} \\
\text{in } \ (\text{etc\_fetch cur\_thy thy\_st tc})
\]

current\_theory\_name returns the name of the current theory.

HOL Constant

\[ \text{current\_theory\_name} : \mathcal{PDS\_STATE} \to \text{STRING} \]

\[
\forall \text{state} \cdot \text{current\_theory\_name} \text{ state } = \\
\text{tc\_name(current\_theory\_contents state)}
\]

current\_abstract\_theory returns the abstract theory corresponding to the current theory. This function is used later to abbreviate the specification of various conditions.

HOL Constant

\[ \text{current\_abstract\_theory} : \mathcal{PDS\_STATE} \to \text{THEORY} \]

\[
\forall \text{state} \cdot \text{current\_abstract\_theory} \text{ state } = \text{Fst(interpret\_theory\_contents \{tc\exists \text{anc} \\
\text{anc} \in \text{theory\_ancestors} \text{ state} \ (\text{current\_theory\_name state)} \\
\land \ \text{theory\_contents state anc tc}\})}
\]

theory\_info returns the THEORY\_INFO associated with a given theory name in the current state (and returns rubbish if the name does not identify a theory in the state).

HOL Constant

\[ \text{theory\_info} : \mathcal{PDS\_STATE} \to \text{STRING} \to \mathcal{PDS\_THEORY\_INFO} \]

\[
\forall \text{name} \cdot \text{theory\_info} \text{ state name } = \\
\text{let } (\text{cur\_thy}, \text{cur\_hier}, \text{thy\_st}, \text{hier\_st}, \_1) = \text{dest\_state} \text{ state} \\
\text{in let hier = eh\_fetch cur\_hier hier\_st h} \\
\text{in eti\_let tc\_name(etc\_fetch (ti\_contents ti) thy\_st tc) = name} \\
\land \ \neg ti\_status ti = \text{TSDeleted}
\]

current\_theory\_status returns the status value associated with the current theory. Note that this status cannot be TSDeleted in the states arising from the operations we will define.

HOL Constant

\[ \text{current\_theory\_status} : \mathcal{PDS\_STATE} \to \text{STATUS} \]

\[
\forall \text{state} \cdot \text{current\_theory\_status} \text{ state } = \\
\text{ti\_status (theory\_info state (current\_theory\_name state))}
\]

Several of the operations we wish to define involve the important notion of checking whether a theorem is in scope. The check is carried out as follows.
1. we fetch the theory contents addressed by the theory component of the theorem;

2. we fetch the `THEORY_INFO` associated with the name in the theory contents computed in step 1;

3. We return true iff. the following three conditions hold: (a) the address in the `THEORY_INFO` is the same as that in the theorem; (b) the scope flag in the `THEORY_INFO` is true; (c) the level number in the theorem is not one of the deleted levels in the theory contents.

HOL Constant

\[
\text{check_thm} : (\text{UD}) PDS\_STATE \rightarrow (\text{UD}) PDS\_THM \rightarrow \text{BOOL}
\]

\[
\forall \text{state thm} \bullet \text{check_thm state thm} \Leftrightarrow \\
\text{let } (\text{cur_thy}, \text{cur_hier}, \text{thy_st}, \text{hier_st}, \_1) = \text{dest_state state} \\
in \text{let } tc = \text{etc}\text{fetch (pt\_theory thm)} \text{thy_st tc} \\
in \text{let } ti = \text{theory\_info state (tc\_name tc)} \\
in (\text{pt\_theory thm} = ti\_contents ti \\
\land ti\_inscope ti \\
\land \neg \text{pt\_level thm } \in tc\_deleted\_levels tc)
\]

HOL Constant

\[
\text{check_thm\_address} : (\text{UD}) PDS\_STATE \rightarrow (\text{UD}) PDS\_THM ADDR \rightarrow \text{BOOL}
\]

\[
\forall \text{state thm\_ad} \bullet \text{check_thm\_address state thm\_ad} \Leftrightarrow \\
\text{let } (\_1, \_2, \_3, \_4, \text{thm\_st}) = \text{dest_state state} \\
in \exists \text{thm}\bullet \text{fetch thm\_ad thm\_st thm } \land \text{check_thm state thm}
\]

HOL Constant

\[
\text{fetch_thms} : \\
(\text{UD}) PDS\_STATE \rightarrow (\text{UD}) PDS\_THM ADDR LIST \rightarrow (\text{UD}) PDS\_THM LIST
\]

\[
\forall \text{state thm\_ads} \bullet \text{fetch_thms state thm\_ads} = \\
\text{let } (\_1, \_2, \_3, \_4, \text{thm\_st}) = \text{dest_state state} \\
in \text{Map } (\lambda a \bullet \text{fetch a thm\_st thm\_ads}) \text{ thm\_ads}
\]

We may sometimes need to know whether one theory hierarchy is an ancestor of another. This is essentially inclusion of lists of theory addresses viewed as sets. If the hierarchies in question are given by their addresses relative to a state, the following function gives the relation. Note that it returns false if either of the addresses is not valid for the hierarchy store in the state.
HOL Constant

\textbf{hierarchy\_ancestor} : \((\textsc{ UD})\text{PDS\_STATE} \rightarrow \text{\textsc{ UD}}\text{HIERARCHY})\text{ADDR} \rightarrow \text{\textsc{ UD}}\text{HIERARCHY})\text{ADDR} \rightarrow \text{BOOL}

\forall \text{state hier\_ad1 hier\_ad2} \bullet \text{hierarchy\_ancestor state hier\_ad1 hier\_ad2} \Leftrightarrow
\text{let } (\_1, \text{cur\_hier, \_2, hier\_st, \_3) = dest\_state state}
\text{in } \forall h1 h2\bullet
\text{fetch hier\_ad1 hier\_st h1 } \land \text{fetch hier\_ad2 hier\_st h2} \Rightarrow
\text{Elems (Map ti\_contents h1) } \subseteq \text{Elems (Map ti\_contents h2)}

\textit{pds\_mk\_thm} makes a theorem from a given sequent with \textit{theory} and \textit{level} values taken from the current theory of a state. It makes no checks whatsoever. It is the responsibility of a function using \textit{mk\_thm} to store the resulting theorem in the theorem store.

HOL Constant

\textbf{pds\_mk\_thm} : \((\textsc{ UD})\text{PDS\_STATE} \rightarrow \text{SEQ} \rightarrow (\textsc{ UD})\text{PDS\_THM}

\forall \text{state seq}\bullet \text{pds\_mk\_thm state seq} =
\text{let } \text{cur\_thy = ps\_current\_theory state}
\text{in } \text{let lev = tc\_current\_level (current\_theory\_contents state)}
\text{in } \text{MkPDS\_THM cur\_thy lev seq}

\textit{make\_current} does most of the work of opening a theory. It is defined here because it is needed both in \textit{open\_theory} and in \textit{load\_hierarchy}, q.v. It is given a name which must identify a theory which has not been deleted. On this assumption, it carries out the following steps.

1. compute a modified theory hierarchy in which the \textit{inscope} flags are true for the new current theory and its ancestors only;
2. assign the result of step 1 to the current hierarchy;
3. set the current theory to the address of the theory contents identified by the name (as found in the corresponding \textit{THEORY\_INFO}).

HOL Constant

\textbf{make\_current} : \text{STRING} \rightarrow (\textsc{ UD})\text{PDS\_STATE} \rightarrow (\textsc{ UD})\text{PDS\_STATE}

\forall \text{thyn state}\bullet \text{make\_current thyn state} =
\text{let } (\text{cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st}) = dest\_state state
\text{in } \text{let } f1 = \lambda ti\bullet \text{tc\_name(elec\_fetch (ti\_contents ti) thy\_st tc)}
\text{in } \text{let } f2 = \lambda ti\bullet (f1 ti) \in \text{theory\_ancestors state thyn}
\text{in } \text{let } f3 = \lambda ti\bullet \text{MkTHEORY\_INFO(ti\_status ti)(f2 ti)(ti\_contents ti)}
\text{in } \text{let hier' = Map f3 (elec\_fetch cur\_hier hier\_st h)}
\text{in } \text{let hier\_st' = (cur\_hier <- hier') hier\_st}
\text{in } \text{let cur\_thy' = ti\_contents (theory\_info state thyn)}
\text{in } \text{MkPDS\_STATE cur\_thy' cur\_hier thy\_st hier\_st' thm\_st}
7.3 Operations on Hierarchies

7.3.1 freeze_hierarchy

freeze_hierarchy changes the status of all undeleted theories in the current hierarchy to TSAnccestor (in readiness for subsequent new_hierarchy operations). It performs the following steps:

1. compute a modified theory hierarchy from the one held in the current hierarchy by setting the status of all undeleted theories to be TSAnccestor;
2. assign the result of step 1 to the current hierarchy;

HOL Constant freeze_hierarchy : (UD)PDS_STATE → (UD)PDS_STATE

∀state•freeze_hierarchy state =
let (cur_thy, cur_hier, thy_st, hier_st, thm_st) = dest_state state
in let f1 = λn•if n = TSDeleted then n else TSAnccestor
in let f2 = λti•MkTHEORY_INFO(f1(ti.status ti))(ti.inscope ti)(ti.contents ti)
in let hier’ = Map f2 (eh•fetch cur_hier hier_st h)
in let hier_st’ = (cur_hier <- hier’) hier_st
in MkPDS_STATE cur_thy cur_hier thy_st hier_st’ thm_st

7.3.2 new_hierarchy

new_hierarchy creates a new hierarchy. It performs the following steps:

1. if there is an theory in the current hierarchy with status other than TSAnccestor or TSDeleted then leave the state alone.
2. allocate a new theory hierarchy initially equal to the current hierarchy.
3. return a state with the current hierarchy equal to the one allocated in step 2.

HOL Constant new_hierarchy : (UD)PDS_STATE → (UD)PDS_STATE

∀state•new_hierarchy state =
let (cur_thy, cur_hier, thy_st, hier_st, thm_st) = dest_state state
in let hier = eh•fetch cur_hier hier_st h
in
if (∃ti•ti ∈ Elems hier
  ∧ ¬ti.status ti ∈ {TSAnccestor; TSDeleted})
then state
else let (hier_st’, cur_hier’) = ε(st, a)•new hier hier_st (st, a)
in MkPDS_STATE cur_thy cur_hier’ thy_st hier_st’ thm_st
7.3.3 *load_hierarchy*

This operation typically corresponds to loading a theory into the system from filestore. Not all implementations will require it, since in a persistent object store approach it may be possible to arrange for the state of the system to persist from session to session.

The parameter to *load_hierarchy* is the address of the hierarchy to load. This might in practice be a metalanguage variable or a file name.

The algorithm is as follows:

1. if the address of the hierarchy to be loaded is not valid for the hierarchy store then leave the state alone;
2. otherwise, if the hierarchy we wish to load is not a descendant of the current hierarchy then leave the state alone;
3. otherwise, compute the state in which the current hierarchy is the address of the new hierarchy and all other fields are as in the old state.
4. return the result of making the original current theory current again in the state computed in step 3.

Note that the current theory is unchanged by this operation. The resulting state is nonetheless well-formed, since the new current hierarchy is a descendant of the old one.

HOL Constant

\[
\text{load\_hierarchy} : (\langle UD \rangle \text{HIERARCHY}) \text{ADDR} \to (\langle UD \rangle \text{PDS\_STATE}) \to (\langle UD \rangle \text{PDS\_STATE})
\]

\[
\forall \text{hier state} \cdot \text{load\_hierarchy hier state} = \\
\text{let } (\text{cur\_thy}, \text{cur\_hier}, \text{thy\_st}, \text{hier\_st}, \text{thm\_st}) = \text{dest\_state state} \\
in \\
\text{if } \neg (\text{hierarchy\_ancestor state cur\_hier hier}) \\
\text{then state} \\
\text{else let } \text{cur\_thyn} = \text{current\_theory\_name state} \\
in \text{let } \text{st'} = \text{MkPDS\_STATE cur\_thy hier\_st hier\_st thm\_st} \\
in \text{make\_current cur\_thyn st'}
\]

7.4 Operations on Theory Attributes

7.4.1 *open_theory*

*open_theory* takes one argument which is the name of the theory to be opened (i.e. made the current theory).

1. if the name is not the name of any theory or it is the name of a theory which has been deleted, then we leave the state alone;

---

\[^2\text{It will be required with a persistent object store mechanism such as the PolyML one, since the state variables inside the abstract datatype will be held in the HOL system database not the user's database and so their values will not be permanently updated by the theory management operations.}\]
2. otherwise, return the state obtained by using \texttt{make\_current} to make the named theory the current theory.

\begin{verbatim}
HOL Constant
open\_theory : STRING \to ('UD)PDS\_STATE \to ('UD)PDS\_STATE

\forall thyn state\bullet open\_theory thyn state =
\begin{align*}
& \text{if } \lnot thyn \in \text{theory\_names state} \lor \text{ti\_status(theory\_info state thyn) = TSDeleted} \\
& \text{then } \text{state} \\
& \text{else } \text{make\_current thyn state}
\end{align*}
\end{verbatim}

\textbf{7.4.2 delete\_theory}

\textit{delete\_theory} takes one argument which is the name of the theory to be deleted. The algorithm is as follows:

1. if the name is not the name of any theory, or if the theory it names does not have status \texttt{TSNormal} or has children or if it is in scope we leave the state alone;
2. otherwise, we compute a modified theory hierarchy in which the theory to be deleted has its status attribute set to \texttt{TSDeleted};
3. We assign to the theory contents for this theory an empty theory of the same name;
4. we assign the result of step 2 to the current hierarchy

Before we define \textit{delete\_theory}, we specify a function to compute the empty theory required in step 3. This is also used to support \textit{new\_theory}. The function is parameterised by the theory name and the desired parents.

\begin{verbatim}
HOL Constant
empty\_theory :
STRING \to (STRING LIST) \to 'UD \to ('UD) THEORY\_CONTENTS

\forall thyn pars ud\bullet empty\_theory thyn pars ud =
\begin{align*}
& \text{MkTHEORY\_CONTENTS} \\
& \text{thyn} \\
& \text{initial\_dict} \quad \text{initial\_dict} \\
& \text{pars} \\
& \text{initial\_dict} \quad \text{initial\_dict} \quad \text{initial\_dict} \\
& \text{0} \quad \{\} \\
& \text{ud}
\end{align*}
\end{verbatim}

For \textit{delete\_theory} we also need an arbitrary user datum value:

\begin{verbatim}
HOL Constant
arbitrary\_ud : 'UD

T
\end{verbatim}
\textbf{delete\_theory} : STRING → (UD)PDS\_STATE → (UD)PDS\_STATE

\begin{verbatim}
∀thy\nstate·delete\_theory thyn state =
  if ¬thyn ∈ theory\_names state
  ∨ ¬ti\_status\(\text{theory\_info state thyn}) =\text{TSNormal}
  ∨ ti\_inscope\(\text{theory\_info state thyn})
  ∨ ∃\text{childname tc · theory\_contents state childname tc}
    ∧ thyn ∈ Elems (tc\_parents tc)
  then state
  else let (cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st) = dest\_state state
      in let ti = theory\_info state thyn
          in let f = λti'·
              if ti' = ti
                then MkTHEORY\_INFO TSDeleted F (ti\_contents ti)
                else ti
          in let hier' = Map f (\text{eh • fetch cur\_hier hier\_st h})
          in let hier\_st' = (cur\_hier <- hier') hier\_st
          in let thy\_st' = (ti\_contents ti <- thy) thy\_st
          in MkPDS\_STATE cur\_thy cur\_hier thy\_st' hier\_st' thm\_st
\end{verbatim}

7.4.3 \textit{new\_theory}

\textit{new\_theory} takes two arguments, the first of which is the name of the theory to be created. The new theory has the current theory as parent. The current theory is not changed\(^3\). The second argument to \textit{new\_theory} gives an initial user-defined data value for the new theory.

1. if the name is the name of an existing, undeleted, theory, then we leave the state alone;
2. otherwise, we allocate space in the theory store for the new theory initialised to an empty theory with the given name and user-defined data, and with the current theory as its parent;
3. we compute a new theory hierarchy by pushing a \textit{THEORY\_INFO} for the new theory onto the current theory hierarchy;
4. we assign the result of step 3 to the current hierarchy

\(^3\)The user interface to this function may open the new theory after performing the primitive operation described here.
new\_theory : STRING → 'UD → ('UD)PDS\_STATE → ('UD)PDS\_STATE

∀thyn ud state ● new\_theory thyn ud state =
if thyn ∈ theory\_names state
then state
else let (cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st) = dest\_state state
in let thy = empty\_theory thyn [current\_theory\_name state] ud
in let (thy\_st', addr) = ε(st, a) ● new thy thy\_st (st, a)
in let ti = MkTHEORY\_INFO TSNormal F addr
in let hier' = Cons ti (eh\_fetch cur\_hier hier\_st h)
in let hier\_st' = (cur\_hier <- hier') hier\_st
in MkPDS\_STATE cur\_thy cur\_hier thy\_st' hier\_st' thm\_st

7.4.4 new\_parent

new\_parent takes one argument, which is the name of the theory to be added as new parent of the current theory. The algorithm is as follows (in which we should recall that the ancestors of a theory are taken to include the theory itself):

1. we check to see whether any of the following conditions is satisfied:
   
   (a) the name is not the name of an existing theory;
   
   (b) the name is already the name of a parent of the current theory;
   
   (c) an ancestor, anc, of the theory identified by the name contains a type name or a constant name which is in the current abstract theory, but anc is not already an ancestor of the current theory.

   if any of the above conditions hold, then we leave the state alone;

2. otherwise, we compute a new theory contents from the current theory contents by adding the name to the set of its parents;

3. we compute a new theory hierarchy in which the inscope flags are true for those theories which are either ancestors of the original current theory or ancestors of the new parent;

4. we assign to the current theory the results of step 2 and to the current hierarchy the results of step 3.
HOL Constant

\[ \text{new\_parent} : \text{STRING} \rightarrow (\text{UD})\text{PDS\_STATE} \rightarrow (\text{UD})\text{PDS\_STATE} \]

\[ \forall \text{thyn \ state \ new\_parent \ thyn \ state} = \]
   \[ \text{if } \neg \text{thyn} \in \text{theory\_names \ state} \]
   \[ \lor \text{ thyn} \in \text{Elems(tc\_parents (current\_theory\_contents \ state))} \]
   \[ \lor \exists \text{ancn} \in \text{theory\_ancestors \ state \ thyn} \]
   \[ \bigcap \text{theory\_ancestors \ state \ (current\_theory\_name \ state))} \]
   \[ \land \text{let anc} = \epsilon\text{anc\_theory\_contents \ state \ ancn \ anc} \]
   \[ \text{in let cur\_thy} = \text{current\_abstract\_theory \ state} \]
   \[ \land \exists \text{ty} \in \text{Dom(types \ curr\_thy)} \]
   \[ \land \text{lookup \ ty (tc\_ty\_env \ anc \ nlev)} \]
   \[ \lor \exists \text{con} \in \text{Dom (constants \ curr\_thy)} \]
   \[ \land \text{lookup \ con (tc\_con\_env \ anc \ tylev)} \]
   \[ \text{then \ state} \]
   \[ \text{else \ let (curr\_thy, curr\_hier, thyn\_st, hier\_st, thm\_st)} = \text{dest\_state \ state} \]
   \[ \text{in let curr\_thyn} = \text{current\_theory\_name \ state} \]
   \[ \text{in let f1} = \lambda \text{ti} \in \text{tc\_name (tc\_fetch (ti\_contents \ ti) \ thyn\_st \ tc)} \]
   \[ \text{in let f2} = \lambda \text{ti}(f1 \ ti) \in \text{theory\_ancestors \ state \ thyn} \land \text{ti\_inscope \ ti} \]
   \[ \text{in let b3} = \lambda \text{ti} \in \text{MkTHEORY\_INFO(ti\_status \ ti)(f2 \ ti)(ti\_contents \ ti)} \]
   \[ \text{in let hier'} = \text{Map b3 (che\_fetch curr\_hier hier\_st \ h)} \]
   \[ \text{in let tc} = \text{current\_theory\_contents \ state} \]
   \[ \text{in let (nm, t\_e, e\_e, pars, ax\_d, def\_d, thm\_d, lev, x\_levs, ud)} = \text{dest\_theory\_contents \ tc} \]
   \[ \text{in let tc'} = \text{MkTHEORY\_CONTENTS} \]
   \[ \text{nm t\_e e\_e (Cons \ thyn \ pars) \ ax\_d \ def\_d \ thm\_d \ lev \ x\_levs \ ud} \]
   \[ \text{in let hier\_st'} = \text{(curr\_hier <- hier')} \text{gier\_st} \]
   \[ \text{in let thy\_st'} = \text{(curr\_thy <- tc')} \text{h_thy\_st} \]
   \[ \text{in MkPDS\_STATE curr\_thy curr\_hier thyn\_st' hier\_st' thm\_st} \]

7.4.5 \text{duplicate\_theory}

duplicate\_theory makes a copy of a theory, with the same contents (except for the name) but with no descendants. It takes two arguments, the name of the theory to be duplicated and the name of the copy. In order that the ancestor relations is always rooted, the initial theory may not be duplicated.

The algorithm is as follows:

1. if the name of the theory to be duplicated does not identify an existing theory, or if the name of the copy does, or if the theory to be duplicated is the initial theory, then we leave the state alone.

2. otherwise, we compute a new theory contents from the contents of the theory to be duplicated by changing the name to that of the copy;
3. we allocate space in the theory store for the theory contents computed in step 2;

4. we compute a new theory hierarchy by pushing a `THEORY_INFO` for the new theory onto the current one;

5. we assign the result of step 3 to the current hierarchy

HOL Constant

\[
\text{duplicate\_theory} : \ STRING \rightarrow \ STRING \rightarrow \ ('UD)PDS\_STATE \rightarrow \ ('UD)PDS\_STATE
\]

\[
\forall \text{thy} \ \text{copyn} \ \text{state} \bullet \text{duplicate\_theory thy copyn state} = \\
\begin{align*}
&\text{if} \\
&\quad -\text{thy} \in \text{theory\_names state} \\
&\quad \lor \ \text{copyn} \in \text{theory\_names state} \\
&\quad \lor \ \text{thy} = "MIN" \\
&\text{then state} \\
&\text{else let (cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st) = dest\_state state} \\
&\hspace{1em} \text{in let } tc = \text{etc\_theory\_contents state thy tc} \\
&\hspace{2em} \text{in let (nm, t\_e, c\_e, pars, ax\_d, def\_d, thm\_d, lev, x\_levs, ud) = dest\_theory\_contents tc} \\
&\hspace{3em} \text{in let } tc' = \text{MkTHEORY\_CONTENTS} \\
&\hspace{4em} \text{copyn t\_e c\_e (Cons thy pars) ax\_d def\_d thm\_d lev x\_levs ud} \\
&\hspace{5em} \text{in let (thy\_st', addr) = e(st, a)\bullet} \\
&\hspace{6em} \text{new tc' thy\_st (st, a)} \\
&\hspace{7em} \text{in let } ti = \text{MkTHEORY\_INFO TSNormal F addr} \\
&\hspace{8em} \text{in let hier' = Cons ti (eh\_fetch cur\_hier hier\_st h)} \\
&\hspace{9em} \text{in let hier\_st' = (cur\_hier <- hier') hier\_st} \\
&\hspace{10em} \text{in MkPDS\_STATE cur\_thy cur\_hier thy\_st' hier\_st' thm\_st}
\end{align*}
\]

7.4.6 lock\_theory

`lock\_theory` takes a single parameter which is the name of the theory to lock. A locked theory may not be deleted or have its contents changed.

1. if the name is not the name of any theory, or if the theory it names does not have status `TSNormal`, then we leave the state alone;

2. otherwise, we compute a modified theory hierarchy in which the theory to be locked has status attribute set to `TSLocked`.

3. we assign the result of step 2 to the current hierarchy
HOL Constant

\textit{lock\_theory} : \textit{STRING} → (UD)PDS\_STATE → (UD)PDS\_STATE

\[ ∀\text{thyn state} \cdot \text{lock\_theory thyn state} = \]
\[ \text{if} \quad \neg \text{thyn ∈ theory\_names state} \]
\[ \lor \quad \neg \text{ti\_status(theory\_info state thyn)} = \text{TSLocked} \]
\[ \text{then state} \]
\[ \text{else let} \ (\text{cur\_thy}, \text{cur\_hier}, \text{thy\_st}, \text{hier\_st}, \text{thm\_st}) = \text{dest\_state state} \]
\[ \text{in let} \ \text{ti} = \text{theory\_info state thyn} \]
\[ \text{in let} \ f = \lambda \text{ti} \cdot \]
\[ \quad \text{if} \ \text{ti}' = \text{ti} \]
\[ \quad \text{then MkTHEORY\_INFO TSLocked}(\text{ti\_inscope ti})(\text{ti\_contents ti}) \]
\[ \quad \text{else ti} \]
\[ \text{in let} \ \text{hier}' = \text{Map f (ch\_fetch cur\_hier hier\_st h)} \]
\[ \text{in let} \ \text{hier\_st}' = (\text{cur\_hier} \leftarrow \text{hier}') \text{ hier\_st} \]
\[ \text{in MkPDS\_STATE cur\_thy cur\_hier thy\_st hier\_st' thm\_st} \]

7.4.7 \textit{unlock\_theory}

\textit{unlock\_theory} takes a single parameter which is the name of the theory to unlock.

1. if the name is not the name of any theory, or if the theory it names does not have status \text{TSLocked}, then we leave the state alone;

2. otherwise, we compute a modified theory hierarchy in which the theory to be locked has status attribute set to \text{TSNormal}.

3. we assign the result of step 2 to the current hierarchy

HOL Constant

\textit{unlock\_theory} : \textit{STRING} → (UD)PDS\_STATE → (UD)PDS\_STATE

\[ ∀\text{thyn state} \cdot \text{unlock\_theory thyn state} = \]
\[ \text{if} \quad \neg \text{thyn ∈ theory\_names state} \]
\[ \lor \quad \neg \text{ti\_status(theory\_info state thyn)} = \text{TSLocked} \]
\[ \text{then state} \]
\[ \text{else let} \ (\text{cur\_thy}, \text{cur\_hier}, \text{thy\_st}, \text{hier\_st}, \text{thm\_st}) = \text{dest\_state state} \]
\[ \text{in let} \ \text{ti} = \text{theory\_info state thyn} \]
\[ \text{in let} \ f = \lambda \text{ti} \cdot \]
\[ \quad \text{if} \ \text{ti}' = \text{ti} \]
\[ \quad \text{then MkTHEORY\_INFO TSNormal} (\text{ti\_inscope ti})(\text{ti\_contents ti}) \]
\[ \quad \text{else ti} \]
\[ \text{in let} \ \text{hier}' = \text{Map f (ch\_fetch cur\_hier hier\_st h)} \]
\[ \text{in let} \ \text{hier\_st}' = (\text{cur\_hier} \leftarrow \text{hier}') \text{ hier\_st} \]
\[ \text{in MkPDS\_STATE cur\_thy cur\_hier thy\_st hier\_st' thm\_st} \]
7.5 Operations on Theory Contents

7.5.1 \texttt{save\_thm}

\texttt{save\_thm} takes two parameters. The first parameter is the key under which the theorem is to be saved. The second parameter is the theorem. The theorem is saved in the current theory.

1. we fetch the contents of the current theory;

2. if the key is already in use as a key into the theorem dictionary of the theory fetched in step 1, or if the current theory does not have status \texttt{TSNormal} (e.g. because it is locked), or if the theorem is not in scope (see below), then we leave the state alone.

3. we compute a new theory contents by entering the theorem into the theorem dictionary (which was computed along the way in step 2) under the given key.

4. we assign the new theory contents to the current theory.

Note that we take the level number associated with the stored theorem from the theorem if the theorem belongs to the current theory. We take it as 0 if the theorem does not belong to the current theory (since if it belongs to an ancestor it depends on no definitions in the current theory). Thus, we allow further definitions to be made after a theorem has been inferred but before it is saved, without requiring it to be deleted if some of the subsequent definitions are deleted. There is no particular requirement for this feature, but it is as easy to provide as any other formulation.

Note also that we do not update the theorems proved field, since if the model is correct the theorem must already be in it.

\begin{verbatim}
HOL Constant

\texttt{save\_thm \colon STRING \rightarrow ('UD)PDS\_THM \rightarrow ('UD)PDS\_STATE}

\forall key \; \text{thm} \; \text{state}\bullet \text{save\_thm key thm state} =
\let \text{tc} = \text{current\_theory\_contents state}
\in
\text{if} \; \text{key} \in \text{keys} (\text{tc\_theorem\_dict tc})
\lor \; \text{\neg current\_theory\_status state} = \text{TSNormal}
\lor \; \text{\neg check\_thm state thm}
\text{then state}
\text{else let} \text{(cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st)} = \text{dest\_state state}
\text{in let level = if pt\_theory thm = cur\_thy then pt\_level thm else 0}
\text{in let thm\_d' = enter key (pt\_sequent thm, level) (tc\_theorem\_dict tc)}
\text{in let} \text{(nm, t\_e, c\_e, pars, ax\_d, def\_d, thm\_d, lev, x\_levs, ud)} =
\text{dest\_theory\_contents} \text{ tc}
\text{in let tc' = MkTHEORY\_CONTENTS}
\text{nm t\_e c\_e pars ax\_d def\_d thm\_d' lev x\_levs ud}
\text{in let thy\_st' = (cur\_thy <- tc') thy\_st}
\text{in MkPDS\_STATE cur\_thy cur\_hier thy\_st' hier\_st thm\_st}
\end{verbatim}

29
7.5.2  

```
delete_extension` allows the latest (undeleted) definition or axiom to be deleted from the current theory. Here “definition” is taken to include constants or types introduced with `new_type` or `new_constant` (i.e. which do not have a defining axiom).
```

1. if there is nothing in the current theory to be deleted or if the current theory has children or does not have status `TSNormal`, we leave the state alone;
2. we calculate the most recent level number, `dlev` say, of any object stored in the theory;
3. we remove all definitions and axioms with level number equal to `dlev` from the definition and axiom dictionaries and similarly for the type and constant environments; we increment the current level and add `dlev` to the set of deleted levels;
4. we assign the theory contents computed in the previous step to the current theory.

The following utility is used to assist in step 2:

```
HOL Constant

```
```
is_latest_level : ('UD)PDS_STATE → ℕ → BOOL
```
```
∧ state lev • is_latest_level state lev ⇔
let tc = current_theory_contents state
in let (nm, t_e, c_e, pars, ax_d, def_d, thm_d, lev, x_levs, ud) =
dest_theory_contents tc
in let present = {lv} (\_1 key • lookup key t_e (_1, lv))
∨ (\_1 key • lookup key c_e (_1, lv))
∨ (\_1 key • lookup key ax_d (_1, lv))

in
lev ∈ present ∧ (\forall lv • lv ∈ present ⇒ lv ≤ lev)
```
```
HOL Constant

```
delete_extension : ('UD)PDS_STATE → ('UD)PDS_STATE
```
```
∧ state • delete_extension state =
let tc = current_theory_contents state
in
if (\neg (\exists lev • is_latest_level state lev))
∨ (\exists childname tc • theory_contents state childname tc
∧ current_theory_name state ∈ Elems (tc_parents tc))
∨ \neg current_theory_status state = TSNormal
then state
else let (cur_thy, cur_hier, thy_st, hier_st, thm_st) = dest_state state
in let (nm, t_e, c_e, pars, ax_d, def_d, thm_d, lev, x_levs, ud) =
dest_theory_contents tc
in let dlev = elv • is_latest_level state lv
```
```
4In practice, the user interface to this facility will be capable of recursively deleting definitions and axioms until a desired definition or axiom has been deleted.
```

30
in let def_d' = block_delete \{(\_1, lv)lv = dlev\} def_d
in let ax_d' = block_delete \{(\_1, lv)lv = dlev\} ax_d
in let t_e' = block_delete \{(\_1, lv)lv = dlev\} t_e
in let c_e' = block_delete \{(\_1, lv)lv = dlev\} c_e
in let lev' = lev + 1
in let tc' = MkTHEORY_CONTENTS
    nm t_e' c_e' pars ax_d' def_d' thm_d lev (x_levs \cup \{dlev\}) ud
in let thy_st' = (cur_thy <- tc') thy_st
in MkPDS_STATE cur_thy cur_hier thy_st' hier_st thm_st

7.5.3 `delete_thm`

`delete_thm` deletes a theorem from the current theory. The algorithm is very simple:

1. if the key is not valid for the theorem dictionary for the current theory, or if the current theory does not have status `TSNormal`, we leave the state alone;
2. otherwise, we assign to the current theory a new theory contents in which the indicated theorem has been removed from the theorem dictionary.

SML

HOL Constant

\[ \text{`delete_thm` : STRING } \rightarrow \]
\[ 'UD)PDS\_STATE \rightarrow 'UD)PDS\_STATE \]

\forall key state \bullet \text{`delete_thm` key state =}
  let tc = current_theory_contents state
  in
  if \neg key \in \text{keys (tc_theorem_dict tc)}
  \lor \neg \text{current_theory_status state} = \text{TSNormal}
  then state
  else let (cur_thy, cur_hier, thy_st, hier_st, thm_st) = dest_state state
    in let (nm, t_e, c_e, pars, ax_d, def_d, thm_d, lev, x_levs, ud) =
        dest_theory_contents tc
    in let thm_d' = delete key thm_d
    in let tc' = MkTHEORY_CONTENTS
        nm t_e c_e pars ax_d def_d thm_d' lev x_levs ud
    in let thy_st' = (cur_thy <- tc') thy_st
    in MkPDS_STATE cur_thy cur_hier thy_st' hier_st thm_st

7.5.4 `pds_new_axiom`

`pds_new_axiom` adds a new axiom to a theory. It has two parameters, the first of which is the term giving the new axiom and the second of which gives the key under which the axiom is to be stored. The new axiom is a sequent with no assumptions and with conclusion the given term. The algorithm is:

31
1. if the key is already in use for an axiom in the current theory, or if the new axiom is not a well-formed sequent with respect to the current theory, then we leave the state alone

2. otherwise, let $lev$ be the current level number for the current theory;

3. we assign to the current theory a new theory contents in which the new axiom has been entered in the axiom dictionary at level $lev + 1$ and the new current level is $lev + 1$;

4. we return a result state with the theory store modified by the assignment of step 3 and with the new axiom added to the set of theorems proved.

HOL Constant

\[ \text{pds\_new\_axiom} : \text{TERM} \rightarrow \text{STRING} \rightarrow (\text{'UD})\text{PDS\_STATE} \rightarrow (\text{'UD})\text{PDS\_STATE} \]

\[
\forall \text{tm} \ \text{key} \ \text{state}\cdot \text{pds\_new\_axiom} \ \text{tm} \ \text{key} \ \text{state} = \\
\text{let} \ \text{tc} = \text{current\_theory\_contents} \ \text{state} \\
\text{in} \ \text{let} \ \text{seq} = (\{\}, \ \text{tm}) \\
\text{in} \ \text{if} \ \text{key} \in \text{keys} (\text{tc\_axiom\_dict} \ \text{tc}) \\
\text{\lor} \ \text{\neg} \text{seq} \in \text{sequents} (\text{current\_abstract\_theory} \ \text{state}) \\
\text{\lor} \ \text{\neg} \text{current\_theory\_status} \ \text{state} = \text{TSNormal} \\
\text{then} \ \text{state} \\
\text{else} \ \text{let} (\text{cur\_thy}, \ \text{cur\_hier}, \ \text{thy\_st}, \ \text{hier\_st}, \ \text{thm\_st}) = \text{dest\_state} \ \text{state} \\
\text{in} \ \text{let} (\text{nm}, \ \text{t\_e}, \ \text{c\_e}, \ \text{pars}, \ \text{ax\_d}, \ \text{def\_d}, \ \text{thm\_d}, \ \text{lev}, \ \text{x\_levs}, \ \text{ud}) = \\
\text{dest\_theory\_contents} \ \text{tc} \\
\text{in} \ \text{let} \ \text{lev}' = \text{lev} + 1 \\
\text{in} \ \text{let} \ \text{ax\_d}' = \text{enter} \ \text{key} (\text{seq}, \ \text{lev}') \ \text{ax\_d} \\
\text{in} \ \text{let} \ \text{tc}' = \text{MkTHEORY\_CONTENTS} \\
\text{nm} \ \text{t\_e} \ \text{c\_e} \ \text{pars} \ \text{ax\_d}' \ \text{def\_d} \ \text{thm\_d} \ \text{lev}' \ \text{x\_levs} \ \text{ud} \\
\text{in} \ \text{let} \ \text{thy\_st}' = (\text{cur\_thy} < - \text{tc}') \ \text{thy\_st} \\
\text{in} \ \text{let} (\text{thm\_st}', \ _1) = \text{eps\_new} (\text{pds\_mk\_thm} \ \text{state} \ \text{seq}) \ \text{thm\_st} \ \text{sa} \\
\text{in} \ \text{MkPDS\_STATE} \ \text{cur\_thy} \ \text{cur\_hier} \ \text{thy\_st}' \ \text{hier\_st} \ \text{thm\_st}'
\]

7.5.5 General Definitional Mechanisms

The definitional mechanisms to be supplied will be closely based on the ones identified in [6]. We wish to defer the formal specification of the mechanisms which introduce new definitional axioms, while still specifying something about their role in our model of the system. To do this we supply a “generic” definitional mechanism, the function make_definition below, which is parameterised by a function (called a DEFINER) which represents an implementation of the definitional mechanisms. To avoid complicating the handling of definitional axioms, the “definitional” mechanisms new_type and new_constant which do not introduce new definitional axioms are defined here.

The input to the DEFINER has the following type, which is also used in section 7.6 below:

SML

[declare_type_abbrev("SUBSYS\_INPUT", ["IP", "UD"], "IP × (\text{'UD})\text{PDS\_THM \ LIST}\);
The input to the system which is used to derive the input to the `DEFINER` has the type:

```sml
declare_type_abbrev("PDS_INPUT", ["IP", "UD"], "; IP \times (UD)PDS_THM_ADDR LIST");
```

The addresses in a `PDS_INPUT` are intended to be addresses for the theorem store in the state. The parameter then has the type:

```sml
declare_type_abbrev("DEFINER", ["IP", "UD"], 
  "IP \times (IP \times UD)SUBSYS_INPUT \times (UD)PDS_STATE \mapsto 
   (SEQ \times ((STRING \times N) LIST) \times ((STRING \times TYPE) LIST) \times (STRING LIST))")
```

Thus, a `DEFINER` is a partial function, represented as a set of pairs, which computes a 4-tuple comprising:

- a sequent which is to be the definitional axiom resulting from the definition;
- a list of type names and arities for any new types introduced by the definition;
- a list of constant names and types for any new constants introduced by the definition;
- a list of keys under which the definitional axiom is to be saved on the theory.

The algorithm for `make_definition` is as follows:

1. if the input and state are not in the domain of the `DEFINER`, or if the theorems in the input are not all valid in the current theory then leave the state alone;
2. otherwise, apply the `DEFINER` to the input-state pair;
3. let `lev` be the current level number;
4. using the result of step 2 and the key parameter, compute a modified theory contents from the current theory contents; the modified theory contents has level number `lev + 1` and has the new definition (with level number `lev + 1`) and new type and constant dictionary entries as returned by the `DEFINER`;
5. assign the result of step 4 to the current theory;
6. return a state with the theory store modified as in step 5 and with the new definitional axiom added to the set of theorems proved.
HOL Constant

```
make_definition : ('IP, 'UD)DEFINER →
((('IP, 'UD)PDS_INPUT × ('UD)PDS_STATE) → ('UD)PDS_STATE)

∀definer pars thm_ads state
make_definition definer ((pars, thm_ads), state) =
if ∃thm_ad•thm_ad ∈ Elems thm_ads ∧ ¬check_thm_address state thm_ad
then state
else let thms = fetch_thms state thm_ads
in
if ¬((pars, thms), state) ∈ Dom definer
then state
else let (seq, tys, cons, ks) = definer©((pars, thms), state)
in let (cur_thy, cur_hier, thy_st, hier_st, thm_st) = dest_state state
in let tc = current_theory_contents state
in let (nm, t_e, c_e, pars, ax_d, def_d, thm_d, lev, x_levs, ud) =
dest_theory_contents tc
in let lev' = lev+1
in let def_d' = Fold (λk•enter k (seq, lev')) ks def_d
in let t_e' = Fold (Uncurry enter) (Map (λ(s,x)•(s, (x, lev')))) tys t_e
in let c_e' = Fold (Uncurry enter) (Map (λ(s,x)•(s, (x, lev')))) cons c_e
in let tc' = MKTHEORY_CONTENTS
    nm t_e' c_e' pars ax_d def_d' thm_d lev' x_levs ud
in let thy_st' = (cur_thy <- tc') thy_st
in let (thm_st', _1) = εsa•new(pds_mk_thm state seq)thm_st sa
in MkPDS_STATE cur_thy cur_hier thy_st' hier_st thm_st'
```

7.5.6 pds_new_type

`pds_new_type` introduce a new type with a given arity without any associated definitional axiom. It takes two parameters, the first being the name of the type and the second being the arity. A type with the same name as a type which is already in scope in the current theory is not allowed.

1. if some ancestor of the current theory contains a type with the same name then we leave the state alone;
2. otherwise, let `lev` be the current level number;
3. using the result of step 2 and the parameters, compute a modified theory contents from the current theory contents; the modified theory contents has level number `lev + 1` and a new type entry for the new type;
4. assign the result of step 4 to the current theory;
5. return a state with the theory store modified as in step 5.
HOL Constant

\[
pds\_new\_type : \quad STRING \to \mathbb{N} \to (‘UD)PDS\_STATE \to (‘UD)PDS\_STATE
\]

\[
\forall ty\ \text{arity state}\cdot \\
pds\_new\_type\ ty\ \text{arity state} = \\
\quad \text{if } ty \in \text{Dom(types (current_abstract_theory state))} \text{ then state} \\
\quad \text{else let } (cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st) = \text{dest_state state} \\
\quad \text{in let } tc = \text{current_theory_contents state} \\
\quad \text{in let } (nm, t\_e, c\_e, pars, ax\_d, def\_d, thm\_d, lev, x\_levs, ud) = \\
\quad \quad \text{dest_theory_contents tc} \\
\quad \text{in let } lev' = lev + 1 \\
\quad \text{in let } t\_e' = \text{enter ty (arity, lev') } t\_e \\
\quad \text{in let } tc' = \text{MkTHEORY\_CONTENTS} \\
\quad \quad nm\ t\_e'\ c\_e\ pars\ ax\_d\ def\_d\ thm\_d\ lev'\ x\_levs\ ud \\
\quad \text{in let } thy\_st' = (\text{cur}\_thy\ <\ -\ tc')\ thy\_st \\
\quad \text{in MkPDS\_STATE cur_thy cur_hier thy_st' hier_st thm_st}
\]

7.5.7 \textit{pds\_new\_constant}

\textit{pds\_new\_constant} introduce a new constant with a given type without any associated definitional axiom. It takes two parameters, the first being the name of the constant and the second being the type. A constant with the same name as a constant which is already in scope in the current theory is not allowed.

1. if some ancestor of the current theory contains a constant with the same name, or if the supplied type of the constant is not well-formed with respect to the current theory, then we leave the state alone;
2. otherwise, let \textit{lev} be the current level number;
3. using the result of step 2 and the parameters, compute a modified theory contents from the current theory contents; the modified theory contents has level number \textit{lev} + 1 and a new constant entry for the new constant;
4. assign the result of step 4 to the current theory;
5. return a state with the theory store modified as in step 5.
\begin{verbatim}
pds_new_constant : STRING → TYPE → ('UD)PDS_STATE → ('UD)PDS_STATE
\end{verbatim}

\[\forall \text{con ty state}\cdot \]
\[\text{pds_new_constant con ty state} = \]
\[\text{if } \text{con} \in \text{Dom}\left(\text{constants}\left(\text{current}\_\text{abstract}\_\text{theory}\ \text{state}\right)\right) \land \]
\[\forall \neg \text{ty} \in \text{wf}\_\text{type}\left(\text{types}\left(\text{current}\_\text{abstract}\_\text{theory}\ \text{state}\right)\right) \]
\[\text{then state} \]
\[\text{else let } (\text{cur_thy}, \text{cur_hier}, \text{thy_st}, \text{hier_st}, \text{thm_st}) = \text{dest}\_\text{state}\ \text{state} \]
\[\text{in let } tc = \text{current}\_\text{theory}\_\text{contents}\ \text{state} \]
\[\text{in let } (\text{nm}, \text{t}_e, \text{c}_e, \text{pars}, \text{ax}_d, \text{def}_d, \text{thm}_d, \text{lev}, x\_\text{levs}, \text{ud}) = \]
\[\text{dest}\_\text{theory}\_\text{contents}\ \text{tc} \]
\[\text{in let lev'} = \text{lev} + 1 \]
\[\text{in let } \text{c}_e' = \text{enter}\ \text{con}\ (\text{ty}, \text{lev'}) \ \text{c}_e \]
\[\text{in let } \text{tc}' = \text{MkTHEORY}\_\text{CONTENTS} \]
\[\text{nm} \ \text{t}_e \ \text{c}_e' \ \text{pars} \ \text{ax}_d \ \text{def}_d \ \text{thm}_d \ \text{lev} \ \text{z}\_\text{levs} \ \text{ud} \]
\[\text{in let } (\text{thy_st}') = (\text{cur_thy} \leftarrow \text{tc}') \ \text{thy_st} \]
\[\text{in } \text{MkPDS}\_\text{STATE} \ \text{cur_thy} \ \text{cur_hier} \ \text{thy_st}' \ \text{hier_st} \ \text{thy_st} \]

7.6 Inference

The inference rules to be supplied will typically comprise primitive rules implementing the rules specified in [5] together with rules which define string and other literals and rules which, while they could be derived from the primitive rules, are built-in for reasons of efficiency. As a very special case, we consider the inference rules to include the functions which given a theory name and a key return the axiom (or definition or theorem) stored under that key in the indicated theory.

As with the definitional mechanisms, we wish to defer specification of the rules, and so we complete the present specification by defining a “generic” inference function, \textit{make_inference}, parameterised by a function which represents an implementation of such a set of rules.

\begin{verbatim}
declare_type_abbrev("INFERRER", ["'IP", "'UD"],
\text{"'"}:(''IP', 'UD)\text{SUBSYS}\_\text{INPUT} \times (''UD)\text{PDS}\_\text{STATE} \rightarrow \text{SEQ}");
\end{verbatim}

An \textit{INFERRER} is a partial function, represented as a set of pairs, which, returns a sequent.

The algorithm for \textit{make_inference} is as follows:

1. if the input and state are not in the domain of the \textit{INFERRER}, or if the theorems in the input are not all valid in the current theory then leave the state alone;
2. otherwise, apply the \textit{INFERRER} to the input-state pair;
3. compute a theorem, with the current theory as its theory field, the current level number as its level field, and with the result of step 2 as its sequent field;
4. return a state with the result of step 3 added to the theorems proved field.

HOL Constant

\[
\text{make\_inference} : \quad (\text{'IP, 'UD})\text{INFERRER} \to \\
((\text{'IP, 'UD})\text{PDS\_INPUT} \times (\text{'UD})\text{PDS\_STATE}) \to \\
(\text{'UD})\text{PDS\_STATE}
\]

\[
\forall \text{inferrer \ pars \ thm\_ads \ state} \bullet \\
\text{make\_inference} \ \text{inferrer} \ ((\text{pars, thm\_ads}), \text{state}) = \\
\begin{array}{ll}
\text{if} & (\exists \text{thm\_ad} \bullet \text{thm\_ad} \in \text{Elems thm\_ads} \land \lnot \text{check\_thm\_address state thm\_ad}) \\
\text{then} & \text{state} \\
\text{else} & \text{let thms} = \text{fetch\_thms state thm\_ads} \\
& \text{in} \\
& \begin{array}{ll}
\text{if} & \lnot((\text{pars, thms}), \text{state}) \in \text{Dom inferrer} \\
\forall & \lnot \text{current\_theory\_status state} = \text{TSNormal} \\
\text{then} & \text{state} \\
\text{else} & \text{let seq} = \text{inferrer@((pars, thms), state)} \\
& \text{in let} \ (\text{cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st}) = \text{dest\_state state} \\
& \text{in let} \ (\text{thy\_st'}, \_1) = \epsilon \text{so\_new(pds\_mk\_thm state seq)thy\_st sa} \\
& \text{in MkPDS\_STATE cur\_thy cur\_hier thy\_st hier\_st thm\_st'}
\end{array}
\end{array}
\]

8 SYSTEM CONSTRUCTION

We have defined the operations on the state in terms of two subsystems: the definitional mechanisms and the inference rules. We now wish to say how the operations and two such subsystems are to be combined to produce a system.

8.1 Auxiliary Definitions

We will say that a state-to-state transition function is \textit{allowed}, if it is one of the operations on states defined in section 7 above. Since some of these operations are parameterised by the definitional mechanism or inference rules, so is this property:

HOL Constant

\[
\text{allowed} : \quad (\text{'IP, 'UD})\text{DEFINER} \to \\
(\text{'IP, 'UD})\text{INFERRER} \to \\
((\text{'UD})\text{PDS\_STATE} \to (\text{'UD})\text{PDS\_STATE}) \to \text{BOOL}
\]

\[
\forall \text{definer inferrer trans} \bullet \\
\text{allowed definer inferrer trans} \Leftrightarrow \\
\begin{array}{ll}
\text{trans} & = \text{freeze\_hierarchy} \\
\forall & \text{trans} = \text{new\_hierarchy} \\
\forall & \exists \text{addr} \bullet \text{trans} = \text{load\_hierarchy addr} \\
\forall & \exists \text{thyn} \bullet \text{trans} = \text{open\_theory thyn} \\
\forall & \exists \text{thyn} \bullet \text{trans} = \text{delete\_theory thyn}
\end{array}
\]
8.2 The System Construction

The construction of the system also involves a third subsystem: a “command interpreter”, which, given a DEFINER and an INFERRER maps inputs onto transition functions. Thus it has the following type:

\[
\text{SML} \quad \text{declare\_type\_abbrev("INTERPRETER", ["IP", "UD"],}
\]
\[
\begin{align*}
& \quad r:(\text{"IP", "UD"})\text{DEFINER} \to (\text{"IP", "UD"})\text{INFERRER} \to \\
& \quad (\text{"IP", "UD"})\text{PDS\_INPUT} \to \\
& \quad (\text{"UD"})\text{PDS\_STATE} \to (\text{"UD"})\text{PDS\_STATE});
\end{align*}
\]

The loosely specified function \textit{pds} constructs a system from a DEFINER, an INFERRER and an INTERPRETER. After expanding the type abbreviations, the systems it constructs may be seen to have the following type:

\[
\begin{align*}
\text{Discussion} : & \quad (\text{"IP" \times (\text{"UD"})\text{PDS\_THM\ list}}) \times (\text{"UD"})\text{PDS\_STATE}) \\
& \quad \to (\text{"UD"})\text{PDS\_STATE} \times ((\text{"UD"})\text{PDS\_THM\ STORE}))
\end{align*}
\]

Thus inputs to the system are composed of unspecified “parameters”, together with lists of theorems. Its output is taken to be the theorem store (which, in practice, certainly includes any theorem returned by one of the constructors of the abstract datatype).

\textit{pds} constructs \textit{HOL\_SYSTEM}s in the sense of [8], allowing us to assert the critical properties defined in that document for these systems.

\[
\text{HOL\ Constant} \quad \text{pds} : (\text{"IP", "UD"})\text{DEFINER} \to \\
(\text{"IP", "UD"})\text{INFERRER} \to \\
(\text{"IP", "UD"})\text{INTERPRETER} \to \\
(\text{"IP", "UD"})\text{PDS\_INPUT}, \\
(\text{"UD"})\text{PDS\_THM\ STORE}, \\
(\text{"UD"})\text{PDS\_STATE} \quad )\text{HOL\_SYSTEM}
\]
\forall \text{definer inferrer interpreter} •
\text{pds definer inferrer interpreter} =
( (\lambda((\text{pars}, \text{thm}_\text{ads}), \text{state}) •
  \text{let state}' = \text{interpreter definer inferrer} (\text{pars}, \text{thm}_\text{ads}) \text{ state}
  \text{in} (\text{state}', \text{ps_theorem_store state'})),
  \text{interpret \_state} )

8.3 Subsystem Critical Properties

We will use the term \textit{good} of subsystems which satisfy their critical properties. The critical properties can be expressed either syntactically or semantically. We choose the semantic formulation, which is felt to be somewhat simpler.

A \textit{DEFINER} is good if the extension of abstract theories induced by its intended effect on concrete theories is definitional:

HOL Constant

\textbf{good\_definer} : (‘IP, ‘UD)\textit{DEFINER} \to \textit{BOOL}

\forall \text{definer • good\_definer definer} \leftrightarrow
\forall \text{pars thms state} •
  ( (\text{pars, thms}, \text{state}) \in \text{Dom definer}
  \Rightarrow
  \text{let thy} = \text{current\_abstract\_theory state}
  \text{in} \text{let (tyenv, conenv, axs)} = \text{rep\_theory thy}
  \text{in} \text{let (seq, tys, cons, _J) = definer@((\text{pars, thms}), \text{state})}
  \text{in} \text{let tyenv'} = \text{Fold} (\lambda m • \lambda te • te ∪ \{tn\}) \text{ tys tyenv}
  \text{in} \text{let conenv'} = \text{Fold} (\lambda st • \lambda ce • ce ∪ \{st\}) \text{ cons conenv}
  \text{in} \text{let axs'} = \text{axs} ∪ \{seq\}
  \text{in} \text{let thy'} = \text{abs\_theory(tyenv', conenv', axs')}
  \text{in} \text{thy' \in definitional\_extension thy'}

An \textit{INFERRER} is good if the sequent it computes is always (a) valid with respect to the current abstract theory and the theorems it is given as part of its parameter and (b) is well-formed with respect to the current abstract theory. As in the definition of \textit{valid} in [6], an apparently unused parameter must be used to ensure that the type of the universe of models appears in the type of \textit{good\_inferrer}.

HOL Constant

\textbf{good\_inferrer} : ‘U \to (‘IP, ‘UD)\textit{INFERRER} \to \textit{BOOL}

\forall \text{v inferrer • good\_inferrer v inferrer} \leftrightarrow
\forall \text{pars thms state} •
  ( (\text{pars, thms}, \text{state}) \in \text{Dom inferrer}
  \Rightarrow
  \text{let thy} = \text{current\_abstract\_theory state}
  \text{in} \text{let seq = inferrer@((\text{pars, thms}), \text{state})}
  \text{in} ( \text{seq} \in \text{sequents thy}
  \wedge \text{seq} \in \text{valid v thy} )

39
An \textit{INTERPRETER} is good if it always returns allowable state transitions.

HOL Constant

\begin{center}
\texttt{good}\textsubscript{\textit{interpreter}} : ('IP, 'UD)\textit{INTERPRETER} \rightarrow BOOL
\end{center}

\begin{center}
\forall \text{interpreter}\bullet \text{good}\textsubscript{\textit{interpreter}} \leftrightarrow \\
\forall \text{definer}\text{\textsubscript{inferrer}} \text{\textsubscript{pars}} \text{\textsubscript{thm}\textsubscript{ads}}\bullet \\
\text{allowed definier inferrer(\text{interpreter definier inferrer(pars, thm_ads))}}
\end{center}

In terms of these notions of goodness we may now formulate a conjecture that good components, according to the above syntactic definitions, make a system meeting its critical requirements either in the semantic formulation:

Conjecture

\begin{center}
\begin{align*}
\forall \text{definer}\text{\textsubscript{inferrer}} \text{\textsubscript{interpreter}}\bullet \\
& ( \text{good}\textsubscript{\textit{definer}} \text{\textsubscript{definer}} \\
& \land \text{good}\textsubscript{\textit{inferrer}} \text{\textsubscript{inferrer}} \\
& \land \forall v:U\bullet \text{good}\textsubscript{\textit{interpreter}} v \text{\textsubscript{interpreter}}) \\
\Rightarrow & ( \text{standard (pds definier inferrer interpreter)} \\
& \land \forall v:U\bullet \text{validity}\textsubscript{\textit{preserving}} v \text{\textsubscript{pds definier inferrer interpreter}})
\end{align*}
\end{center}

or in the syntactic formulation:

Conjecture

\begin{center}
\begin{align*}
\forall \text{definer}\text{\textsubscript{inferrer}} \text{\textsubscript{interpreter}}\bullet \\
& ( \text{good}\textsubscript{\textit{definer}} \text{\textsubscript{definer}} \\
& \land \text{good}\textsubscript{\textit{inferrer}} \text{\textsubscript{inferrer}} \\
& \land \forall v:U\bullet \text{good}\textsubscript{\textit{interpreter}} v \text{\textsubscript{interpreter}}) \\
\Rightarrow & ( \text{standard (pds definier inferrer interpreter)} \\
& \land \text{derivability}\textsubscript{\textit{preserving}} \text{\textsubscript{pds definier inferrer interpreter}})
\end{align*}
\end{center}
9 THEORY LISTING

10 THE THEORY spc005

10.1 Parents

spc004

10.2 Constants

initial_dict  \((\text{CHAR LIST} \times \text{'}X\text{')} \times \text{SET})\) 
enter \(\text{CHAR LIST}
\text{'}X
\text{'}(\text{CHAR LIST} \times \text{'}X) \times \text{SET})\)
lookup \(\text{CHAR LIST} \times (\text{CHAR LIST} \times \text{'}X) \times \text{SET} \rightarrow \text{'}X \rightarrow \text{BOOL})\)
delete \(\text{CHAR LIST} \rightarrow (\text{CHAR LIST} \times \text{'}X) \times \text{SET} \rightarrow (\text{CHAR LIST} \times \text{'}X) \times \text{SET})\)
block_delete \(\text{'}X \times \text{SET} \rightarrow (\text{CHAR LIST} \times \text{'}X) \times \text{SET} \rightarrow (\text{CHAR LIST} \times \text{'}X) \times \text{SET})\)
keys \(\text{CHAR LIST} \times \text{'}X \times \text{SET} \rightarrow \text{CHAR LIST} \times \text{SET})\)
$\text{\texttt{\textasciitilde}}$ \(\text{'}X \times \text{ADDR} \times \text{'}X \times \text{SET} \rightarrow (\text{'}X \times \text{ADDR} \times \text{'}X \times \text{SET})\)
fetch \(\text{'}X \times \text{ADDR} \times \text{'}X \times \text{SET} \rightarrow \text{'}X \rightarrow \text{BOOL})\)
new \(\text{'}X
\text{'}(\text{'}X \times \text{ADDR} \times \text{'}X \times \text{SET})\)
\(\rightarrow \text{'}X \times \text{ ADDR} \times \text{'}X \times \text{SET} \times \text{'}X \times \text{ ADDR} \rightarrow \text{BOOL})\)
initial_store \(\text{'}X \times \text{ADDR} \times \text{'}X \times \text{SET})\)
tc_user_data \(\text{UD THEORY\_CONTENTS} \rightarrow \text{'}UD\text{UD THEORY\_CONTENTS} \rightarrow \text{N SET})\)
tc_deleted_levels \(\text{UD THEORY\_CONTENTS} \rightarrow \text{N})\)
tc_current_level \(\text{UD THEORY\_CONTENTS} \rightarrow \text{N})\)
tc_theorem_level \(\text{UD THEORY\_CONTENTS} \rightarrow \text{UD THEORY\_CONTENTS} \times (\text{TERM SET} \times \text{TERM}) \times \text{N}) \times \text{SET})\)
tc_definition_dict \(\text{UD THEORY\_CONTENTS} \rightarrow \text{UD THEORY\_CONTENTS} \times (\text{TERM SET} \times \text{TERM}) \times \text{N}) \times \text{SET})\)
tc_axiom_dict \(\text{UD THEORY\_CONTENTS} \rightarrow \text{UD THEORY\_CONTENTS} \times (\text{TERM SET} \times \text{TERM}) \times \text{N}) \times \text{SET})\)
tc_parents \(\text{UD THEORY\_CONTENTS} \rightarrow \text{CHAR LIST} \times \text{SET})\)
tc_con_env \(\text{UD THEORY\_CONTENTS} \rightarrow \text{CHAR LIST} \times \text{TYPE} \times \text{N}) \times \text{SET})\)
tc_ty_env \(\text{UD THEORY\_CONTENTS} \rightarrow \text{CHAR LIST} \times \text{CHAR LIST} \times \text{N} \times \text{N}) \times \text{SET})\)
tc_name \(\text{UD THEORY\_CONTENTS} \rightarrow \text{CHAR LIST})\)
MkTHEORY\_CONTENTS \(\text{CHAR LIST}
\rightarrow \text{CHAR LIST} \times \text{N} \times \text{N}) \times \text{SET})\)
\(\rightarrow \text{CHAR LIST} \times \text{N} \times \text{N}) \rightarrow \text{SET})\)
\(\rightarrow \text{'}UD\text{)}
→ 'UD THEORY_CONTENTS
TSDel ected  ONE + ONE + ONE + ONE
TSA ncestor  ONE + ONE + ONE + ONE
TSLocked  ONE + ONE + ONE + ONE
TSN ormal  ONE + ONE + ONE + ONE
ti_contents  'UD THEORY_INFO → 'UD THEORY_CONTENTS_ADDR
ti_inscope  'UD THEORY_INFO → BOOL
ti_status  'UD THEORY_INFO → ONE + ONE + ONE + ONE
MkTHEORY_INFO
  ONE + ONE + ONE + ONE
  → BOOL
  → 'UD THEORY_CONTENTS_ADDR
  → 'UD THEORY_INFO
pt_sequent  'UD PDS_THM → TERM SET × TERM
pt_level  'UD PDS_THM → \textbb{N}
pt_theory  'UD PDS_THM → 'UD THEORY_CONTENTS_ADDR
MkPDS_THM  'UD THEORY_CONTENTS_ADDR
  → \textbb{N}
  → TERM SET × TERM
  → 'UD PDS_THM
ps_theorem_store  'UD PDS_STATE → ('UD PDS_THM_ADDR × 'UD PDS_THM) SET
ps_hierarchy_store  'UD PDS_STATE
  → ('UD THEORY_INFO_LIST_ADDR × 'UD THEORY_INFO_LIST) SET
ps_theory_store  'UD PDS_STATE
  → ('UD THEORY_CONTENTS_ADDR × 'UD THEORY_CONTENTS) SET
ps_current_hierarchy  'UD PDS_STATE → 'UD THEORY_INFO_LIST_ADDR
ps_current_theory  'UD PDS_STATE → 'UD THEORY_CONTENTS_ADDR
MkPDS_STATE  'UD THEORY_CONTENTS_ADDR
  → 'UD THEORY_INFO_LIST_ADDR
  → ('UD THEORY_CONTENTS_ADDR × 'UD THEORY_CONTENTS) SET
  → ('UD THEORY_INFO_LIST_ADDR × 'UD THEORY_INFO_LIST) SET
  → ('UD PDS_THM_ADDR × 'UD PDS_THM) SET
  → 'UD PDS_STATE
initial_theory  'UD
  → ('UD THEORY_CONTENTS_ADDR × 'UD THEORY_CONTENTS) SET
  × 'UD THEORY_INFO
initial_state  'UD → 'UD PDS_STATE
theory_contents  'UD PDS_STATE → CHAR LIST → 'UD THEORY_CONTENTS → BOOL
theory_names  'UD PDS_STATE → CHAR LIST SET
theory_ancestors  'UD PDS_STATE → CHAR LIST → CHAR LIST SET
interpret_theory_contents  'UD THEORY_CONTENTS_SET
  → THEORY
  × (TERM SET × TERM) SET
× (TERM SET × TERM) SET

interpret_state

'UD PDS_STATE → THEORY_HIERARCHY

dest_state

'UD PDS_STATE

→ 'UD THEORY_CONTENTS_ADDR

× 'UD THEORY_INFO_LIST_ADDR

× ('UD THEORY_CONTENTS_ADDR × 'UD THEORY_CONTENTS) SET

× ('UD THEORY_INFO_LIST_ADDR

× 'UD THEORY_INFO_LIST) SET

× ('UD PDS_THM_ADDR × 'UD PDS_THM) SET

dest_theory_contents

'UD THEORY_CONTENTS

→ CHAR_LIST

× (CHAR_LIST × N × N) SET

× (CHAR_LIST × TYPE × N) SET

× CHAR_LIST_LIST

× (CHAR_LIST × (TERM_SET × TERM) × N) SET

× (CHAR_LIST × (TERM_SET × TERM) × N) SET

× (CHAR_LIST × (TERM_SET × TERM) × N) SET

× N

× N SET

× 'UD
current_theory_contents

'UD PDS_STATE → 'UD THEORY_CONTENTS
current_theory_name

'UD PDS_STATE → CHAR LIST
current_abstract_theory

'UD PDS_STATE → THEORY
theory_info

'UD PDS_STATE → CHAR_LIST → 'UD THEORY_INFO
current_theory_status

'UD PDS_STATE → ONE + ONE + ONE + ONE
check_thm

'UD PDS_STATE → 'UD PDS_THM → BOOL
check_thm_address

'UD PDS_STATE → 'UD PDS_THM_ADDR → BOOL

fetch_thms

'UD PDS_STATE

→ 'UD PDS_THM_ADDR_LIST

→ 'UD PDS_THM_LIST

hierarchy_ancestor

'UD PDS_STATE

→ 'UD THEORY_INFO_LIST_ADDR

→ 'UD THEORY_INFO_LIST_ADDR

→ BOOL
pds_mk_thm

'UD PDS_STATE → TERM_SET × TERM → 'UD PDS_THM

make_current

CHAR_LIST → 'UD PDS_STATE → 'UD PDS_STATE

freeze_hierarchy

'UD PDS_STATE

→ 'UD PDS_STATE

new_hierarchy

'UD PDS_STATE → 'UD PDS_STATE

load_hierarchy

'UD THEORY_INFO_LIST_ADDR

→ 'UD PDS_STATE

→ 'UD PDS_STATE

open_theory

CHAR_LIST → 'UD PDS_STATE → 'UD PDS_STATE

empty_theory

CHAR_LIST → CHAR_LIST_LIST → 'UD → 'UD THEORY_CONTENTS

arbitrary_ud

'UD

delete_theory

CHAR_LIST → 'UD PDS_STATE → 'UD PDS_STATE
new\_theory CHAR\ LIST → 'UD → UD PDS\_STATE → UD PDS\_STATE
new\_parent CHAR\ LIST → 'UD PDS\_STATE → UD PDS\_STATE
duplicate\_theory
  CHAR\ LIST → CHAR\ LIST → UD PDS\_STATE → UD PDS\_STATE
lock\_theory CHAR\ LIST → UD PDS\_STATE → UD PDS\_STATE
unlock\_theory CHAR\ LIST → UD PDS\_STATE → UD PDS\_STATE
save\_thm CHAR\ LIST → UD PDS\_THM → UD PDS\_STATE → UD PDS\_STATE
is\_latest\_level UD PDS\_STATE → N → BOOL
delete\_extension UD PDS\_STATE → UD PDS\_STATE
delete\_thm
  CHAR\ LIST → UD PDS\_STATE → UD PDS\_STATE
pds\_new\_axiom
  TERM → CHAR\ LIST → UD PDS\_STATE → UD PDS\_STATE
make\_definition
  (((IP × UD PDS\_THM\ LIST) × UD PDS\_STATE)
    × (TERM\ SET × TERM)
    × (CHAR\ LIST × N)\ LIST
    × (CHAR\ LIST × TYPE)\ LIST
    × CHAR\ LIST\ LIST)\ SET
  → (IP × UD PDS\_THM\ ADDR\ LIST) × UD PDS\_STATE
  → UD PDS\_STATE
pds\_new\_type
  CHAR\ LIST → N → UD PDS\_STATE → UD PDS\_STATE
pds\_new\_constant
  CHAR\ LIST → TYPE → UD PDS\_STATE → UD PDS\_STATE
make\_inference
  (((IP × UD PDS\_THM\ LIST) × UD PDS\_STATE)
    × (TERM\ SET)
    × TERM)\ SET
  → (IP × UD PDS\_THM\ ADDR\ LIST) × UD PDS\_STATE
  → UD PDS\_STATE
allowed
  (((IP × UD PDS\_THM\ LIST) × UD PDS\_STATE)
    × (TERM\ SET × TERM)
    × (CHAR\ LIST × N)\ LIST
    × (CHAR\ LIST × TYPE)\ LIST
    × CHAR\ LIST\ LIST)\ SET
  → (((IP × UD PDS\_THM\ LIST) × UD PDS\_STATE)
    × TERM\ SET
    × TERM)\ SET
  → (UD PDS\_STATE → UD PDS\_STATE)
  → BOOL
pds
  (((IP × UD PDS\_THM\ LIST) × UD PDS\_STATE)
    × (TERM\ SET × TERM)
    × (CHAR\ LIST × N)\ LIST
    × (CHAR\ LIST × TYPE)\ LIST
    × CHAR\ LIST\ LIST)\ SET
  → (((IP × UD PDS\_THM\ LIST) × UD PDS\_STATE)
    × TERM\ SET
    × TERM)\ SET
  → (((IP × UD PDS\_THM\ LIST) × UD PDS\_STATE)
    × (TERM\ SET × TERM)
    × (CHAR\ LIST × N)\ LIST
    × (CHAR\ LIST × TYPE)\ LIST
    × CHAR\ LIST\ LIST)\ SET
  → (((IP × UD PDS\_THM\ LIST) × UD PDS\_STATE)
    × TERM\ SET
    × TERM)\ SET
→ 'IP × 'UD PDS_THM ADDR LIST
→ 'UD PDS_STATE
→ 'UD PDS_STATE
→ ('IP × 'UD PDS_THM ADDR LIST) × 'UD PDS_STATE
→ 'UD PDS_STATE
× ('UD PDS_THM ADDR × 'UD PDS_THM) SET
× ('UD PDS_STATE → THEORY_HIERARCHY)
good_definer
(((IP × 'UD PDS_THM LIST) × 'UD PDS_STATE)
× (TERM SET × TERM)
× (CHAR LIST × N) LIST
× (CHAR LIST × TYPE) LIST
× CHAR LIST LIST) SET
→ BOOL

good_inerrer

'U
→ (((IP × 'UD PDS_THM LIST) × 'UD PDS_STATE)
× TERM SET
× TERM) SET
→ BOOL
good_interpreter
(((IP × 'UD PDS_THM LIST) × 'UD PDS_STATE)
× (TERM SET × TERM)
× (CHAR LIST × N) LIST
× (CHAR LIST × TYPE) LIST
× CHAR LIST LIST) SET
→ (((IP × 'UD PDS_THM LIST) × 'UD PDS_STATE)
× TERM SET
× TERM) SET
→ 'IP × 'UD PDS_THM ADDR LIST
→ 'UD PDS_STATE
→ 'UD PDS_STATE)
→ BOOL

10.3 Types

'1 ADDR
'1 THEORY_CONTENTS
'1 THEORY_INFO
'1 PDS_THM
'1 PDS_STATE

10.4 Type Abbreviations

'X DICT (CHAR LIST × 'X) SET
'X STORE ('X ADDR × 'X) SET
STATUS ONE + ONE + ONE + ONE
'UD HIERARCHY
'UD THEORY_INFO LIST
('IP, 'UD) SUBSYS_INPUT
'IP × 'UD PDS_THM LIST
('IP, 'UD) PDS_INPUT
'IP × 'UD PDS_THM ADDR LIST
('IP, 'UD) DEFINER
(((IP × 'UD PDS_THM LIST) × 'UD PDS_STATE)
× (TERM SET × TERM)
× (CHAR LIST × N) LIST
× (CHAR LIST × TYPE) LIST

45
10.5 Fixity

*Right Infix 300:*

\<-

10.6 Definitions

initial_dict \(\vdash\) initial_dict = {}

enter \(\vdash\) \(\forall\) key item dict

\(\bullet\) enter key item dict = dict \(\uplus\) \{\{key, item\}\}

lookup \(\vdash\) \(\forall\) key dict item

\(\bullet\) lookup key dict item \(\Leftrightarrow\) (key, item) \(\in\) dict

delete \(\vdash\) \(\forall\) key dict\(\bullet\) delete key dict = \{key\} \(\triangleleft\) dict

block_delete \(\vdash\) \(\forall\) a dict\(\bullet\) block_delete a dict = dict \(\triangleright\) a

keys \(\vdash\) keys = Dom

ADDR_DEF \(\vdash\) \(\exists\) f \(\bullet\) TypeDefn (\(\lambda\) x \(\bullet\) Fst x = (\(\epsilon\) x \(\bullet\) T)) f

\(<-\) \(\vdash\) ConstSpec

\(\lambda\) $"<\sim"^n\)

\(\bullet\) \(\forall\) addr value st

\(\bullet\) addr \(\in\) Dom st

\(\Rightarrow\) $"<\sim"^n\) addr value st

\(=\) st \(\uplus\) \{(addr, value)\}

\(\sim<\)

fetch \(\vdash\) \(\forall\) addr st value

\(\bullet\) fetch addr st value \(\Leftrightarrow\) (addr, value) \(\in\) st

new \(\vdash\) \(\forall\) value st1 st2 addr

\(\bullet\) new value st1 (st2, addr)

\(\Leftrightarrow\) \(\neg\) addr \(\in\) Dom st1 \(\land\) st2 = st1 \(\uplus\) \{(addr, value)\}

initial_store \(\vdash\) initial_store = {}

THEORY_CONTENTS

\(\vdash\) \(\exists\) f \(\bullet\) TypeDefn (\(\lambda\) x \(\bullet\) T) f

MkTHEORY_CONTENTS

tc_name
tc_ty_env
tc_con_env
tc_parents
tc_axiom_dict
tc_definition_dict
∀ t x1 x2 x3 x4 x5 x6 x7 x8 x9 x10

● tc_name

(MkTHEORY CONTENTS
  x1
  x2
  x3
  x4
  x5
  x6
  x7
  x8
  x9
  x10)
= x1
∧ tc_by_env

(MkTHEORY CONTENTS
  x1
  x2
  x3
  x4
  x5
  x6
  x7
  x8
  x9
  x10)
= x2
∧ tc_con_env

(MkTHEORY CONTENTS
  x1
  x2
  x3
  x4
  x5
  x6
  x7
  x8
  x9
  x10)
= x3
∧ tc_parents

(MkTHEORY CONTENTS
  x1
  x2
  x3
  x4
  x5
  x6
  x7
  x8
  x9
  x10)
= x4
∧ tc_axiom_dict

(MkTHEORY CONTENTS

47
\[ x_1 = x_5 \]
\[ \land \text{te_definition_dict} \]
\[ (\text{MkTHEORY\_CONTENTS} \]
\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]
\[ x_5 \]
\[ x_6 \]
\[ x_7 \]
\[ x_8 \]
\[ x_9 \]
\[ x_{10} ) \]
\[ = x_6 \]
\[ \land \text{te_theorem_dict} \]
\[ (\text{MkTHEORY\_CONTENTS} \]
\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]
\[ x_5 \]
\[ x_6 \]
\[ x_7 \]
\[ x_8 \]
\[ x_9 \]
\[ x_{10} ) \]
\[ = x_7 \]
\[ \land \text{te_current_level} \]
\[ (\text{MkTHEORY\_CONTENTS} \]
\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]
\[ x_5 \]
\[ x_6 \]
\[ x_7 \]
\[ x_8 \]
\[ x_9 \]
\[ x_{10} ) \]
\[ = x_8 \]
\[ \land \text{te_deleted_levels} \]
\[ (\text{MkTHEORY\_CONTENTS} \]
\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]
\[ x_5 \]
\[ x_6 \]
\begin{align*}
x7 = x9 \\
x8 &= x10 \\
\text{\& te_user_data (MkTHEORY_CONTENTS)} \\
x1 &= x2 \\
x3 &= x4 \\
x5 &= x6 \\
x7 &= x8 \\
x9 &= x10 \\
\end{align*}

\[MkTHEORY_CONTENTS\]

\begin{align*}
& (\text{te_name t}) \\
& (\text{te_ty Env t}) \\
& (\text{te_con_env t}) \\
& (\text{te_parents t}) \\
& (\text{te_axiom_dict t}) \\
& (\text{te_definition_dict t}) \\
& (\text{te_theorem_dict t}) \\
& (\text{te_current_level t}) \\
& (\text{te_deleted_levels t}) \\
& (\text{te_user_data t}) \\
\end{align*}

\begin{align*}
\text{TSNormal} &= t \\
\text{TSLocked} \\
\text{TSAncestor} &\vdash \text{ConstSpec} \\
& (\lambda (\text{TSNormal'}, \text{TSLocked'}, \text{TSAncestor'}, \text{TSDeleted'})) \\
& \bullet [\text{TSNormal'}; \text{TSLocked'}; \text{TSAncestor'}; \text{TSDeleted'}] \\
& \in \text{Distinct} \\
& (\text{TSNormal}, \text{TSLocked}, \text{TSAncestor}, \text{TSDeleted}) \\
\text{THEADY_INFO} &\vdash \exists f \bullet \textbf{TypeDefn} (\lambda x: T) f \\
\text{MkTHEORY_INFO} \\
\text{ti_status} \\
\text{ti_inscope} \\
\text{ti_contents} &\vdash \forall t x1 x2 x3 \\
& \bullet \text{ti_status (MkTHEORY_INFO x1 x2 x3)} = x1 \\
& \& \text{ti_inscope (MkTHEORY_INFO x1 x2 x3)} \leftrightarrow x2 \\
& \& \text{ti_contents (MkTHEORY_INFO x1 x2 x3)} = x3 \\
& \& \text{MkTHEORY_INFO} \\
& (\text{ti_status t}) \\
& (\text{ti_inscope t}) \\
& (\text{ti_contents t}) \\
\text{PDS_THM} \\
\text{MkPDS_THM} &\vdash \exists f \bullet \textbf{TypeDefn} (\lambda x: T) f \\
\text{pt_theory} \\
\text{pt_level} \\
\text{pt_sequent} &\vdash \forall t x1 x2 x3 \\
& \bullet \text{pt_theory (MkPDS_THM x1 x2 x3)} = x1 \\
& \& \text{pt_level (MkPDS_THM x1 x2 x3)} = x2
\end{align*}
\[ pt_{\text{sequent}} (\text{MkPDS\_THM} x1 x2 x3) = x3 \]
\[ \text{MkPDS\_THM} \]
\[ (pt_{\text{theory}} t) \]
\[ (pt_{\text{level}} t) \]
\[ (pt_{\text{sequent}} t) \]
\[ = t \]

**PDS\_STATE** \( \vdash \exists f \cdot \text{TypeDefn} (\lambda x \cdot T) \ f \)

**MarkPDS\_STATE**

<table>
<thead>
<tr>
<th>ps_current_theory</th>
<th>ps_current_hierarchy</th>
<th>ps_theory_store</th>
<th>ps_hierarchy_store</th>
<th>ps_theorem_store</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vdash \forall t \ x1 \ x2 \ x3 \ x4 \ x5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \cdot \text{ps_current_theory\ (MkPDS_STATE} x1 \ x2 \ x3 \ x4 \ x5) = x1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sim \text{ps_current_hierarchy\ (MkPDS_STATE} x1 \ x2 \ x3 \ x4 \ x5) = x2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sim \text{ps_theory_store\ (MkPDS_STATE} x1 \ x2 \ x3 \ x4 \ x5) = x3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sim \text{ps_hierarchy_store\ (MkPDS_STATE} x1 \ x2 \ x3 \ x4 \ x5) = x4 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sim \text{ps_theorem_store\ (MkPDS_STATE} x1 \ x2 \ x3 \ x4 \ x5) = x5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**initial\_theory**

\( \vdash \forall ud \)

| \( \cdot \text{initial\_theory\ ud} = \) |
| \( \sim \text{(let contents) = MkTHEORY\_CONTENTS "MIN" initial\_dict initial\_dict [ initial\_dict initial\_dict initial\_dict 0 } \{ \} \) |
| \( \) ud |
| \( \) in \( \) let \( \) (st, addr) |
| \( \) = \( \) (\( \) \( \) st, addr) |
| \( \) \( \cdot \) new \( \) contents initial\_store \( \) (st, addr) |
| \( \) in \( \) (st, MkTHEORY\_INFO TSNormal T addr)) |

**initial\_state**

\( \vdash \forall ud \)

| \( \cdot \text{initial\_state\ ud} = \) |
| \( \sim \text{(let \( \) \( \) thy\_st, thy\_info\) = initial\_theory\ ud} \) |
| \( \) in \( \) let \( \) (hier\_st, hier\_addr) |
| \( \) = \( \) (\( \) \( \) \( \) st, addr) |
| \( \) \( \cdot \) new \( \) |
| \( \) [thy\_info] |
\[
\begin{aligned}
&\text{initial_store} \\
&(\text{st, addr)}) \\
&\text{in MkPDS\_STATE} \\
&(\text{ti\_contents thy\_info}) \\
&\text{hier\_addr} \\
&\text{thy\_st} \\
&\text{hier\_st} \\
&\text{initial\_store}) \\
\end{aligned}
\]

\text{theory\_contents}
\begin{align*}
\vdash & \forall \text{name thy\_c} \\
&\bullet \text{theory\_contents state name thy\_c} \\
&\Leftrightarrow (\text{let thy\_st = ps\_theory\_store state} \\
&\text{in let hier\_st = ps\_hierarchy\_store state} \\
&\text{in let cur\_hier = ps\_current\_hierarchy state} \\
&\text{in let infos} \\
&\quad = (\epsilon \text{\_fetch cur\_hier hier\_st x}) \\
&\text{in let thys} \\
&\quad = \text{Map} \\
&\quad (\lambda \text{addr} \\
&\quad \quad \bullet \epsilon \text{\_fetch addr thy\_st x}) \\
&\quad \text{a ti\_contents}) \\
&\text{in \exists thy} \\
&\bullet \text{thy \in Elms thys} \\
&\quad \wedge \text{tc\_name thy = name})
\end{align*}

\text{theory\_names}
\begin{align*}
\vdash & \forall \text{name} \\
&\bullet \text{name \in theory\_names state} \\
&\Leftrightarrow (\exists \text{thy\_c} \bullet \text{theory\_contents state name thy\_c})
\end{align*}

\text{theory\_ancestors}
\begin{align*}
\vdash & \forall \text{name} \\
&\bullet \text{theory\_ancestors state name} \\
&\quad = \bigcap \\
&\quad \{ \text{name \in theory\_names state \Rightarrow name \in P} \\
&\quad \wedge (\forall \text{anc1 thy\_c anc2} \\
&\quad \bullet \text{anc1 \in P} \\
&\quad \wedge \text{theory\_contents state anc1 thy\_c} \\
&\quad \wedge \text{anc2 \in Elms (tc\_parents thy\_c)} \\
&\quad \Rightarrow \text{anc2 \in P}) \}
\end{align*}

\text{interpret\_theory\_contents}
\begin{align*}
\vdash & \forall \text{thy\_cs} \\
&\bullet \text{interpret\_theory\_contents thy\_cs} \\
&\quad = (\text{abs\_theory} \\
&\quad (\{(\text{tyn, arity}) \\
&\quad \exists \text{thy\_c lev} \\
&\quad \bullet \text{thy\_c \in thy\_cs} \\
&\quad \wedge \text{lookup} \\
&\quad \text{tyn} \\
&\quad (\text{tc\_ty\_env thy\_c}) \\
&\quad (\text{arity, lev}))}, \\
&\quad \{(\text{cn, ty}) \\
&\quad \exists \text{thy\_c lev} \\
&\quad \bullet \text{thy\_c \in thy\_cs} \\
&\quad \wedge \text{lookup} \\
&\quad \text{cn} \\
&\quad (\text{tc\_con\_env thy\_c}) \\
&\quad (\text{ty, lev}))}, \\
&\quad \{\text{seq}
\end{align*}
∃thy_c thm lev
  • thy_c ∈ thy_cs
    ∧ (lookup
      thm
      (tc_axiom_dict thy_c)
      (seq, lev)
    ∨ lookup
      thm
      (tc_definition_dict thy_c)
      (seq, lev)))},

{seq
  ∃thy_c thm lev
  • thy_c ∈ thy_cs
    ∧ lookup
      thm
      (tc_definition_dict thy_c)
      (seq, lev)}},

{seq
  ∃thy_c thm lev
  • thy_c ∈ thy_cs
    ∧ lookup
      thm
      (tc_theorem_dict thy_c)
      (seq, lev)}

interpret_state
  ⊢ ∀ state
  • interpret_state state
    = mk_theory_hierarchy
      (λthm
        • interpret_theory_contents
          {tc
            ∃anc
            • anc ∈ theory_ancestors state thyn
              ∧ theory_contents state anc tc})

dest_state
  ⊢ ∀ state
  • dest_state state
    = (ps_current_theory state,
      ps_current_hierarchy state,
      ps_theory_store state,
      ps_hierarchy_store state,
      ps_theorem_store state)

dest_theory_contents
  ⊢ ∀ tc
  • dest_theory_contents tc
    = (tc_name tc, tc_ty_env tc, tc_con_env tc,
      tc_parents tc, tc_axiom_dict tc,
      tc_definition_dict tc, tc_theorem_dict tc,
      tc_current_level tc, tc_deleted_levels tc,
      tc_user_data tc)

current_theory_contents
  ⊢ ∀ state
  • current_theory_contents state
    = (let (cur_thy, _1, thy_st, _2, _3)
      = dest_state state
      in ❋ tc • fetch cur_thy thy_st tc)

current_theory_name
  ⊢ ∀ state
  • current_theory_name state
current_abstract_theory
\[ \forall \text{state} \]
  \[ \bullet \ current\_abstract\_theory\ state = \text{Fst} \]
  \[ \text{interpret theory contents} \{ \text{tc} \} \]
  \[ \exists \text{ancestors} \]
  \[ \bullet \text{anc} \in \text{theory ancestors} \]
  \[ \text{state} \]
  \[ \text{(current theory name state)} \]
  \[ \land \text{theory contents state anc tc} \}

theory_info
\[ \forall \text{state name} \]
  \[ \bullet \text{theory info state name} \]
  \[ = (\text{let} \ (\text{cur_thy, cur_hier, thy_st, hier_st, \_}) \]
  \[ = \text{dest state state} \]
  \[ \text{in let hier = (\varepsilon \text{let fetch cur_hier hier_st h})} \]
  \[ \text{in } \varepsilon \text{ ti} \]
  \[ \bullet \text{te_name} \]
  \[ (\varepsilon \text{ tc} \]
  \[ \bullet \text{fetch (ti.contents ti) thy_st tc} \]
  \[ = \text{name} \]
  \[ \land \neg \text{ti.status ti} = \text{TSDeleted} \}

current_theory_status
\[ \forall \text{state} \]
  \[ \bullet \text{current theory status state} \]
  \[ = \text{ti.status} \]
  \[ \text{(theory info state (current theory name state))} \]

check_thm
\[ \forall \text{state thm} \]
  \[ \bullet \text{check_thm state thm} \]
  \[ \Leftrightarrow (\text{let} \ (\text{cur_thy, cur_hier, thy_st, hier_st, \_}) \]
  \[ = \text{dest state state} \]
  \[ \text{in let tc} \]
  \[ = (\varepsilon \text{ tc} \bullet \text{fetch (pt.theory thm) thy_st tc}) \]
  \[ \text{in let ti = theory info state (te_name tc)} \]
  \[ \text{in pt.theory thm = ti.contents ti} \]
  \[ \land \text{ti.inscope ti} \]
  \[ \land \neg \text{ti.level thm} \in \text{te.deleted_levels tc} \}

check_thm_address
\[ \forall \text{state thm_ad} \]
  \[ \bullet \text{check_thm_address state thm_ad} \]
  \[ \Leftrightarrow (\text{let} \ (\_1, \_2, \_3, \_4, \text{thm_st}) \]
  \[ = \text{dest state state} \]
  \[ \text{in } \exists \text{ thm} \]
  \[ \bullet \text{fetch thm_ad thm_st thm} \]
  \[ \land \text{check_thm state thm) \}} \]

fetch_thms
\[ \forall \text{state thm_ads} \]
  \[ \bullet \text{fetch_thms state thm_ads} \]
  \[ = (\text{let} \ (\_1, \_2, \_3, \_4, \text{thm_st}) \]
  \[ = \text{dest state state} \]
  \[ \text{in Map} \]
  \[ (\lambda a \bullet \text{let fetch a thm_st thm}) \]
  \[ \text{thm_ads}) \]

hierarchy_ancestor
\[ \forall \text{state hier_ad1 hier_ad2} \]
  \[ \bullet \text{hierarchy_ancestor state hier_ad1 hier_ad2} \]
  \[ \Leftrightarrow (\text{let} \ (\_1, \text{cur_hier, \_2, hier_st, \_3}) \]

53
\(\text{pds\_mk\_thm} \vdash \forall \text{state seq} \)  
- \(\text{pds\_mk\_thm state seq} = (\text{let cur\_thy} = \text{ps\_current\_theory state} \)  
  \text{in let lev} = \text{tc\_current\_level} \)  
  \(\text{(current\_theory\_contents state)} \)  
  \text{in MkPDS\_THM cur\_thy lev seq)\)

\(\text{make\_current} \vdash \forall \text{thyn state} \)  
- \(\text{make\_current thyn state} = (\text{let (cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st)} \)  
  \text{in let f1 ti} = \text{tc\_name} \)  
  \(\epsilon \text{ tc} \)  
  \(\text{let f2 ti} = \text{MkTHEORY\_INFO} \)  
  \(\text{(ti\_status ti)} \)  
  \(\text{(f2 ti)} \)  
  \(\text{(ti\_contents ti)} \)  
  \text{in let hier'} = \text{Map} \)  
  \(f3 \)  
  \(\epsilon \text{ h}\)  
  \text{let hier\_st'} = (\text{cur\_hier <- hier'}\) hier\_st \text{in let cur\_thy'} \)  
  \text{= ti\_contents} \text{ in MkPDS\_STATE} \)  
  \(\text{cur\_thy'} \)  
  \text{cur\_hier} \text{ thy\_st} \text{ hier\_st' thm\_st})

\(\text{freeze\_hierarchy} \vdash \forall \text{state} \)  
- \(\text{freeze\_hierarchy state} = (\text{let (cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st)} \)  
  \text{in let f1 n} = (\text{if n} = \text{TSDeleted} \)  
  \text{then n} \text{ else TSAncestor} \)  
  \text{in let f2 ti} = \text{MkTHEORY\_INFO} \)  
  \(\text{(f1 (ti\_status ti))} \)  
  \(\text{(ti\_inscope ti)} \)
in let hier' = Map f2 (h • fetch cur_hier hier_st h)
in let hier_st' = (cur_hier <~ hier') hier_st
in MkPDS_STATE
  cur_thy
  cur_hier
  thy_st
  hier_st'
  thm_st)

new_hierarchy
⊢ ∀ state
• new_hierarchy state
  = (let (cur_thy, cur_hier, thy_st, hier_st, thm_st)
       = dest_state state
       in let hier = (h • fetch cur_hier hier_st h)
       in if
         ∃ ti
         • ti ∈ Elems hier
            ∧ ¬ ti_status ti
               ∈ {TSAncestor; TSDeleted}
         then state
        else
          (let (hier_st', cur_hier')
           = (st, a)
             • new hier hier_st (st, a))
          in MkPDS_STATE
            cur_thy
            cur_hier'
            thy_st
            hier_st'
            thm_st)

load_hierarchy
⊢ ∀ hier state
• load_hierarchy hier state
  = (let (cur_thy, cur_hier, thy_st, hier_st, thm_st)
       = dest_state state
       in if ¬ hierarchy_ancestor state cur_hier
          then state
          else
            (let cur_thyn = current_theory_name state
             in let st'
               = MkPDS_STATE
                 cur_thy
                 hier
                 thy_st
                 hier_st
                 thm_st
               in make_current cur_thyn st'))

open_theory
⊢ ∀ thyn state
• open_theory thyn state
  = (if
         ¬ thyn ∈ theory_names state
\[ \forall \text{ti\_status (theory\_info state thyn)} = \text{TSDeleted} \text{ then state} \]
\[ \text{else make\_current thyn state} \]

**empty\_theory** \(\vdash \forall \text{thyn pars ud} \)
- empty\_theory thyn pars ud
  \[ = \text{MkTHEORY\_CONTENTS} \]
  thyn
  initial\_dict
  initial\_dict
  pars
  initial\_dict
  initial\_dict
  initial\_dict
  0
  {}
  ud

**arbitrary\_ud** \(\vdash T\)

**delete\_theory** \(\vdash \forall \text{thyn state} \)
- delete\_theory thyn state
  \[ = (\text{if} \neg \text{thyn } \in \text{theory\_names state} \)
  \\[ \forall \neg \text{ti\_status (theory\_info state thyn)} = \text{TSNormal} \]
  \[ \forall \text{ti\_inscope (theory\_info state thyn)} \]
  \[ \forall (\exists \text{childname tc} \]
  \[ \bullet \text{theory\_contents state childname tc} \]
  \[ \land \text{thyn } \in \text{Elems (tc\_parents tc)} \)
  \[ \text{then state} \]
  \[ \text{else} \]
  \[ (\text{let} (\text{cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st}) \]
  \[ = \text{dest\_state state} \]
  \[ \text{in let ti } = \text{theory\_info state thyn} \]
  \[ \text{in let f ti} \]
  \[ = (\text{if} \text{ti}' = \text{ti} \]
  \[ \text{then} \]
  \[ \text{MkTHEORY\_INFO} \]
  \[ \text{TSDeleted} \]
  \[ f \]
  \[ (\text{ti\_contents ti}) \]
  \[ \text{else ti}) \]
  \[ \text{in let hier'}) \]
  \[ = \text{Map} \]
  \[ f \]
  \[ (e h \bullet \text{fetch cur\_hier hier\_st h}) \]
  \[ \text{in let hier\_st'} \]
  \[ = (\text{cur\_hier } \leftarrow \text{hier'}) \text{hier\_st} \]
  \[ \text{in let thy} \]
  \[ = \text{empty\_theory} \]
  \[ \text{thyn} \]
  \[ [] \]
  \[ \text{arbitrary\_ud} \]
  \[ \text{in let thy\_st'} \]
  \[ = (\text{ti\_contents ti } \leftarrow \text{thyn}) \]
  \[ \text{thy\_st} \]
  \[ \text{in MkPDS\_STATE} \]
\[ \textbf{new\_theory} \vdash \forall \text{thyn} \text{ ud state} \]
\[ \bullet \text{new\_theory thyn} \text{ ud state} \]
\[ = (\text{if} \text{ thyn} \in \text{theory\_names state} \]
\[ \text{then state} \]
\[ \text{else} \]
\[ (\text{let} \ (\text{cur\_thy}, \text{cur\_hier}, \text{thy\_st}, \text{hier\_st}, \text{thm\_st}) \]
\[ = \text{dest\_state state} \]
\[ \text{in let thy} \]
\[ = \text{empty\_theory thyn} \]
\[ \text{[current\_theory\_name state]} \]
\[ \text{ud} \]
\[ \text{in let (thy\_st', addr)} \]
\[ = (e (st, a)) \]
\[ \bullet \text{new thy thy\_st (st, a)} \]
\[ \text{in let ti} \]
\[ = \text{MkTHEORY\_INFO TSNormal F addr} \]
\[ \text{in let hier'} \]
\[ = \text{Cons} \]
\[ \text{ti} \]
\[ (e h) \]
\[ \bullet \text{fetch cur\_hier hier\_st h)} \]
\[ \text{in let hier\_st'} \]
\[ = (\text{cur\_hier} \text{<- hier'}) \text{ hier\_st} \]
\[ \text{in MkPDS\_STATE} \]
\[ \text{cur\_thy} \]
\[ \text{cur\_hier} \]
\[ \text{thy\_st'} \]
\[ \text{hier\_st'} \]
\[ \text{thm\_st}) \]

\[ \textbf{new\_parent} \vdash \forall \text{thyn state} \]
\[ \bullet \text{new\_parent thyn} \text{ state} \]
\[ = (\text{if} \]
\[ \neg \text{thyn} \in \text{theory\_names state} \]
\[ \lor \text{thyn} \]
\[ \in \text{Elems} \]
\[ (tc\_parents \]
\[ (\text{current\_theory\_contents state})) \]
\[ \lor (\exists \text{ancn} \]
\[ \bullet \text{ancn} \]
\[ \in \text{theory\_ancestors state thyn} \]
\[ \setminus \text{theory\_ancestors} \]
\[ \text{state} \]
\[ \text{[current\_theory\_name state]} \]
\[ \land (\text{let anc} \]
\[ = (e \text{ anc}) \]
\[ \bullet \text{theory\_contents state ancn anc}) \]
\[ \text{in let cur\_thy} \]
\[ = \text{current\_abstract\_theory state} \]
\[ \text{in (}\exists \text{ ty nlev} \]
\[ \bullet \text{ ty} \in \text{Dom (types cur\_thy) \]
\[ \land \text{lookup} \]

57
\[
\begin{align*}
    & ty \\
    & (tc_{ty}\_env\_anc) \\
    & nlev \\
    & \lor (\exists con tylev) \\
    & \bullet con \in \text{Dom (constants cur_thy)} \\
    & \land \text{lookup} \\
    & con \\
    & (tc_{con}\_env\_anc) \\
    & tylev))
\end{align*}
\]

then state

else

(\text{let (cur_thy, cur_hier, thy_st, hier_st, thm_st)}
\text{ = dest_state state}
\text{in let cur_thyn = current_theory_name state}
\text{in let f1 ti}
\text{ = tc}\_\text{name}
\text{\(\epsilon\) tc}
\text{\bullet fetch}
\text{(ti.contents ti)}
\text{thy_st}
\text{tc})

in let f2 ti
\text{\(\Leftrightarrow\) f1 ti}
\text{\(\in\) theory_ancestors state thyn}
\text{\lor ti\_inscope ti}
\text{in let f3 ti}
\text{ = MKTHEORY\_INFO}
\text{(ti.status ti)}
\text{(f2 ti)}
\text{(ti.contents ti)}
\text{in let hier'}
\text{ = Map}
\text{f3}
\text{\(\epsilon\) h}
\text{\bullet fetch}
\text{cur_hier}
\text{hier\_st}
\text{h}

in let tc
\text{ = current\_theory\_contents}
\text{state}
\text{in let (nm, t_e, e_e, pars,}
\text{ax_d, def_d, thm_d,}
\text{lev, x_levs, ud)}
\text{ = dest\_theory\_contents tc}
\text{in let tc'}
\text{ = MKTHEORY\_CONTENTS}
\text{nm}
\text{t_e}
\text{e_e}
\text{(Cons thyn pars)}
\text{ax_d}
\text{def_d}
\text{thm_d}
\text{lev}
\text{x_levs}
\text{ud}

58
in let hier$_{st}''$
  = (cur$_{hier} < -$ hier$'$)
hier$_{st}$
in let thy$_{st}''$
  = (cur$_{thy} < -$ tc$'$)
thy$_{st}$
in MkPDS$_{STATE}$
cur$_{thy}$
cur$_{hier}$
thy$_{st}''$
hier$_{st}''$
thm$_{st}$))

duplicate_theory

\[\vdash \forall \text{thyn copyn state} \]
\[
\bullet \text{duplicate_theory thyn copyn state} = (if \]
\[
\neg \text{thyn } \in \text{theory_names state} \\
\lor \text{copyn } \in \text{theory_names state} \\
\lor \text{thyn } = "\text{MIN}"
then state
else
(let (cur$_{thy}$, cur$_{hier}$, thy$_{st}$, hier$_{st}$, thm$_{st}$)
  = dest$_{state}$ state
in let tc
  = (e tc)
\bullet \text{theory_contents state thyn tc})
in let (nm, t.e, c.e, pars, ax$_{d}$, def$_{d}$, thm$_{d}$, lev, x.levs, ud)
  = dest$_{theory_contents}$ tc
in let tc$
  = \text{MkTHEORY\_CONTENTS}$
copyn
t.e
c.e
(Cons thyn pars)
ax$_{d}$
def$_{d}$
thm$_{d}$
lev
x.levs
ud
in let (thy$_{st}'$, addr)
  = (e (st, a))
\bullet \text{new tc' thy$_{st}$ (st, a))}
in let ti
  = MkTHEORY\_INFO
  TSNormal
  F
  addr
in let hier$'$
  = Cons
ti
  (e h)
\bullet \text{fetch}
cur$_{hier}$
hier$_{st}$
h$)
in let hier_st'
    = (cur_hier <-- hier')
  hier_st
in MkPDS_STATE
  cur_thy
  cur_hier
  thy_st'
  hier_st'
  thm_st))

lock_theory ⊢ ∀ thyn state
  • lock_theory thyn state
    = (if
      ¬ thyn ∈ theory_names state
      ∨ ¬ ti_status (theory_info state thyn)
      = TSNormal
    then state
    else
      (let (cur_thy, cur_hier, thy_st, hier_st,
            thm_st)
           = dest_state state
       in let ti = theory_info state thyn
       in let f ti'
           = (if ti' = ti
               then
                 MkTHEORY_INFO
                 TSLocked
                 (ti_inscope ti)
                 (ti_contents ti)
               else ti)
       in let hier'
           = Map
               f
               (e h• fetch cur_hier hier_st h)
       in let hier_st'
           = (cur_hier <-- hier') hier_st
       in MkPDS_STATE
       cur_thy
       cur_hier
       thy_st
       hier_st'
       thm_st))

unlock_theory ⊢ ∀ thyn state
  • unlock_theory thyn state
    = (if
      ¬ thyn ∈ theory_names state
      ∨ ¬ ti_status (theory_info state thyn)
      = TSLocked
    then state
    else
      (let (cur_thy, cur_hier, thy_st, hier_st,
            thm_st)
           = dest_state state
       in let ti = theory_info state thyn
       in let f ti'
           = (if ti' = ti
               then
                 MKTHEORY_INFO
               else
                 TI_NORMAL
               else ti)
       in let hier'
           = Map
               f
               (e h• fetch cur_hier hier_st h)
       in let hier_st'
           = (cur_hier <-- hier') hier_st
       in MkPDS_STATE
       cur_thy
       cur_hier
       thy_st
       hier_st'
       thm_st))
TSNormal
  (ti_inscope ti)
  (ti_contents ti)
else ti
in let hier'
  = Map
    f
    ((h • fetch cur_hier hier_st h)
    in let hier_st'
    = (cur_hier <- hier') hier_st
    in MkPDS_STATE
      cur_thy
      cur_hier
      thy_st
      hier_st'
      thm_st))
save_thm ⊢ ∀ key thm state
  • save_thm key thm state
    = (let tc = current_theory_contents state
       in if
         key ∈ keys (tc_theorem_dict tc)
         ∀ ¬ current_theory_status state = TSNormal
         ∀ ¬ check_thm state thm
       then state
       else
         (let (cur_thy, cur_hier, thy_st, hier_st, thm_st)
          = dest_state state
          in let level
          = (if pt_theory thm = cur_thy
             then pt_level thm
             else 0)
          in let thm_d'
          = enter
            key
            (pt_sequent thm, level)
            (tc_theorem_dict tc)
          in let (nm, t_e, c_e, pars, ax_d,
            def_d, thm_d, lev, x_levs, ud)
          = dest_theory_contents tc
          in let tc'
          = MkTHEORY_CONTENTS
            nm
            t_e
            c_e
            pars
            ax_d
            def_d
            thm_d'
            lev
            x_levs
            ud
          in let thy_st'
          = (cur_thy <- tc') thy_st
          in MkPDS_STATE
            cur_thy
            cur_hier
            thy_st'
is_latest_level

⊢ ∀ state lev
  • is_latest_level state lev
  ⇔ (let tc = current_theory_contents state
      in let (nm, t_e, c_e, pars, ax_d, def_d, thm_d, lev, x_levs, ud)
      = dest_theory_contents tc
      in let present
      = {lv
         | (∃ 1 key • lookup key t_e (_1, lv))
         ∨ (∃ 1 key
             • lookup key c_e (_1, lv))
         ∨ (∃ 1 key
             • lookup key ax_d (_1, lv))}
      in lev ∈ present
      ∧ (∀ lv • lv ∈ present ⇒ lv ≤ lev))

delete_extension

⊢ ∀ state
  • delete_extension state
  = (let tc = current_theory_contents state
      in if
        ¬ (∃ lev • is_latest_level state lev)
        ∨ (∃ childname tc
            • theory_contents state childname tc
            ∧ current_theory_name state
            ∈ Elems (tc_parents tc))
        ∨ ¬ current_theory_status state = TNormal
        then state
        else (let (cur_thy, cur_hier, thy_st, hier_st, thm_st)
        = dest_state state
        in let (nm, t_e, c_e, pars, ax_d, def_d, thm_d, lev, x_levs, ud)
        = dest_theory_contents tc
        in let lev
        = (e lv • is_latest_level state lv)
        in let def_d'
        = block_delete
        {(1, lv)lv = def_d}
        def_d
        in let ax_d'
        = block_delete
        {(1, lv)lv = ax_d}
        ax_d
        in let t_e'
        = block_delete
        {(1, lv)lv = t_e}
        t_e
        in let c_e'
        = block_delete
        {(1, lv)lv = c_e}
        c_e
        in let lev' = lev + 1
        in let tc'
        = MkTHEORY_CONTENTS
\texttt{delete_thm} \quad \vdash \forall \text{key state} \\
\textbullet \texttt{delete_thm key state} \\
\quad = (\text{let } tc = \text{current theory contents state} \\
\quad \text{in if} \\
\quad \quad \neg \text{key } \in \text{keys (tc theorem dict tc)} \\
\quad \quad \vee \neg \text{current theory status state} = \text{TSNormal} \\
\quad \text{then state} \\
\quad \quad \text{else} \\
\quad \quad \text{(let (cur_thy, cur_hier, thy_st, hier_st, thm_st) } \\
\quad \quad \quad \text{= dest state state} \\
\quad \quad \text{in let (nm, t.e, c.e, pars, ax.d, def.d, } \\
\quad \quad \quad \text{thm.d, lev, x.levs, ud) } \\
\quad \quad \quad \text{= dest theory contents tc} \\
\quad \quad \text{in let thm.d'} = \text{delete key thm_d} \\
\quad \quad \text{in let tc' } \\
\quad \quad \quad \text{= MkTHEORY CONTENTS} \\
\quad \quad \quad \text{nm} \\
\quad \quad \quad \text{t.e} \\
\quad \quad \quad \text{c.e} \\
\quad \quad \quad \text{pars} \\
\quad \quad \quad \text{ax.d} \\
\quad \quad \quad \text{def.d} \\
\quad \quad \quad \text{thm.d'} \\
\quad \quad \quad \text{lev} \\
\quad \quad \quad \text{x.levs} \\
\quad \quad \quad \text{ud} \\
\quad \quad \text{in let thy_st' } \\
\quad \quad \quad \text{= (cur_thy } \prec \text{ tc') thy_st} \\
\quad \quad \text{in MkPDS STATE} \\
\quad \quad \quad \text{cur_thy} \\
\quad \quad \quad \text{cur_hier} \\
\quad \quad \quad \text{thy_st'} \\
\quad \quad \quad \text{hier_st} \\
\quad \quad \quad \text{thm_st}) \\
\texttt{pds_new_axiom} \\
\quad \vdash \forall \text{tm key state} \\
\textbullet \texttt{pds_new_axiom tm key state} \\
\quad = (\text{let tc = current theory contents state} \\
\quad \text{in let thy_st'= (let thy_st' = (cur_thy } \prec \text{ tc') thy_st} \\
\quad \text{in MkPDS STATE} \\
\quad \quad \text{cur_thy} \\
\quad \quad \text{cur_hier} \\
\quad \quad \text{thy_st'} \\
\quad \quad \text{hier_st} \\
\quad \quad \text{thm_st}) \\
\text{63}
in let seq = (\}, tm)
in if
key \in keys (tc_axiom_dict tc)
\lor \neg seq
\in sequents
(current_abstract_theory state)
\lor \neg current_theory_status state
= TSNormal
then state
else
(let (cur_thy, cur_hier, thy_st, hier_st, thm_st)
  = dest_state state
in let (nm, t_e, c_e, pars, ax_d, def_d, thm_d, lev, x_levs, ud)
  = dest_theory_contents tc
in let lev' = lev + 1
in let ax_d' = enter key (seq, lev') ax_d
in let tc'
  = MkTHEORY_CONTENTS
    nm
    t_e
    c_e
    pars
    ax_d'
    def_d
    thm_d
    lev'
    x_levs
    ud
in let thy_st'
  = (cur_thy <- tc') thy_st
in let (thm_st', _1)
  = (\ sa
    \ new
    (pds_mk_thm
      state
      seq)
    thm_st
    sa)
in MkPDS_STATE
  cur_thy
  cur_hier
  thy_st'
  hier_st
  thm_st'))

make_definition
\forall definer pars thm_ads state
  \bullet make_definition definer ((pars, thm_ads), state)
  = (if
    \exists thm_ad
    \bullet thm_ad \in Elems thm_ads
    \land \neg check_thm_address state thm_ad
    then state
  else
    (let thms = fetch_thms state thm_ads
     in if \neg ((pars, thms), state) \in Dom definer

64
then state
else
  (let (seq, tys, cons, ks)
   = definer ® ((pars, thms), state)
   in let (cur_thy, cur_hier, thy_st,
   hier_st, thm_st)
   = dest_state state
   in let tc
   = current_theory_contents state
   in let (nm, t_e, c_e, pars, ax_d,
   def_d, thm_d, lev, x_levs,
   ud)
   = dest_theory_contents tc
   in let lev' = lev + 1
   in let def_d'
   = Fold
   (λ k
   ⋅ enter
   k
   (seq, lev'))
   def_d
   in let t_e'
   = Fold
   (Uncurry enter)
   (Map
   (λ (s, x)
   ⋅ (s, x, lev')))
   tys)
   t_e
   in let c_e'
   = Fold
   (Uncurry enter)
   (Map
   (λ (s, x)
   ⋅ (s, x,
   lev')))
   cons)
   c_e
   in let tc'
   = MkTHEORY_CONTENTS
   nm
   t_e'
   c_e'
   pars
   ax_d
   def_d'
   thm_d
   lev'
   x_levs
   ud
   in let thy_st'
   = (cur_thy
   ⊲ tc')
   thy_st
   in let (thm_st', _1)
   = (ε sa
   ⋅ new

65
\[(\text{pds\_mk\_thm \ state \ seq}) \\]
\[\text{thm\_st} \]
\[\text{st}\]

\[\text{in \ MkPDS\_STATE} \]
\[\text{cur\_thy} \]
\[\text{cur\_hier} \]
\[\text{thy\_st'} \]
\[\text{hier\_st} \]
\[\text{thm\_st'})])\]

\textbf{pds\_new\_type} \vdash \forall \ ty \ \text{arity \ state} \\
\bullet \ pds\_new\_type \ ty \ \text{arity \ state} \\
\[= \text{(if \ ty} \]
\[\in \text{Dom} \]
\[\text{(types \ (current\_abstract\_theory \ state))} \]
\[\text{then \ state} \]
\[\text{else} \]
\[\text{(let \ (cur\_thy, \ cur\_hier, \ thy\_st, \ hier\_st,} \]
\[\text{thm\_st) = dest\_state \ state} \]
\[\text{in \ let \ tc = current\_theory\_contents \ state} \]
\[\text{in \ let \ (nm, \ t\_e, \ c\_e, \ pars, \ ax\_d, \ def\_d,} \]
\[\text{thm\_d, \ lev, \ x\_levs, \ ud)} \]
\[\text{= dest\_theory\_contents \ tc} \]
\[\text{in \ let \ lev' = lev + 1} \]
\[\text{in \ let \ t\_e' = enter \ ty \ (arity, \ lev') \ t\_e} \]
\[\text{in \ let \ tc' = MkTHEORY\_CONTENTS} \]
\[\text{nm} \]
\[\text{t\_e'} \]
\[\text{c\_e} \]
\[\text{pars} \]
\[\text{ax\_d} \]
\[\text{def\_d} \]
\[\text{thm\_d} \]
\[\text{lev'} \]
\[\text{x\_levs} \]
\[\text{ud} \]
\[\text{in \ let \ thy\_st'} \]
\[\text{= (cur\_thy \lhd \ lh hy\_st} \]
\[\text{in \ MkPDS\_STATE} \]
\[\text{cur\_thy} \]
\[\text{cur\_hier} \]
\[\text{thy\_st'} \]
\[\text{hier\_st} \]
\[\text{thm\_st'})\)

\textbf{pds\_new\_constant} \\
\vdash \forall \ con \ ty \ \text{state} \\
\bullet \ pds\_new\_constant \ con \ ty \ \text{state} \\
\[= \text{(if \ con} \]
\[\in \text{Dom} \]
\[\text{(constants)} \]
\[\text{(current\_abstract\_theory \ state))} \]
\[\forall \ - \ ty \]
\[\in \text{wf\_type} \]
\[\text{(types}} \]

66
\[
\text{(current\_abstract\_theory \text{state})}
\]

\text{then state}
\text{else}
(\text{let (cur\_thy, cur\_hier, thy\_st, hier\_st, thm\_st)}
\quad = \text{dest\_state state}
\text{in let tc = current\_theory\_contents state}
\text{in let (nm, t\_e, c\_e, pars, ax\_d, def\_d, thm\_d, lev, x\_levs, ud)}
\quad = \text{dest\_theory\_contents tc}
\text{in let lev' = lev + 1}
\text{in let c\_e'}
\quad = \text{enter con (ty, lev') c\_e}
\text{in let tc'}
\quad = \text{MkTHEORY\_CONTENTS}
\quad \text{nm}
\quad \text{t\_e}
\quad c\_e'
\quad \text{pars}
\quad \text{ax\_d}
\quad \text{def\_d}
\quad \text{thm\_d}
\quad \text{lev'}
\quad \text{x\_levs}
\quad \text{ud}
\text{in let thy\_st'}
\quad = (\text{cur\_thy} <\text{ tc'}) \text{thy\_st}
\text{in MkPDS\_STATE}
\quad \text{cur\_thy}
\quad \text{cur\_hier}
\quad \text{thy\_st'}
\quad \text{hier\_st}
\quad \text{thm\_st})
\]

\text{make\_inference}
\[
\vdash \forall \text{infer\_pars \text{thm\_ads state}
\quad \bullet \text{make\_inference \text{infer\_err (pars, thm\_ads), state}}
\quad = (\text{if thm\_ad}
\quad \exists \text{ thm\_ad} 
\quad \bullet \text{thm\_ad} \in \text{Elems thm\_ads}
\quad \wedge \neg \text{check\_thm\_address state thm\_ad}
\quad \text{then state}
\quad \text{else}
\quad (\text{let thms = fetch\_thms state thm\_ads}
\quad \text{in if}
\quad \neg (\text{pars, thms), state} \in \text{Dom infer\_err}
\quad \vee \neg \text{current\_theory\_status state}
\quad = \text{TSNormal}
\quad \text{then state}
\quad \text{else}
\quad (\text{let seq}
\quad = \text{infer\_err} \oplus (\text{pars, thms), state}
\quad \text{in let (cur\_thy, cur\_hier, thy\_st,}
\quad \text{hier\_st, thm\_st})
\quad = \text{dest\_state state}
\quad \text{in let (thm\_st', l)}
\quad = (\epsilon \text{ sa}
\quad \bullet \text{new}
\quad (\text{pds\_mk\_thm state seq})
\quad 67
\]
\[
\text{thm_st} \quad \text{sa})
\]

in MkPDS\_STATE

\text{cur\_thy}
\text{cur\_hier}
\text{thy\_st}
\text{hier\_st}
\text{thm\_st'})})\}

\text{allowed} \quad \vdash \forall \text{definer inferrer trans}
\text{allowed definer inferrer trans}
\leftrightarrow \text{trans} = \text{freeze\_hierarchy}
\lor \text{trans} = \text{new\_hierarchy}
\lor (\exists \text{addr})
\bullet \text{trans} = \text{load\_hierarchy addr}
\lor (\exists \text{thyn})
\bullet \text{trans} = \text{open\_theory thyn}
\lor (\exists \text{thyn})
\bullet \text{trans} = \text{delete\_theory thyn}
\lor (\exists \text{thyn ad})
\bullet \text{trans} = \text{new\_theory thyn ad}
\lor (\exists \text{thyn})
\bullet \text{trans} = \text{new\_parent thyn}
\lor (\exists \text{thyn copyn})
\bullet \text{trans} = \text{duplicate\_theory thyn copyn}
\lor (\exists \text{thyn})
\bullet \text{trans} = \text{lock\_theory thyn}
\lor (\exists \text{thyn})
\bullet \text{trans} = \text{unlock\_theory thyn}

\text{thyn}
\lor (\exists \text{key thm})
\bullet \text{trans} = \text{save\_thm}

\text{key}
\text{thm}
\lor \text{trans} = \text{delete\_extension}
\lor (\exists \text{key})
\bullet \text{trans} = \text{delete\_thm key}
\lor (\exists \text{tm key})
\bullet \text{trans} = \text{pds\_new\_axiom tm key}
\lor (\exists \text{ty arity})
\bullet \text{trans} = \text{pds\_new\_type ty arity}
\lor (\exists \text{con ty})
\bullet \text{trans} = \text{pds\_new\_constant con ty}
\lor (\exists \text{pars thm\_ads})
\bullet \text{trans} = \text{Curry}
\quad (\text{make\_definition definer})
\quad (\text{pars, thm\_ads})
\lor (\exists \text{pars thm\_ads})
\bullet \text{trans} = \text{Curry}
\quad (\text{make\_inference inferrer})

68
\[
(pars, thm\_ads)(\text{definer inferer interpreter})
\]
\[\vdash \forall \text{definer inferer interpreter}
\]
\[\bullet \ pds \text{ definer inferer interpreter}
= (\lambda (pars, thm\_ads), state)
\]
\[\bullet \ (\text{let state'}
= \text{interpreter}
\text{definer}
\text{inferer}
(pars, thm\_ads)
state
\text{in (state', ps\_theorem\_store state'))})
\]
\[\text{interpret}\_\text{state}
\]
\[\vdash \forall \text{definer}
\]
\[\bullet \ \text{good}\_\text{definer definer}
\equiv (\forall \text{pars thms state}
\bullet \ ((\text{pars, thms}), \text{state}) \in \text{Dom definer}
\Rightarrow (\text{let thy} = \text{current\_abstract\_theory state}
\text{in let (tyenv, conenv, axs)}
= \text{rep\_theory thy}
\text{in let (seq, tys, cons, _1)}
= \text{definer @ ((pars, thms), state)}
\text{in let tyenv'
= Fold}
(\lambda \text{tn te} \cdot \text{te} \cup \{\text{tn}\})
tys
tyenv
\text{in let conenv'
= Fold}
(\lambda \text{st ce} \cdot \text{ce} \cup \{\text{st}\})
cons
conenv
\text{in let axs' = axs} \cup \{\text{seq}\}
\text{in let thy'}
= \text{abs\_theory}
(\text{tyenv', conenv', axs'})
\text{in thy}
\in \text{definitional\_extension}
\text{thy'})
\]
\[\vdash \forall \text{v inferer}
\]
\[\bullet \ \text{good}\_\text{inferer v inferer}
\equiv (\forall \text{pars thms state}
\bullet \ ((\text{pars, thms}), \text{state}) \in \text{Dom inferer}
\Rightarrow (\text{let thy} = \text{current\_abstract\_theory state}
\text{in let seq}
= \text{inferer @ ((pars, thms), state)}
\text{in seq} \in \text{sequents thy}
\wedge \text{seq} \in \text{valid v thy)})
\]
\[\vdash \forall \text{v inferer}
\]
\[\bullet \ \text{good}\_\text{interpreter interpreter}
\equiv (\forall \text{definer inferer pars thm\_ads}
\bullet \ \text{allowed}
\text{definer}
\text{inferer}
(\text{interpreter}
\text{definer})
\]
69
\textit{inferrer}

\((\text{pars, thm\_ads}))\)
11 INDEX

< - ............................................ 9 dest_state ............................... 43
< - ............................................ 41 dest_state ............................... 52
< - ............................................ 46 dest_theory_contents ................ 17
ADDR_DEF .................................... 9 dest_theory_contents ............... 43
ADDR_DEF .................................... 46 dest_theory_contents ............... 52
ADDR ........................................... 9 DICT ....................................... 8
ADDR ......................................... 45 DICT ............................. 45
allowed ........................................ 37 duplicate_theory ..................... 27
allowed ......................................... 44 duplicate_theory ..................... 44
allowed ......................................... 68 duplicate_theory ................. 59
arbitrary_ud .................................. 23 empty_theory ......................... 23
arbitrary_ud .................................. 43 empty_theory ......................... 43
arbitrary_ud .................................. 56 empty_theory ......................... 56
block_delete .................................. 8 enter ...................................... 8
block_delete .................................. 41 enter ...................................... 41
block_delete .................................. 46 enter ...................................... 46
check_thm_address ............................ 19 fetch_thms .............................. 19
check_thm_address ............................ 43 fetch_thms .............................. 43
check_thm_address ............................ 53 fetch_thms .............................. 53
check_thm ...................................... 19 fetch ...................................... 9
check_thm ...................................... 43 fetch ...................................... 41
check_thm ...................................... 53 fetch ...................................... 46
current_abstract_theory ...................... 18 freeze_hierarchy ....................... 21
current_abstract_theory ...................... 43 freeze_hierarchy ....................... 43
current_abstract_theory ...................... 53 freeze_hierarchy ....................... 54
current_theory_contents ...................... 18 good_definer ............................ 39
current_theory_contents ...................... 43 good_definer ............................ 45
current_theory_contents ...................... 52 good_definer ............................ 69
current_theory_name ........................... 18 good_inerrer ............................ 39
current_theory_name ........................... 43 good_inerrer ............................ 45
current_theory_name ........................... 52 good_inerrer ............................ 69
current_theory_status ......................... 18 good_interpreter ....................... 40
current_theory_status ......................... 43 good_interpreter ....................... 45
current_theory_status ......................... 53 good_interpreter ....................... 69
DEFINER ........................................ 45 hierarchy_ancestor ................. 20
delete_extension ................................ 30 hierarchy_ancestor ................. 43
delete_extension ................................ 44 hierarchy_ancestor ................. 53
delete_extension ................................ 62 HIERARCHY ............................. 12
delete_theory .................................. 24 HIERARCHY ............................. 45
delete_theory .................................. 43 INFERRER ................................ 46
delete_theory .................................. 56 initial_dict ............................. 8
delete_thm ...................................... 31 initial_dict ............................. 41
delete_thm ...................................... 44 initial_dict ............................. 46
delete_thm ...................................... 63 initial_state ......................... 13
delete ............................................ 8 initial_state ......................... 42
delete ............................................ 41 initial_state ......................... 50
delete ............................................ 46 initial_store ......................... 10
dest_state ...................................... 17 initial_store ......................... 41