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.

Reference: DS/FMU/IED/SPC005; issue 2.7

1 DOCUMENT CONTROL

1.1 Contents list

1.2 Document cross references

[1] Michael J.C. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF. Lecture Notes in Computer Science. Vol. 78.* Springer-Verlag, 1979.

[2] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* MIT Press, 1990.

[3] DS/FMU/IED/DEF004. *ProofPower Product Requirement Specification.* R.D. Arthan, Lemma 1 Ltd., `http://www.lemma-one.com`.

[4] DS/FMU/IED/HLD007. *HOL PDS Logical Kernel High Level Design.* R.D. Arthan and K. Blackburn, Lemma 1 Ltd., `http://www.lemma-one.com`.

[5] DS/FMU/IED/SPC001. *HOL Formalised: Language and Overview.* R.D. Arthan, Lemma 1 Ltd., `http://www.lemma-one.com`.

[6] DS/FMU/IED/SPC002. *HOL Formalised: Semantics.* R.D. Arthan, Lemma 1 Ltd., `http://www.lemma-one.com`.

[7] DS/FMU/IED/SPC003. *HOL Formalised: Deductive System.* R.D. Arthan, Lemma 1 Ltd., `http://www.lemma-one.com`.

[8] DS/FMU/IED/SPC004. *HOL Formalised: Proof Development System.* R.D. Arthan, Lemma 1 Ltd., `http://www.lemma-one.com`.

[9] *The HOL System: Description.* SRI International, 4 December 1989.

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 constructosr, 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 axiomatic. Thus proofs (i.e. computations of theorems) are carried out with respect to a context determined by a callection 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 *DEFINER*s and *INTERPRETER*s 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

```
declare_type_abbrev("DICT", ["'X"], ⌜:STRING ↔ 'X⌝);
```

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

HOL Constant

$$\mathbf{initial\_dict} : ('X)DICT$$

$$initial\_dict = \{\}$$

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

HOL Constant

$$\mathbf{enter} : STRING \to 'X \to ('X)DICT \to ('X)DICT$$

$$\forall key\ item\ dict \bullet enter\ key\ item\ dict = dict \oplus \{(key, item)\}$$

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.

HOL Constant

$$\mathbf{lookup} : STRING \to ('X)DICT \to 'X \to BOOL$$

$$\forall key\ dict\ item \bullet \quad lookup\ key\ dict\ item \Leftrightarrow (key, item) \in dict$$

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

HOL Constant

$$\mathbf{delete} : STRING \to ('X)DICT \to ('X)DICT$$

$$\forall key\ dict \bullet\ delete\ key\ dict = \{key\} \lhd dict$$

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

HOL Constant

$$\mathbf{block\_delete} : ('X\ SET) \to ('X)DICT \to ('X)DICT$$

$$\forall a\ dict \bullet\ block\_delete\ a\ dict = dict \rhd a$$

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

HOL Constant

$$\mathbf{keys} : ('X)DICT \to STRING\ SET$$

$$keys = Dom$$

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 $((\epsilon x :' X \bullet T), n)$ 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(([], ⌜∃a:'X × ℕ•(λx•Fst x = εx:'X•T) a⌝),
                ∃_tac⌜((εx:'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 defind to create new storage locations, but only to modify existing ones.

SML

```
declare_infix(300, "<-");
```

HOL Constant

$$\$<- : ('X)ADDR \to 'X \to ('X)STORE \to ('X)STORE$$

$$\forall\ addr\ value\ st \bullet$$
$$\quad addr \in Dom\ st \Rightarrow$$
$$\quad (addr <- value)\ st = st \oplus \{(addr,\ value)\}$$

fetch is the dereferencing operation:

HOL Constant

$$\mathbf{fetch} : ('X)ADDR \to ('X)STORE \to 'X \to BOOL$$

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

new is the allocation operation:

HOL Constant

$$\mathbf{new} : 'X \to ('X)STORE \to (('X)STORE \times ('X)ADDR) \to BOOL$$

$$\forall\ value\ st1\ st2\ addr \bullet$$
$$\quad new\ value\ st1\ (st2,\ addr) \Leftrightarrow$$
$$\qquad \neg addr \in Dom\ st1$$
$$\quad \wedge \quad st2 = st1 \oplus \{(addr,\ value)\}$$

Stores are constructed using *new* from an initial empty store:

HOL Constant

$$\textbf{initial\_store} : ('X)STORE$$

$$initial\_store = \{\}$$

6 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 $'UD$.

6.2 System Inputs

We will systematically use the type variable $'IP$ 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 $'IP$ 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).

HOL Labelled Product

THEORY_CONTENTS

$$\begin{array}{ll}
tc_name & : STRING; \\
tc_ty_env & : (\mathbb{N} \times \mathbb{N})\ DICT; \\
tc_con_env & : (TYPE \times \mathbb{N})\ DICT; \\
tc_parents & : STRING\ LIST; \\
tc_axiom_dict & : (SEQ \times \mathbb{N})\ DICT; \\
tc_definition_dict & : (SEQ \times \mathbb{N})\ DICT; \\
tc_theorem_dict & : (SEQ \times \mathbb{N})\ DICT; \\
tc_current_level & : \mathbb{N}; \\
tc_deleted_levels & : \mathbb{N}\ SET; \\
tc_user_data & : 'UD
\end{array}$$

Here the fields have the following significance:

Field	Description
name	This gives the name of the theory.
ty_env	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.
con_env	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.
parents	This is the set of names of parents of this theory.
axiom_dict	This contains the non-definitional axioms of the theory. Each axiom is marked with the level number which was current when the axiom was introduced.
definition_dict	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.
theorem_dict	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).
user_data	This contains the user-defined data stored in the theory
current_level	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.
deleted_levels	This is the set of level numbers corresponding to extensions which have been deleted.

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;
TSDeleted   : STATUS
```

$[TSNormal;\ TSLocked;\ TSAncestor;\ TSDeleted] \in Distinct$

The significance of the theory status values is as follows:

Value	Description
TSNormal	A theory which can be modified while this theory hierarchy is current;
TSLocked	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);
TSAncestor	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);
TSDeleted	A theory which has been deleted.

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

HOL Labelled Product

THEORY_INFO

$$
\begin{array}{ll}
ti\_status & : STATUS;\\
ti\_inscope & : BOOL;\\
ti\_contents & : (('UD)\,THEORY\_CONTENTS)\,ADDR
\end{array}
$$

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

A theory hierarchy will comprise a list of *THEORY_INFO*s:

SML

```
declare_type_abbrev("HIERARCHY", ["'UD"], ⌜:(('UD)THEORY_INFO)LIST⌝);
```

6.5 Concrete Theorems

A theorem is represented by the following data type:

HOL Labelled Product

PDS_THM

$$
\begin{array}{ll}
pt\_theory & : (('UD)\,THEORY\_CONTENTS)\,ADDR;\\
pt\_level & : \mathbb{N};\\
pt\_sequent & : SEQ
\end{array}
$$

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:

HOL Labelled Product

PDS_STATE

$$
\begin{array}{ll}
ps\_current\_theory & : (('UD)THEORY\_CONTENTS)ADDR; \\
ps\_current\_hierarchy & : (('UD)HIERARCHY)ADDR; \\
ps\_theory\_store & : (('UD)THEORY\_CONTENTS)STORE; \\
ps\_hierarchy\_store & : (('UD)HIERARCHY)\ STORE; \\
ps\_theorem\_store & : (('UD)PDS\_THM)\ STORE
\end{array}
$$

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 *MIN* not *INIT*) The initial theory information comes supplied with a store containing the contents of a suitable initial theory:

HOL Constant

$$
\begin{array}{lll}
\mathbf{initial\_theory} & : 'UD \to & \\
 & (& (('UD)THEORY\_CONTENTS)STORE) \\
 & \times & ('UD)THEORY\_INFO
\end{array}
$$

$$
\begin{array}{l}
\forall ud \bullet initial\_theory\ ud = \\
\quad let\ contents = MkTHEORY\_CONTENTS \\
\qquad "MIN" \\
\qquad initial\_dict \quad initial\_dict \\
\qquad [] \\
\qquad initial\_dict \quad initial\_dict \quad initial\_dict \\
\qquad 0 \quad \{\} \\
\qquad ud \\
\quad in\ let\ (st,\ addr) = \epsilon(st,\ addr) \bullet new\ contents\ initial\_store\ (st,\ addr) \\
\quad in\ (st,\ MkTHEORY\_INFO\ \ TSNormal\ \ T\ \ addr)
\end{array}
$$

The initial state is then as follows:

HOL Constant

$$
\mathbf{initial\_state}\ : 'UD \to ('UD)PDS\_STATE
$$

$$
\begin{array}{l}
\forall ud \bullet initial\_state\ ud = \\
let\ (thy\_st,\ thy\_info) = initial\_theory\ ud \\
in\ let\ (hier\_st,\ hier\_addr) = \epsilon(st,\ addr) \bullet new\ [thy\_info]\ initial\_store\ (st,\ addr) \\
in\ MkPDS\_STATE\ (ti\_contents\ thy\_info)\ hier\_addr\ thy\_st\ hier\_st\ initial\_store
\end{array}
$$

6.6.2 Interpretation Mapping

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

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.

HOL Constant

$$\textbf{theory\_contents} : ('UD)PDS\_STATE \to STRING \to ('UD)THEORY\_CONTENTS \to BOOL$$

$$\begin{array}{l}
\forall state\ name\ thy_c \bullet theory_contents\ state\ name\ thy_c \Leftrightarrow \\
\quad let\ thy_st = ps_theory_store\ state \\
\quad in\ let\ hier_st = ps_hierarchy_store\ state \\
\quad in\ let\ cur_hier = ps_current_hierarchy\ state \\
\quad in\ let\ infos = \epsilon x \bullet fetch\ cur_hier\ hier_st\ x \\
\quad in\ let\ thys = Map((\lambda addr \bullet \epsilon x \bullet fetch\ addr\ thy_st\ x)\ o\ ti_contents)\ infos \\
\quad in\ \exists thy \bullet\ thy \in Elems\ thys \wedge tc_name\ thy = name
\end{array}$$

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

HOL Constant

$$\textbf{theory\_names} : ('UD)PDS\_STATE \to STRING\ SET$$

$$\begin{array}{l}
\forall state\ name \bullet name \in theory_names\ state \Leftrightarrow \\
\quad \exists thy_c \bullet theory_contents\ state\ name\ thy_c
\end{array}$$

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):

HOL Constant

$$\textbf{theory\_ancestors} : ('UD)PDS\_STATE \to STRING \to STRING\ \ SET$$

$$\begin{array}{lll}
\multicolumn{3}{l}{\forall state\ name \bullet theory_ancestors\ state\ name = \bigcap\{P{:}STRING\ SET\ |} \\
 & \multicolumn{2}{l}{(name \in theory_names\ state \Rightarrow name \in P)} \\
\wedge & \multicolumn{2}{l}{(\forall anc1\ thy_c\ anc2 \bullet} \\
 & & anc1 \in P \\
 & \wedge & theory_contents\ state\ anc1\ thy_c \\
 & \wedge & anc2 \in Elems\ (tc_parents\ thy_c) \\
 & \Rightarrow & anc2 \in P)\}
\end{array}$$

Given a set of theory contents, *interpret_theory_contents* constructs a *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].

HOL Constant

$$\textbf{interpret\_theory\_contents} : (('UD)THEORY\_CONTENTS\ \ SET) \rightarrow (THEORY \times (SEQ\ SET) \times (SEQ\ SET))$$

$$\begin{aligned}
&\forall thy_cs \bullet interpret_theory_contents\ thy_cs = (\\
&\quad abs_theory(\\
&\quad \{(tyn,\ arity) \mid \exists thy_c\ lev \bullet thy_c \in thy_cs\\
&\qquad\qquad \wedge\ lookup\ tyn\ (tc_ty_env\ thy_c)\ (arity,\ lev)\},\\
&\quad \{(cn,\ ty) \mid \exists thy_c\ lev \bullet thy_c \in thy_cs\\
&\qquad\qquad \wedge\ lookup\ cn\ (tc_con_env\ thy_c)\ (ty,\ lev)\},\\
&\quad \{seq \mid \exists thy_c\ thmn\ lev \bullet thy_c \in thy_cs\ \wedge\\
&\qquad\qquad (lookup\ thmn\ (tc_axiom_dict\ thy_c)\ (seq,\ lev)\\
&\qquad \vee \qquad lookup\ thmn\ (tc_definition_dict\ thy_c)\ (seq,\ lev))\}\\
&\quad),\\
&\quad \{seq \mid \exists thy_c\ thmn\ lev \bullet thy_c \in thy_cs\ \wedge\\
&\qquad lookup\ thmn\ (tc_definition_dict\ thy_c)\ (seq,\ lev)\},\\
&\quad \{seq \mid \exists thy_c\ thmn\ lev \bullet thy_c \in thy_cs\ \wedge\\
&\qquad lookup\ thmn\ (tc_theorem_dict\ thy_c)\ (seq,\ lev)\}\\
&)
\end{aligned}$$

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).

HOL Constant

$$\textbf{interpret\_state} : ('UD)PDS\_STATE \rightarrow THEORY\_HIERARCHY$$

$$\begin{aligned}
&\forall state \bullet interpret_state\ state = mk_theory_hierarchy(\lambda thyn \bullet\\
&\quad interpret_theory_contents\ \{tc \mid \exists anc \bullet\\
&\qquad\qquad anc \in theory_ancestors\ state\ thyn\\
&\qquad \wedge \qquad theory_contents\ state\ anc\ tc\})
\end{aligned}$$

(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_INFO*s 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

$$
\begin{array}{l}
\textbf{dest\_state} : ('UD)PDS\_STATE \rightarrow \\
\quad (\quad ('UD)THEORY\_CONTENTS\ ADDR \\
\quad \times \quad ('UD)HIERARCHY\ ADDR \\
\quad \times \quad ('UD)\ THEORY\_CONTENTS\ STORE \\
\quad \times \quad ('UD)HIERARCHY\ STORE \\
\quad \times \quad ('UD)PDS\_THM\ STORE \quad)
\end{array}
$$

$$
\begin{array}{l}
\forall state \bullet dest\_state\ state = (\\
\quad ps\_current\_theory\ state, \\
\quad ps\_current\_hierarchy\ state, \\
\quad ps\_theory\_store\ state, \\
\quad ps\_hierarchy\_store\ state, \\
\quad ps\_theorem\_store\ state)
\end{array}
$$

Similarly, the following destructor function for theory contents is useful

HOL Constant

$$
\begin{array}{l}
\textbf{dest\_theory\_contents} : ('UD)THEORY\_CONTENTS \rightarrow \\
\quad (\quad STRING \\
\quad \times \quad (\mathbb{N} \times \mathbb{N})\ DICT \\
\quad \times \quad (TYPE \times \mathbb{N})\ DICT \\
\quad \times \quad STRING\ LIST \\
\quad \times \quad (SEQ \times \mathbb{N})\ DICT \\
\quad \times \quad (SEQ \times \mathbb{N})\ DICT \\
\quad \times \quad (SEQ \times \mathbb{N})\ DICT \\
\quad \times \quad \mathbb{N} \\
\quad \times \quad \mathbb{N}\ SET \\
\quad \times \quad 'UD \quad)
\end{array}
$$

$$
\begin{array}{l}
\forall tc \bullet dest\_theory\_contents\ tc = (\\
\quad tc\_name\ tc, \\
\quad tc\_ty\_env\ tc, \\
\quad tc\_con\_env\ tc, \\
\quad tc\_parents\ tc, \\
\quad tc\_axiom\_dict\ tc, \\
\quad tc\_definition\_dict\ tc, \\
\quad tc\_theorem\_dict\ tc, \\
\quad tc\_current\_level\ tc, \\
\quad tc\_deleted\_levels\ tc, \\
\quad tc\_user\_data\ tc)
\end{array}
$$

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

$$\textbf{current\_theory\_contents} : ('UD)PDS\_STATE \to ('UD)THEORY\_CONTENTS$$

$$\begin{array}{l}
\forall state \bullet current_theory_contents\ state = \\
let\ (cur_thy,\ _1,\ thy_st,\ _2,\ _3) = dest_state\ state \\
in \quad (\epsilon tc \bullet fetch\ cur_thy\ thy_st\ tc)
\end{array}$$

current_theory_name returns the name of the current theory.

HOL Constant

$$\textbf{current\_theory\_name} : ('UD)PDS\_STATE \to STRING$$

$$\begin{array}{l}
\forall state \bullet current_theory_name\ state = \\
\quad tc_name(current_theory_contents\ state)
\end{array}$$

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

$$\textbf{current\_abstract\_theory} : ('UD)PDS\_STATE \to THEORY$$

$$\begin{array}{l}
\forall state \bullet current_abstract_theory\ state = Fst(interpret_theory_contents\{tc | \exists anc \bullet \\
\qquad\qquad anc \in theory_ancestors\ state\ (current_theory_name\ state) \\
\qquad \wedge \quad theory_contents\ state\ anc\ tc\})
\end{array}$$

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

$$\textbf{theory\_info} : ('UD)PDS\_STATE \to STRING \to ('UD)\ THEORY\_INFO$$

$$\begin{array}{l}
\forall state\ name \bullet theory_info\ state\ name = \\
let\ (cur_thy,\ cur_hier,\ thy_st,\ hier_st,\ _1) = dest_state\ state \\
in\ let\ hier = \epsilon h \bullet fetch\ cur_hier\ hier_st\ h \\
in \quad \epsilon ti \bullet \quad tc_name(\epsilon tc \bullet fetch\ (ti_contents\ ti)\ thy_st\ tc) = name \\
\qquad \wedge \quad \neg ti_status\ ti = TSDeleted
\end{array}$$

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

$$\textbf{current\_theory\_status} : ('UD)PDS\_STATE \to STATUS$$

$$\begin{array}{l}
\forall state \bullet current_theory_status\ state = \\
ti_status\ (theory_info\ state\ (current_theory_name\ state))
\end{array}$$

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

$$\mathbf{check\_thm} : ('UD)PDS\_STATE \to ('UD)PDS\_THM \to BOOL$$

$$\begin{array}{l}
\forall state\ thm \bullet check_thm\ state\ thm \Leftrightarrow \\
let\ (cur_thy,\ cur_hier,\ thy_st,\ hier_st,\ _1) = dest_state\ state \\
in\ let\ tc = \epsilon tc \bullet fetch\ (pt_theory\ thm)\ thy_st\ tc \\
in\ let\ ti = theory_info\ state\ (tc_name\ tc) \\
in\ (\quad pt_theory\ thm = ti_contents\ ti \\
\land \quad ti_inscope\ ti \\
\land \quad \neg pt_level\ thm \in tc_deleted_levels\ tc\ )
\end{array}$$

check_thm_address determines whether an address identifies a theorem in the theorem store which is in scope:

HOL Constant

$$\mathbf{check\_thm\_address} : ('UD)PDS\_STATE \to ('UD)PDS\_THM\ ADDR \to BOOL$$

$$\begin{array}{l}
\forall state\ thm_ad \bullet check_thm_address\ state\ thm_ad \Leftrightarrow \\
let\ (_1,\ _2,\ _3,\ _4,\ thm_st) = dest_state\ state \\
in\ \exists thm \bullet fetch\ thm_ad\ thm_st\ thm \land check_thm\ state\ thm
\end{array}$$

fetch_thms fetches a list of theorems from the theorem store given a list of addresses. It is the responsibility of a function using *fetch_thms* to check the validity of the addresses:

HOL Constant

$$\begin{array}{l}
\mathbf{fetch_thms} : \\
('UD)PDS_STATE \to ('UD)PDS_THM\ ADDR\ LIST \to ('UD)PDS_THM\ LIST
\end{array}$$

$$\begin{array}{l}
\forall state\ thm_ads \bullet fetch_thms\ state\ thm_ads = \\
let\ (_1,\ _2,\ _3,\ _4,\ thm_st) = dest_state\ state \\
in\ Map\ (\lambda a \bullet \epsilon thm \bullet fetch\ a\ thm_st\ thm)\ thm_ads
\end{array}$$

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

$$\textbf{hierarchy\_ancestor} : ('UD)PDS\_STATE \to ((\text{'}UD)HIERARCHY)ADDR \to ((\text{'}UD)HIERARCHY)ADDR \to BOOL$$

$$\begin{array}{l}
\forall state\ hier_ad1\ hier_ad2\ \bullet hierarchy_ancestor\ state\ hier_ad1\ hier_ad2 \Leftrightarrow \\
let\ (_1,\ cur_hier,\ _2,\ hier_st,\ _3) = dest_state\ state \\
in\ \forall h1\ h2 \bullet \\
\quad fetch\ hier_ad1\ hier_st\ h1 \land fetch\ hier_ad2\ hier_st\ h2 \\
\Rightarrow \quad Elems\ (Map\ ti_contents\ h1) \subseteq Elems\ (Map\ ti_contents\ h2)
\end{array}$$

pds_mk_thm makes a theorem from a given sequent with *theory* and *level* values taken from the current theory of a state. It makes no checks whatsoever. It is the responsibility of a function using *mk_thm* to store the resulting theorem in the theorem store.

HOL Constant

$$\textbf{pds\_mk\_thm} : ('UD)PDS\_STATE \to SEQ \to ('UD)PDS\_THM$$

$$\begin{array}{l}
\forall state\ seq \bullet pds_mk_thm\ state\ seq = \\
let\ cur_thy = ps_current_theory\ state \\
in\ let\ lev = tc_current_level\ (current_theory_contents\ state) \\
in\ MkPDS_THM\ cur_thy\ lev\ seq
\end{array}$$

make_current does most of the work of opening a theory. It is defined here because it is needed both in *open_theory* and in *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 *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 *THEORY_INFO*).

HOL Constant

$$\textbf{make\_current} : STRING \to ('UD)PDS\_STATE \to ('UD)PDS\_STATE$$

$$\begin{array}{l}
\forall thyn\ state \bullet make_current\ thyn\ state = \\
let\ (cur_thy,\ cur_hier,\ thy_st,\ hier_st,\ thm_st) = dest_state\ state \\
in\ let\ f1 = \lambda ti \bullet tc_name(\epsilon tc \bullet fetch\ (ti_contents\ ti)\ thy_st\ tc) \\
in\ let\ f2 = \lambda ti \bullet (f1\ ti) \in theory_ancestors\ state\ thyn \\
in\ let\ f3 = \lambda ti \bullet MkTHEORY_INFO(ti_status\ ti)(f2\ ti)(ti_contents\ ti) \\
in\ let\ hier' = Map\ f3\ (\epsilon h \bullet fetch\ cur_hier\ hier_st\ h) \\
in\ let\ hier_st' = (cur_hier <- hier')\ hier_st \\
in\ let\ cur_thy' = ti_contents\ (theory_info\ state\ thyn) \\
in\ MkPDS_STATE\ cur_thy'\ cur_hier\ thy_st\ hier_st'\ thm_st
\end{array}$$

7.3 Operations on Hierarchies

7.3.1 *freeze_hierarchy*

freeze_hierarchy changes the status of all undeleted theories in the current hierarchy to *TSAncestor* (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 *TSAncestor*;
2. assign the result of step 1 to the current hierarchy;

HOL Constant

$$\textbf{freeze\_hierarchy} : ('UD)PDS\_STATE \to ('UD)PDS\_STATE$$

$$\begin{array}{l}
\forall state \bullet freeze_hierarchy\ state = \\
let\ (cur_thy,\ cur_hier,\ thy_st,\ hier_st,\ thm_st) = dest_state\ state \\
in\ let\ f1 = \lambda n \bullet if\ n = TSDeleted\ then\ n\ else\ TSAncestor \\
in\ let\ f2 = \lambda ti \bullet MkTHEORY_INFO(f1(ti_status\ ti))(ti_inscope\ ti)(ti_contents\ ti) \\
in\ let\ hier' = \ Map\ f2\ (\epsilon h \bullet fetch\ cur_hier\ hier_st\ h) \\
in\ let\ hier_st' = (cur_hier <- hier')\ hier_st \\
in\ MkPDS_STATE\ cur_thy\ cur_hier\ thy_st\ hier_st'\ thm_st
\end{array}$$

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 *TSAncestor* 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

$$\textbf{new\_hierarchy} : ('UD)PDS\_STATE \to ('UD)PDS\_STATE$$

$$\begin{array}{lll}
\multicolumn{3}{l}{\forall state \bullet new_hierarchy\ state =} \\
\multicolumn{3}{l}{let\ (cur_thy,\ cur_hier,\ thy_st,\ hier_st,\ thm_st) = dest_state\ state} \\
\multicolumn{3}{l}{in\ let\ hier = \epsilon h \bullet fetch\ cur_hier\ hier_st\ h} \\
\multicolumn{3}{l}{in} \\
if & \multicolumn{2}{l}{(\exists ti \bullet ti \in Elems\ hier} \\
 & \land & \neg ti_status\ ti \in \{TSAncestor;\ TSDeleted\}) \\
then & \multicolumn{2}{l}{state} \\
else & \multicolumn{2}{l}{let\ (hier_st',\ cur_hier') = \epsilon(st,\ a) \bullet new\ hier\ hier_st\ (st,\ a)} \\
 & \multicolumn{2}{l}{in\ MkPDS_STATE\ cur_thy\ cur_hier'\ thy_st\ hier_st'\ thm_st}
\end{array}$$

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[2].

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 hierararchy 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

$$\textbf{load\_hierarchy} : \quad ((\mathit{'UD})\mathit{HIERARCHY})\mathit{ADDR} \rightarrow (\mathit{'UD})\mathit{PDS\_STATE} \rightarrow (\mathit{'UD})\mathit{PDS\_STATE}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\forall hier\ state \bullet load_hierarchy\ hier\ state =} \\
\multicolumn{2}{l}{let\ (cur_thy,\ cur_hier,\ thy_st,\ hier_st,\ thm_st) = dest_state\ state} \\
\multicolumn{2}{l}{in} \\
if & \neg(hierarchy_ancestor\ state\ cur_hier\ hier) \\
then & state \\
else & let\ cur_thyn = current_theory_name\ state \\
 & in\ let\ st' = MkPDS_STATE\ cur_thy\ hier\ thy_st\ hier_st\ thm_st \\
 & in\ make_current\ cur_thyn\ st'
\end{array}$$

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] 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 $make_current$ to make the named theory the current theory.

HOL Constant

$$\textbf{open\_theory} : STRING \to ('UD)PDS\_STATE \to ('UD)PDS\_STATE$$

$$\forall thyn\ state \bullet open\_theory\ thyn\ state =$$
$$if\ \neg thyn \in theory\_names\ state \lor ti\_status(theory\_info\ state\ thyn) = TSDeleted$$
$$then \quad state$$
$$else \quad make\_current\ thyn\ state$$

7.4.2 *delete_theory*

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 *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 *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 *delete_theory*, we specify a function to compute the empty theory required in step 3. This is also used to support *new_theory*. The function is parameterised by the theory name and the desired parents.

HOL Constant

$$\textbf{empty\_theory} :$$
$$STRING \to (STRING\ LIST) \to 'UD \to ('UD)\ THEORY\_CONTENTS$$

$$\forall thyn\ pars\ ud \bullet empty\_theory\ thyn\ pars\ ud =$$
$$\quad MkTHEORY\_CONTENTS$$
$$\qquad thyn$$
$$\qquad initial\_dict \quad initial\_dict$$
$$\qquad pars$$
$$\qquad initial\_dict \quad initial\_dict \quad initial\_dict$$
$$\qquad 0 \quad \{\}$$
$$\qquad ud$$

For *delete_theory* we also need an arbitrary user datum value:

HOL Constant

$$\textbf{arbitrary\_ud} : 'UD$$

$$T$$

HOL Constant

$$\textbf{delete\_theory} : STRING \to ('UD)PDS\_STATE \to ('UD)PDS\_STATE$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall thyn\ state \bullet delete\_theory\ thyn\ state =} \\
if & \neg thyn \in theory\_names\ state \\
\vee & \neg ti\_status(theory\_info\ state\ thyn) = TSNormal \\
\vee & ti\_inscope(theory\_info\ state\ thyn) \\
\vee & \exists childname\ tc \bullet theory\_contents\ state\ childname\ tc \\
 & \wedge \quad thyn \in Elems\ (tc\_parents\ tc) \\
then & state \\
else & let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state \\
 & in\ let\ ti = theory\_info\ state\ thyn \\
 & in\ let\ f = \lambda ti' \bullet \\
 & \quad if\ ti' = ti \\
 & \quad then\ MkTHEORY\_INFO\ TSDeleted\ F\ (ti\_contents\ ti) \\
 & \quad else\ ti \\
 & in\ let\ hier' = Map\ f\ (\epsilon h \bullet fetch\ cur\_hier\ hier\_st\ h) \\
 & in\ let\ hier\_st' = (cur\_hier <- hier')\ hier\_st \\
 & in\ let\ thy = empty\_theory\ thyn\ []\ arbitrary\_ud \\
 & in\ let\ thy\_st' = (ti\_contents\ ti <- thy)\ thy\_st \\
 & in\ MkPDS\_STATE\ cur\_thy\ cur\_hier\ thy\_st'\ hier\_st'\ thm\_st
\end{array}
$$

7.4.3 *new_theory*

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 *new_theory* gives an inital 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 *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.

HOL Constant

$$\textbf{new\_theory} : STRING \to {}'UD \to ({}'UD)PDS\_STATE \to ({}'UD)PDS\_STATE$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall thyn\ ud\ state \bullet new\_theory\ thyn\ ud\ state =} \\
if & thyn \in theory\_names\ state \\
then & state \\
else & let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state \\
 & in\ let\ thy = empty\_theory\ thyn\ [current\_theory\_name\ state]\ ud \\
 & in\ let\ (thy\_st',\ addr) = \epsilon(st,\ a)\bullet\ new\ thy\ thy\_st\ (st,\ a) \\
 & in\ let\ ti = MkTHEORY\_INFO\ \ TSNormal\ \ F\ \ addr \\
 & in\ let\ hier' = \ Cons\ ti\ (\epsilon h \bullet fetch\ cur\_hier\ hier\_st\ h) \\
 & in\ let\ hier\_st' = (cur\_hier <\!-\ hier')\ hier\_st \\
 & in\ MkPDS\_STATE\ \ cur\_thy\ \ cur\_hier\ \ thy\_st'\ \ hier\_st'\ \ thm\_st
\end{array}
$$

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

$$
\begin{array}{l}
\textbf{new\_parent} : STRING \to ('UD)PDS\_STATE \to \\
\qquad ('UD)PDS\_STATE
\end{array}
$$

$$
\begin{array}{llll}
\multicolumn{4}{l}{\forall thyn\ state \bullet new\_parent\ thyn\ state =} \\
if & \multicolumn{3}{l}{\neg thyn \in theory\_names\ state} \\
\vee & \multicolumn{3}{l}{thyn \in Elems(tc\_parents\ (current\_theory\_contents\ state))} \\
\vee & \multicolumn{3}{l}{\exists ancn \bullet ancn \in (theory\_ancestors\ state\ thyn} \\
& \multicolumn{3}{l}{\qquad\qquad \setminus\ theory\_ancestors\ state\ (current\_theory\_name\ state))} \\
& \wedge & \multicolumn{2}{l}{let\ anc = \epsilon anc \bullet theory\_contents\ state\ ancn\ anc} \\
& & \multicolumn{2}{l}{in\ let\ cur\_thy = current\_abstract\_theory\ state} \\
& & in & (\exists ty\ nlev \bullet\ ty \in Dom(types\ cur\_thy) \\
& & & \wedge \quad lookup\ ty\ (tc\_ty\_env\ anc)\ nlev) \\
& & \vee & (\exists con\ tylev \bullet con \in Dom\ (constants\ cur\_thy) \\
& & & \wedge \quad lookup\ con\ (tc\_con\_env\ anc)\ tylev) \\
\\
then & \multicolumn{3}{l}{state} \\
else & \multicolumn{3}{l}{let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state} \\
& \multicolumn{3}{l}{in\ let\ cur\_thyn = current\_theory\_name\ state} \\
& \multicolumn{3}{l}{in\ let\ f1 = \lambda ti \bullet tc\_name(\epsilon tc \bullet fetch\ (ti\_contents\ ti)\ thy\_st\ tc)} \\
& \multicolumn{3}{l}{in\ let\ f2 = \lambda ti \bullet (f1\ ti) \in theory\_ancestors\ state\ thyn \vee ti\_inscope\ ti} \\
& \multicolumn{3}{l}{in\ let\ f3 = \lambda ti \bullet MkTHEORY\_INFO(ti\_status\ ti)(f2\ ti)(ti\_contents\ ti)} \\
& \multicolumn{3}{l}{in\ let\ hier' = \ Map\ f3\ (\epsilon h \bullet fetch\ cur\_hier\ hier\_st\ h)} \\
& \multicolumn{3}{l}{in\ let\ tc = current\_theory\_contents\ state} \\
& \multicolumn{3}{l}{in\ let\ (nm,\ t\_e,\ c\_e,\ pars,\ ax\_d,\ def\_d,\ thm\_d,\ lev,\ x\_levs,\ ud) =} \\
& \multicolumn{3}{l}{\qquad dest\_theory\_contents\ tc} \\
& \multicolumn{3}{l}{in\ let\ tc' = MkTHEORY\_CONTENTS} \\
& \multicolumn{3}{l}{\qquad nm\ t\_e\ c\_e\ (Cons\ thyn\ pars)\ ax\_d\ def\_d\ thm\_d\ lev\ x\_levs\ ud} \\
& \multicolumn{3}{l}{in\ let\ hier\_st' = (cur\_hier <\!-\ hier')\ hier\_st} \\
& \multicolumn{3}{l}{in\ let\ thy\_st' = (cur\_thy <\!-\ tc')\ thy\_st} \\
& \multicolumn{3}{l}{in\ MkPDS\_STATE\ cur\_thy\ \ cur\_hier\ \ thy\_st'\ \ hier\_st'\ \ thm\_st}
\end{array}
$$

7.4.5 *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

$$\textbf{duplicate\_theory} : STRING \to STRING \to ('UD)PDS\_STATE \to ('UD)PDS\_STATE$$

$$\begin{array}{lll}
\multicolumn{3}{l}{\forall thyn\ copyn\ state \bullet duplicate_theory\ thyn\ copyn\ state =} \\
if & & \neg thyn \in theory_names\ state \\
& \vee & copyn \in theory_names\ state \\
& \vee & thyn = \texttt{"}MIN\texttt{"} \\
& & \\
then & state & \\
else & \multicolumn{2}{l}{let\ (cur_thy,\ cur_hier,\ thy_st,\ hier_st,\ thm_st) = dest_state\ state} \\
& \multicolumn{2}{l}{in\ let\ tc = \epsilon tc \bullet theory_contents\ state\ thyn\ tc} \\
& \multicolumn{2}{l}{in\ let\ (nm,\ t_e,\ c_e,\ pars,\ ax_d,\ def_d,\ thm_d,\ lev,\ x_levs,\ ud) =} \\
& \multicolumn{2}{l}{\qquad dest_theory_contents\ tc} \\
& \multicolumn{2}{l}{in\ let\ tc' = MkTHEORY_CONTENTS} \\
& \multicolumn{2}{l}{\qquad copyn\ t_e\ c_e\ (Cons\ thyn\ pars)\ ax_d\ def_d\ thm_d\ lev\ x_levs\ ud} \\
& \multicolumn{2}{l}{in\ let\ (thy_st',\ addr) = \epsilon(st,\ a) \bullet} \\
& \multicolumn{2}{l}{\qquad new\ tc'\ thy_st\ (st,\ a)} \\
& \multicolumn{2}{l}{in\ let\ ti = MkTHEORY_INFO\ TSNormal\ F\ addr} \\
& \multicolumn{2}{l}{in\ let\ hier' = Cons\ ti\ (\epsilon h \bullet fetch\ cur_hier\ hier_st\ h)} \\
& \multicolumn{2}{l}{in\ let\ hier_st' = (cur_hier <- hier')\ hier_st} \\
& \multicolumn{2}{l}{in\ MkPDS_STATE\ cur_thy\ cur_hier\ thy_st'\ hier_st'\ thm_st}
\end{array}$$

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

SML

HOL Constant

$$\textbf{lock\_theory} : STRING \rightarrow ('UD)PDS\_STATE \rightarrow ('UD)PDS\_STATE$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall thyn\ state \bullet lock\_theory\ thyn\ state =} \\
if & \neg thyn \in theory\_names\ state \\
\vee & \neg ti\_status(theory\_info\ state\ thyn) = TSNormal \\
then & state \\
else & let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state \\
 & in\ let\ ti = theory\_info\ state\ thyn \\
 & in\ let\ f = \lambda ti' \bullet \\
 & \qquad if\ ti' = ti \\
 & \qquad then\ MkTHEORY\_INFO\ TSLocked(ti\_inscope\ ti)(ti\_contents\ ti) \\
 & \qquad else\ ti \\
 & in\ let\ hier' = Map\ f\ (\epsilon h \bullet fetch\ cur\_hier\ hier\_st\ h) \\
 & in\ let\ hier\_st' = (cur\_hier <- hier')\ hier\_st \\
 & in\ MkPDS\_STATE\ cur\_thy\ cur\_hier\ thy\_st\ hier\_st'\ thm\_st
\end{array}
$$

7.4.7 *unlock_theory*

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

HOL Constant

$$\textbf{unlock\_theory} : STRING \rightarrow ('UD)PDS\_STATE \rightarrow ('UD)PDS\_STATE$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall thyn\ state \bullet unlock\_theory\ thyn\ state =} \\
if & \neg thyn \in theory\_names\ state \\
\vee & \neg ti\_status(theory\_info\ state\ thyn) = TSLocked \\
then & state \\
else & let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state \\
 & in\ let\ ti = theory\_info\ state\ thyn \\
 & in\ let\ f = \lambda ti' \bullet \\
 & \qquad if\ ti' = ti \\
 & \qquad then\ MkTHEORY\_INFO\ TSNormal\ (ti\_inscope\ ti)\ (ti\_contents\ ti) \\
 & \qquad else\ ti \\
 & in\ let\ hier' = Map\ f\ (\epsilon h \bullet fetch\ cur\_hier\ hier\_st\ h) \\
 & in\ let\ hier\_st' = (cur\_hier <- hier')\ hier\_st \\
 & in\ MkPDS\_STATE\ cur\_thy\ cur\_hier\ thy\_st\ hier\_st'\ thm\_st
\end{array}
$$

7.5 Operations on Theory Contents

7.5.1 *save_thm*

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 *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.

HOL Constant

$$\mathbf{save\_thm} : STRING \to ('UD)PDS\_THM \to ('UD)PDS\_STATE \to ('UD)PDS\_STATE$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall key\ thm\ state \bullet save\_thm\ key\ thm\ state =} \\
\multicolumn{2}{l}{let\ tc = current\_theory\_contents\ state} \\
\multicolumn{2}{l}{in} \\
if & key \in keys\ (tc\_theorem\_dict\ tc) \\
\vee & \neg current\_theory\_status\ state = TSNormal \\
\vee & \neg check\_thm\ state\ thm \\
then & state \\
else & let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state \\
& in\ let\ level = if\ pt\_theory\ thm = cur\_thy\ then\ pt\_level\ thm\ else\ 0 \\
& in\ let\ thm\_d' = enter\ key\ (pt\_sequent\ thm,\ level)\ (tc\_theorem\_dict\ tc) \\
& in\ let\ (nm,\ t\_e,\ c\_e,\ pars,\ ax\_d,\ def\_d,\ thm\_d,\ lev,\ x\_levs,\ ud) = \\
& \quad dest\_theory\_contents\ tc \\
& in\ let\ tc' = MkTHEORY\_CONTENTS \\
& \quad nm\ t\_e\ c\_e\ pars\ ax\_d\ def\_d\ thm\_d'\ lev\ x\_levs\ ud \\
& in\ let\ thy\_st' = (cur\_thy <- tc')\ thy\_st \\
& in\ MkPDS\_STATE\ cur\_thy\ cur\_hier\ thy\_st'\ hier\_st\ thm\_st
\end{array}
$$

7.5.2 *delete_extension*

delete_extension allows the latest (undeleted) definition or axiom to be deleted from the current theory[4]. 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

$$\mathbf{is\_latest\_level} : ('UD)PDS\_STATE \to \mathbb{N} \to BOOL$$

$$\begin{array}{l}
\forall state\ lev \bullet is_latest_level\ state\ lev \Leftrightarrow \\
let\ tc = current_theory_contents\ state \\
in\ let\ (nm,\ t_e,\ c_e,\ pars,\ ax_d,\ def_d,\ thm_d,\ lev,\ x_levs,\ ud) = \\
\qquad\qquad\qquad\qquad dest_theory_contents\ tc \\
in\ let\ present = \{lv |\ (\exists\ _1\ key \bullet\ lookup\ key\ t_e\ (_1,\ lv)) \\
\qquad\qquad \lor \quad (\exists\ _1\ key \bullet\ lookup\ key\ c_e\ (_1,\ lv)) \\
\qquad\qquad \lor \quad (\exists\ _1\ key \bullet\ lookup\ key\ ax_d\ (_1,\ lv))\} \\
in \quad lev \in present \land (\forall lv \bullet lv \in present \Rightarrow lv \le lev)
\end{array}$$

HOL Constant

$$\mathbf{delete\_extension} : ('UD)PDS\_STATE \to ('UD)PDS\_STATE$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\forall state \bullet delete_extension\ state =} \\
\multicolumn{2}{l}{let\ tc = current_theory_contents\ state} \\
in & \\
if & (\neg(\exists lev \bullet is_latest_level\ state\ lev) \\
\lor & (\exists childname\ tc \bullet theory_contents\ state\ childname\ tc \\
 & \land \quad current_theory_name\ state \in Elems\ (tc_parents\ tc)) \\
\lor & \neg current_theory_status\ state = TSNormal) \\
then & state \\
else & let\ (cur_thy,\ cur_hier,\ thy_st,\ hier_st,\ thm_st) = dest_state\ state \\
 & in\ let\ (nm,\ t_e,\ c_e,\ pars,\ ax_d,\ def_d,\ thm_d,\ lev,\ x_levs,\ ud) = \\
 & \qquad\qquad\qquad dest_theory_contents\ tc \\
 & in\ let\ dlev = \epsilon lv \bullet is_latest_level\ state\ lv
\end{array}$$

[4] In 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.

$$
\begin{array}{l}
\quad in\ let\ def\_d' = block\_delete\ \{(\_1,\ lv) | lv = dlev\}\ def\_d \\
\quad in\ let\ ax\_d' = block\_delete\ \{(\_1,\ lv) | lv = dlev\}\ ax\_d \\
\quad in\ let\ t\_e' = block\_delete\ \{(\_1,\ lv) | lv = dlev\}\ t\_e \\
\quad in\ let\ c\_e' = block\_delete\ \{(\_1,\ lv) | lv = dlev\}\ c\_e \\
\quad in\ let\ lev' = lev+1 \\
\quad in\ let\ tc' = MkTHEORY\_CONTENTS \\
\qquad nm\ t\_e'\ c\_e'\ pars\ ax\_d'\ def\_d'\ thm\_d\ lev'\ (x\_levs \cup \{dlev\})\ ud \\
\quad in\ let\ thy\_st' = (cur\_thy <- tc')\ thy\_st \\
\quad in\ MkPDS\_STATE\ cur\_thy\ cur\_hier\ thy\_st'\ hier\_st\ thm\_st
\end{array}
$$

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

$$
\begin{array}{l}
\mathbf{delete\_thm} : STRING \to \\
\qquad ('UD)PDS\_STATE \to ('UD)PDS\_STATE
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall key\ state \bullet delete\_thm\ key\ state =} \\
\multicolumn{2}{l}{let\ tc = current\_theory\_contents\ state} \\
\multicolumn{2}{l}{in} \\
if & \neg key \in keys\ (tc\_theorem\_dict\ tc) \\
\vee & \neg current\_theory\_status\ state = TSNormal \\
then & state \\
else & let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state \\
 & in\ let\ (nm,\ t\_e,\ c\_e,\ pars,\ ax\_d,\ def\_d,\ thm\_d,\ lev,\ x\_levs,\ ud) = \\
 & \qquad\qquad dest\_theory\_contents\ tc \\
 & in\ let\ thm\_d' = delete\ key\ thm\_d \\
 & in\ let\ tc' = MkTHEORY\_CONTENTS \\
 & \qquad nm\ t\_e\ c\_e\ pars\ ax\_d\ def\_d\ thm\_d'\ lev\ x\_levs\ ud \\
 & in\ let\ thy\_st' = (cur\_thy <- tc')\ thy\_st \\
 & in\ MkPDS\_STATE\ cur\_thy\ cur\_hier\ thy\_st'\ hier\_st\ thm\_st
\end{array}
$$

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:

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

```
pds_new_axiom : TERM → STRING →
        ('UD)PDS_STATE → ('UD)PDS_STATE
─────────────────────────────────────
∀tm key state•pds_new_axiom tm key state =
let tc = current_theory_contents state
in let seq = ({}, tm)
in
if      key ∈ keys (tc_axiom_dict tc)
∨       ¬seq ∈ sequents (current_abstract_theory state)
∨       ¬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) = εsa•new(pds_mk_thm state seq)thm_st sa
        in MkPDS_STATE cur_thy cur_hier thy_st' hier_st thm_st'
```

7.5.5 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 × ('UD)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 × ('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, 'UD)SUBSYS_INPUT × ('UD)PDS_STATE) ↔
        (SEQ × ((STRING × ℕ) LIST) × ((STRING × TYPE) LIST) × (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

$$
\begin{array}{lll}
\textbf{make\_definition}: & ('IP, 'UD)DEFINER \rightarrow \\
& (('IP, 'UD)PDS\_INPUT \times ('UD)PDS\_STATE) \rightarrow \\
& ('UD)PDS\_STATE
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall definer\ pars\ thm\_ads\ state \bullet} \\
\multicolumn{2}{l}{make\_definition\ definer\ ((pars,\ thm\_ads),\ state) =} \\
if & \exists thm\_ad \bullet thm\_ad \in Elems\ thm\_ads \wedge \neg check\_thm\_address\ state\ thm\_ad \\
then & state \\
else & let\ thms = fetch\_thms\ state\ thm\_ads \\
& in \\
if & \neg((pars,\ thms),\ state) \in Dom\ definer \\
then & state \\
else & let\ (seq,\ tys,\ cons,\ ks) = definer@((pars,\ thms),\ state) \\
& in\ let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state \\
& in\ let\ tc = current\_theory\_contents\ state \\
& in\ let\ (nm,\ t\_e,\ c\_e,\ pars,\ ax\_d,\ def\_d,\ thm\_d,\ lev,\ x\_levs,\ ud) = \\
& \qquad\qquad\qquad\qquad dest\_theory\_contents\ tc \\
& in\ let\ lev' = lev{+}1 \\
& in\ let\ def\_d' = Fold\ (\lambda k \bullet enter\ k\ (seq,\ lev'))\ ks\ def\_d \\
& in\ let\ t\_e' = Fold\ (Uncurry\ enter)\ (Map\ (\lambda(s,x) \bullet (s,\ (x,\ lev')))\ tys)\ t\_e \\
& in\ let\ c\_e' = Fold\ (Uncurry\ enter)\ (Map\ (\lambda(s,x) \bullet (s,\ (x,\ lev')))\ cons)\ c\_e \\
& in\ let\ tc' = MkTHEORY\_CONTENTS \\
& \qquad nm\ t\_e'\ c\_e'\ pars\ ax\_d\ def\_d'\ thm\_d\ lev'\ x\_levs\ ud \\
& in\ let\ thy\_st' = (cur\_thy <- tc')\ thy\_st \\
& in\ let\ (thm\_st',\ \_1) = \epsilon sa \bullet new(pds\_mk\_thm\ state\ seq)thm\_st\ sa \\
& in\ MkPDS\_STATE\ cur\_thy\ cur\_hier\ thy\_st'\ hier\_st\ thm\_st'
\end{array}
$$

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 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

$$\mathbf{pds\_new\_type} : \quad STRING \to \mathbb{N} \to ('UD)PDS\_STATE \to ('UD)PDS\_STATE$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall ty\ arity\ state \bullet} \\
\multicolumn{2}{l}{pds\_new\_type\ ty\ arity\ state =} \\
if & ty \in Dom(types\ (current\_abstract\_theory\ state)) \\
then & state \\
else & let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state \\
& in\ let\ tc = current\_theory\_contents\ state \\
& in\ let\ (nm,\ t\_e,\ c\_e,\ pars,\ ax\_d,\ def\_d,\ thm\_d,\ lev,\ x\_levs,\ ud) = \\
& \qquad\qquad\qquad dest\_theory\_contents\ tc \\
& in\ let\ lev' = lev{+}1 \\
& in\ let\ t\_e' = enter\ ty\ (arity,\ lev')\ t\_e \\
& in\ let\ tc' = MkTHEORY\_CONTENTS \\
& \qquad nm\ t\_e'\ c\_e\ pars\ ax\_d\ def\_d\ thm\_d\ lev'\ x\_levs\ ud \\
& in\ let\ thy\_st' = (cur\_thy <- tc')\ thy\_st \\
& in\ MkPDS\_STATE\ cur\_thy\ cur\_hier\ thy\_st'\ hier\_st\ thm\_st
\end{array}
$$

7.5.7 *pds_new_constant*

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 *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 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.

HOL Constant

$$\textbf{pds\_new\_constant} : STRING \rightarrow TYPE \rightarrow ('UD)PDS\_STATE \rightarrow ('UD)PDS\_STATE$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\forall con\ ty\ state \bullet} \\
\multicolumn{2}{l}{pds_new_constant\ con\ ty\ state =} \\
if & con \in Dom(constants\ (current_abstract_theory\ state)) \\
\vee & \neg ty \in wf_type\ (types\ (current_abstract_theory\ state)) \\
 & \\
then & state \\
else & let\ (cur_thy,\ cur_hier,\ thy_st,\ hier_st,\ thm_st) = dest_state\ state \\
 & in\ let\ tc = current_theory_contents\ state \\
 & in\ let\ (nm,\ t_e,\ c_e,\ pars,\ ax_d,\ def_d,\ thm_d,\ lev,\ x_levs,\ ud) = \\
 & \qquad\qquad\qquad dest_theory_contents\ tc \\
 & in\ let\ lev' = lev{+}1 \\
 & in\ let\ c_e' = enter\ con\ (ty,\ lev')\ c_e \\
 & in\ let\ tc' = MkTHEORY_CONTENTS \\
 & \qquad nm\ t_e\ c_e'\ pars\ ax_d\ def_d\ thm_d\ lev'\ x_levs\ ud \\
 & in\ let\ thy_st' = (cur_thy <- tc')\ thy_st \\
 & in\ MkPDS_STATE\ cur_thy\ cur_hier\ thy_st'\ hier_st\ thm_st
\end{array}$$

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, *make_inference*, parameterised by a function which represents an implementation of such a set of rules.

SML

```
declare_type_abbrev("INFERRER", ["'IP", "'UD"],
        ⌜:(('IP, 'UD)SUBSYS_INPUT × ('UD)PDS_STATE) ↔ SEQ⌝);
```

An *INFERRER* is a partial function, represented as a set of pairs, which, returns a sequent.

The algorithm for *make_inference* is as follows:

1. if the input and state are not in the domain of the *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 *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

$$
\begin{array}{ll}
\textbf{make\_inference}: & ('IP, 'UD)INFERRER \to \\
& (('IP, 'UD)PDS\_INPUT \times ('UD)PDS\_STATE) \to \\
& ('UD)PDS\_STATE
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall inferrer\ pars\ thm\_ads\ state \bullet} \\
\multicolumn{2}{l}{make\_inference\ inferrer\ ((pars,\ thm\_ads),\ state) =} \\
if & (\exists thm\_ad \bullet thm\_ad \in Elems\ thm\_ads \land \neg check\_thm\_address\ state\ thm\_ad) \\
then & state \\
else & let\ thms = fetch\_thms\ state\ thm\_ads \\
& in \\
if & \neg((pars,\ thms),\ state) \in Dom\ inferrer \\
\lor & \neg current\_theory\_status\ state = TSNormal \\
then & state \\
else & let\ seq = inferrer@((pars,\ thms),\ state) \\
& in\ let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ thm\_st) = dest\_state\ state \\
& in\ let\ (thm\_st',\ \_1) = \epsilon sa \bullet new(pds\_mk\_thm\ state\ seq) thm\_st\ sa \\
& in\ MkPDS\_STATE\ cur\_thy\ cur\_hier\ thy\_st\ hier\_st\ thm\_st'
\end{array}
$$

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 *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

$$
\begin{array}{ll}
\textbf{allowed}: & ('IP, 'UD)DEFINER \to \\
& ('IP, 'UD)INFERRER \to \\
& (\ ('UD)PDS\_STATE \to ('UD)PDS\_STATE\ ) \to BOOL
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall definer\ inferrer\ trans \bullet} \\
\multicolumn{2}{l}{allowed\ definer\ inferrer\ trans \Leftrightarrow} \\
& trans = freeze\_hierarchy \\
\lor & trans = new\_hierarchy \\
\lor & \exists addr \bullet trans = load\_hierarchy\ addr \\
\lor & \exists thyn \bullet trans = open\_theory\ thyn \\
\lor & \exists thyn \bullet trans = delete\_theory\ thyn
\end{array}
$$

$$\begin{array}{ll}
\lor & \exists thyn\ ud \bullet trans = new_theory\ thyn\ ud \\
\lor & \exists thyn \bullet trans = new_parent\ thyn \\
\lor & \exists thyn\ copyn \bullet trans = duplicate_theory\ thyn\ copyn \\
\lor & \exists thyn \bullet trans = lock_theory\ thyn \\
\lor & \exists thyn \bullet trans = unlock_theory\ thyn \\
\lor & \exists key\ thm \bullet trans = save_thm\ key\ thm \\
\lor & trans = delete_extension \\
\lor & \exists key \bullet trans = delete_thm\ key \\
\lor & \exists tm\ key \bullet trans = pds_new_axiom\ tm\ key \\
\lor & \exists ty\ arity \bullet trans = pds_new_type\ ty\ arity \\
\lor & \exists con\ ty \bullet trans = pds_new_constant\ con\ ty \\
\lor & \exists pars\ thm_ads \bullet trans = Curry(make_definition\ definer)(pars,\ thm_ads) \\
\lor & \exists pars\ thm_ads \bullet trans = Curry(make_inference\ inferrer)(pars,\ thm_ads)
\end{array}$$

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:

SML

```
declare_type_abbrev("INTERPRETER", ["'IP", "'UD"],
        ⌜:('IP, 'UD)DEFINER → ('IP, 'UD)INFERRER →
                ('IP,'UD)PDS_INPUT →
                ('UD)PDS_STATE → ('UD)PDS_STATE⌝);
```

The loosely specified function *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:

Discussion

$$\begin{array}{ll}
: & (('IP \times ('UD)PDS_THM\ list) \times ('UD)PDS_STATE) \\
\to & (('UD)PDS_STATE \times (('UD)PDS_THM\ STORE))
\end{array}$$

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).

pds constructs *HOL_SYSTEM*s in the sense of [8], allowing us to assert the critical properties defined in that document for these systems.

HOL Constant

$$\begin{array}{l}
\mathbf{pds} : ('IP, 'UD)DEFINER \to \\
\quad ('IP, 'UD)INFERRER \to \\
\quad ('IP, 'UD)INTERPRETER \to \\
\quad (\quad ('IP, 'UD)PDS_INPUT, \\
\qquad ('UD)PDS_THM\ STORE, \\
\qquad ('UD)PDS_STATE \quad)HOL_SYSTEM
\end{array}$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall definer\ inferrer\ interpreter \bullet} \\
\multicolumn{2}{l}{pds\ definer\ inferrer\ interpreter =} \\
(& (\lambda((pars,\ thm\_ads),\ state)\bullet \\
 & let\ state' = interpreter\ definer\ inferrer\ (pars,\ thm\_ads)\ state \\
 & in\ (state',\ ps\_theorem\_store\ state')), \\
 & interpret\_state\ )
\end{array}
$$

8.3 Subsystem Critical Properties

We will use the term *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 *DEFINER* is good if the extension of abstract theories induced by its intended effect on concrete theories is definitional:

HOL Constant

$$\mathbf{good\_definer} : ('IP,\ 'UD)DEFINER \to BOOL$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall definer \bullet good\_definer\ definer \Leftrightarrow} \\
\multicolumn{2}{l}{\forall pars\ thms\ state \bullet} \\
 & ((pars,\ thms),\ state) \in Dom\ definer \\
\Rightarrow & let\ thy = current\_abstract\_theory\ state \\
 & in\ let\ (tyenv,\ conenv,\ axs) = rep\_theory\ thy \\
 & in\ let\ (seq,\ tys,\ cons,\ \_1) = definer@((pars,\ thms),\ state) \\
 & in\ let\ tyenv' = Fold\ (\lambda tn \bullet \lambda te \bullet te \cup \{tn\})\ tys\ tyenv \\
 & in\ let\ conenv' = Fold\ (\lambda st \bullet \lambda ce \bullet ce \cup \{st\})\ cons\ conenv \\
 & in\ let\ axs' = axs \cup \{seq\} \\
 & in\ let\ thy' = abs\_theory(tyenv',\ conenv',\ axs') \\
 & in\ \ thy \in definitional\_extension\ thy'
\end{array}
$$

An *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 *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 $good\_inferrer$.

HOL Constant

$$\mathbf{good\_inferrer} : 'U \to ('IP,\ 'UD)INFERRER \to BOOL$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall v\ inferrer \bullet good\_inferrer\ v\ inferrer \Leftrightarrow} \\
\multicolumn{2}{l}{\forall pars\ thms\ state \bullet} \\
 & ((pars,\ thms),\ state) \in Dom\ inferrer \\
\Rightarrow & let\ thy = current\_abstract\_theory\ state \\
 & in\ let\ seq = inferrer@((pars,\ thms),\ state) \\
 & in\ (\quad seq \in sequents\ thy \\
 & \wedge \quad\ \ seq \in valid\ v\ thy)
\end{array}
$$

An *INTERPRETER* is good if it always returns allowable state transitions.

HOL Constant

$$\textbf{good\_interpreter} : ('IP, 'UD)INTERPRETER \to BOOL$$

$$\begin{array}{l}
\forall interpreter \bullet good_interpreter\ interpreter \Leftrightarrow \\
\forall definer\ inferrer\ pars\ thm_ads \bullet \\
\qquad allowed\ definer\ inferrer(interpreter\ definer\ inferrer(pars,\ thm_ads))
\end{array}$$

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

$$\begin{array}{llll}
?\vdash & \forall definer\ inferrer\ interpreter \bullet & & \\
 & & (& good_definer\ definer \\
 & & \wedge & good_inferrer\ inferrer \\
 & & \wedge & \forall v{:}'U \bullet good_interpreter\ v\ interpreter) \\
 & \Rightarrow & (& standard\ (pds\ definer\ inferrer\ interpreter) \\
 & & \wedge & \forall v{:}'U \bullet validity_preserving\ v\ (pds\ definer\ inferrer\ interpreter))
\end{array}$$

or in the syntactic formulation:

Conjecture

$$\begin{array}{llll}
?\vdash & \forall definer\ inferrer\ interpreter \bullet & & \\
 & & (& good_definer\ definer \\
 & & \wedge & good_inferrer\ inferrer \\
 & & \wedge & \forall v{:}'U \bullet good_interpreter\ v\ interpreter) \\
 & \Rightarrow & (& standard\ (pds\ definer\ inferrer\ interpreter) \\
 & & \wedge & derivability_preserving\ (pds\ definer\ inferrer\ interpreter))
\end{array}$$

9 THEORY LISTING

10 THE THEORY spc005

10.1 Parents

spc004

10.2 Constants

initial_dict $(CHAR\ LIST \times {'X})\ SET$

enter $CHAR\ LIST$
$\quad \to {'X}$
$\quad \to (CHAR\ LIST \times {'X})\ SET$
$\quad \to (CHAR\ LIST \times {'X})\ SET$

lookup $CHAR\ LIST \to (CHAR\ LIST \times {'X})\ SET \to {'X} \to BOOL$

delete $CHAR\ LIST \to (CHAR\ LIST \times {'X})\ SET \to (CHAR\ LIST \times {'X})\ SET$

block_delete ${'X}\ SET \to (CHAR\ LIST \times {'X})\ SET \to (CHAR\ LIST \times {'X})\ SET$

keys $(CHAR\ LIST \times {'X})\ SET \to CHAR\ LIST\ SET$

$<- ${'X}\ ADDR \to {'X} \to ({'X}\ ADDR \times {'X})\ SET \to ({'X}\ ADDR \times {'X})\ SET$

fetch ${'X}\ ADDR \to ({'X}\ ADDR \times {'X})\ SET \to {'X} \to BOOL$

new ${'X}$
$\quad \to ({'X}\ ADDR \times {'X})\ SET$
$\quad \to ({'X}\ ADDR \times {'X})\ SET \times {'X}\ ADDR$
$\quad \to BOOL$

initial_store
$({'X}\ ADDR \times {'X})\ SET$

tc_user_data ${'UD}\ THEORY\_CONTENTS \to {'UD}$

tc_deleted_levels
${'UD}\ THEORY\_CONTENTS \to \mathbb{N}\ SET$

tc_current_level
${'UD}\ THEORY\_CONTENTS \to \mathbb{N}$

tc_theorem_dict
${'UD}\ THEORY\_CONTENTS$
$\quad \to (CHAR\ LIST \times (TERM\ SET \times TERM) \times \mathbb{N})\ SET$

tc_definition_dict
${'UD}\ THEORY\_CONTENTS$
$\quad \to (CHAR\ LIST \times (TERM\ SET \times TERM) \times \mathbb{N})\ SET$

tc_axiom_dict
${'UD}\ THEORY\_CONTENTS$
$\quad \to (CHAR\ LIST \times (TERM\ SET \times TERM) \times \mathbb{N})\ SET$

tc_parents ${'UD}\ THEORY\_CONTENTS \to CHAR\ LIST\ LIST$

tc_con_env ${'UD}\ THEORY\_CONTENTS \to (CHAR\ LIST \times TYPE \times \mathbb{N})\ SET$

tc_ty_env ${'UD}\ THEORY\_CONTENTS \to (CHAR\ LIST \times \mathbb{N} \times \mathbb{N})\ SET$

tc_name ${'UD}\ THEORY\_CONTENTS \to CHAR\ LIST$

MkTHEORY_CONTENTS
$CHAR\ LIST$
$\quad \to (CHAR\ LIST \times \mathbb{N} \times \mathbb{N})\ SET$
$\quad \to (CHAR\ LIST \times TYPE \times \mathbb{N})\ SET$
$\quad \to CHAR\ LIST\ LIST$
$\quad \to (CHAR\ LIST \times (TERM\ SET \times TERM) \times \mathbb{N})\ SET$
$\quad \to (CHAR\ LIST \times (TERM\ SET \times TERM) \times \mathbb{N})\ SET$
$\quad \to (CHAR\ LIST \times (TERM\ SET \times TERM) \times \mathbb{N})\ SET$
$\quad \to \mathbb{N}$
$\quad \to \mathbb{N}\ SET$
$\quad \to {'UD}$

$\rightarrow {'UD}\ THEORY\_CONTENTS$

TSDeleted $ONE + ONE + ONE + ONE$

TSAncestor $ONE + ONE + ONE + ONE$

TSLocked $ONE + ONE + ONE + ONE$

TSNormal $ONE + ONE + ONE + ONE$

ti_contents $'UD\ THEORY\_INFO \rightarrow 'UD\ THEORY\_CONTENTS\ ADDR$

ti_inscope $'UD\ THEORY\_INFO \rightarrow BOOL$

ti_status $'UD\ THEORY\_INFO \rightarrow ONE + ONE + ONE + ONE$

MkTHEORY_INFO

$ONE + ONE + ONE + ONE$
$\rightarrow BOOL$
$\rightarrow 'UD\ THEORY\_CONTENTS\ ADDR$
$\rightarrow 'UD\ THEORY\_INFO$

pt_sequent $'UD\ PDS\_THM \rightarrow TERM\ SET \times TERM$

pt_level $'UD\ PDS\_THM \rightarrow \mathbb{N}$

pt_theory $'UD\ PDS\_THM \rightarrow 'UD\ THEORY\_CONTENTS\ ADDR$

MkPDS_THM $'UD\ THEORY\_CONTENTS\ ADDR$
$\rightarrow \mathbb{N}$
$\rightarrow TERM\ SET \times TERM$
$\rightarrow 'UD\ PDS\_THM$

ps_theorem_store

$'UD\ PDS\_STATE \rightarrow ('UD\ PDS\_THM\ ADDR \times 'UD\ PDS\_THM)\ SET$

ps_hierarchy_store

$'UD\ PDS\_STATE$
$\rightarrow ('UD\ THEORY\_INFO\ LIST\ ADDR \times 'UD\ THEORY\_INFO\ LIST)$
SET

ps_theory_store

$'UD\ PDS\_STATE$
$\rightarrow ('UD\ THEORY\_CONTENTS\ ADDR \times 'UD\ THEORY\_CONTENTS)$
SET

ps_current_hierarchy

$'UD\ PDS\_STATE \rightarrow 'UD\ THEORY\_INFO\ LIST\ ADDR$

ps_current_theory

$'UD\ PDS\_STATE \rightarrow 'UD\ THEORY\_CONTENTS\ ADDR$

MkPDS_STATE $'UD\ THEORY\_CONTENTS\ ADDR$
$\rightarrow 'UD\ THEORY\_INFO\ LIST\ ADDR$
$\rightarrow ('UD\ THEORY\_CONTENTS\ ADDR \times 'UD\ THEORY\_CONTENTS)$
SET
$\rightarrow ('UD\ THEORY\_INFO\ LIST\ ADDR \times 'UD\ THEORY\_INFO\ LIST)$
SET
$\rightarrow ('UD\ PDS\_THM\ ADDR \times 'UD\ PDS\_THM)\ SET$
$\rightarrow 'UD\ PDS\_STATE$

initial_theory

$'UD$
$\rightarrow ('UD\ THEORY\_CONTENTS\ ADDR \times 'UD\ THEORY\_CONTENTS)$
SET
$\times 'UD\ THEORY\_INFO$

initial_state

$'UD \rightarrow 'UD\ PDS\_STATE$

theory_contents

$'UD\ PDS\_STATE \rightarrow CHAR\ LIST \rightarrow 'UD\ THEORY\_CONTENTS \rightarrow BOOL$

theory_names $'UD\ PDS\_STATE \rightarrow CHAR\ LIST\ SET$

theory_ancestors

$'UD\ PDS\_STATE \rightarrow CHAR\ LIST \rightarrow CHAR\ LIST\ SET$

interpret_theory_contents

$'UD\ THEORY\_CONTENTS\ SET$
$\rightarrow THEORY$
$\times (TERM\ SET \times TERM)\ SET$

$\times\ (TERM\ SET \times TERM)\ SET$

interpret_state
$'UD\ PDS\_STATE \rightarrow THEORY\_HIERARCHY$

dest_state $'UD\ PDS\_STATE$
$\rightarrow {'UD}\ THEORY\_CONTENTS\ ADDR$
$\times\ {'UD}\ THEORY\_INFO\ LIST\ ADDR$
$\times\ ('UD\ THEORY\_CONTENTS\ ADDR \times {'UD}\ THEORY\_CONTENTS)$
SET
$\times\ ('UD\ THEORY\_INFO\ LIST\ ADDR$
$\times\ {'UD}\ THEORY\_INFO\ LIST)\ SET$
$\times\ ('UD\ PDS\_THM\ ADDR \times {'UD}\ PDS\_THM)\ SET$

dest_theory_contents
$'UD\ THEORY\_CONTENTS$
$\rightarrow CHAR\ LIST$
$\times\ (CHAR\ LIST \times \mathbb{N} \times \mathbb{N})\ SET$
$\times\ (CHAR\ LIST \times TYPE \times \mathbb{N})\ SET$
$\times\ CHAR\ LIST\ LIST$
$\times\ (CHAR\ LIST \times (TERM\ SET \times TERM) \times \mathbb{N})\ SET$
$\times\ (CHAR\ LIST \times (TERM\ SET \times TERM) \times \mathbb{N})\ SET$
$\times\ (CHAR\ LIST \times (TERM\ SET \times TERM) \times \mathbb{N})\ SET$
$\times\ \mathbb{N}$
$\times\ \mathbb{N}\ SET$
$\times\ {'UD}$

current_theory_contents
$'UD\ PDS\_STATE \rightarrow {'UD}\ THEORY\_CONTENTS$

current_theory_name
$'UD\ PDS\_STATE \rightarrow CHAR\ LIST$

current_abstract_theory
$'UD\ PDS\_STATE \rightarrow THEORY$

theory_info $'UD\ PDS\_STATE \rightarrow CHAR\ LIST \rightarrow {'UD}\ THEORY\_INFO$

current_theory_status
$'UD\ PDS\_STATE \rightarrow ONE + ONE + ONE + ONE$

check_thm $'UD\ PDS\_STATE \rightarrow {'UD}\ PDS\_THM \rightarrow BOOL$

check_thm_address
$'UD\ PDS\_STATE \rightarrow {'UD}\ PDS\_THM\ ADDR \rightarrow BOOL$

fetch_thms $'UD\ PDS\_STATE$
$\rightarrow {'UD}\ PDS\_THM\ ADDR\ LIST$
$\rightarrow {'UD}\ PDS\_THM\ LIST$

hierarchy_ancestor
$'UD\ PDS\_STATE$
$\rightarrow {'UD}\ THEORY\_INFO\ LIST\ ADDR$
$\rightarrow {'UD}\ THEORY\_INFO\ LIST\ ADDR$
$\rightarrow BOOL$

pds_mk_thm $'UD\ PDS\_STATE \rightarrow TERM\ SET \times TERM \rightarrow {'UD}\ PDS\_THM$

make_current $CHAR\ LIST \rightarrow {'UD}\ PDS\_STATE \rightarrow {'UD}\ PDS\_STATE$

freeze_hierarchy
$'UD\ PDS\_STATE \rightarrow {'UD}\ PDS\_STATE$

new_hierarchy
$'UD\ PDS\_STATE \rightarrow {'UD}\ PDS\_STATE$

load_hierarchy
$'UD\ THEORY\_INFO\ LIST\ ADDR$
$\rightarrow {'UD}\ PDS\_STATE$
$\rightarrow {'UD}\ PDS\_STATE$

open_theory $CHAR\ LIST \rightarrow {'UD}\ PDS\_STATE \rightarrow {'UD}\ PDS\_STATE$

empty_theory $CHAR\ LIST \rightarrow CHAR\ LIST\ LIST \rightarrow {'UD} \rightarrow {'UD}\ THEORY\_CONTENTS$

arbitrary_ud $'UD$

delete_theory
$CHAR\ LIST \rightarrow {'UD}\ PDS\_STATE \rightarrow {'UD}\ PDS\_STATE$

new_theory $CHAR\ LIST \rightarrow {'}UD \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

new_parent $CHAR\ LIST \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

duplicate_theory
$CHAR\ LIST \rightarrow CHAR\ LIST \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

lock_theory $CHAR\ LIST \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

unlock_theory
$CHAR\ LIST \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

save_thm $CHAR\ LIST \rightarrow {'}UD\ PDS\_THM \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

is_latest_level
${'}UD\ PDS\_STATE \rightarrow \mathbb{N} \rightarrow BOOL$

delete_extension
${'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

delete_thm $CHAR\ LIST \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

pds_new_axiom
$TERM \rightarrow CHAR\ LIST \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

make_definition

$$\begin{aligned}
&((({'}IP \times {'}UD\ PDS_THM\ LIST) \times {'}UD\ PDS_STATE)\\
&\qquad \times (TERM\ SET \times TERM)\\
&\qquad \times (CHAR\ LIST \times \mathbb{N})\ LIST\\
&\qquad \times (CHAR\ LIST \times TYPE)\ LIST\\
&\qquad \times CHAR\ LIST\ LIST)\ SET\\
&\quad \rightarrow ({'}IP \times {'}UD\ PDS_THM\ ADDR\ LIST) \times {'}UD\ PDS_STATE\\
&\quad \rightarrow {'}UD\ PDS_STATE
\end{aligned}$$

pds_new_type $CHAR\ LIST \rightarrow \mathbb{N} \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

pds_new_constant
$CHAR\ LIST \rightarrow TYPE \rightarrow {'}UD\ PDS\_STATE \rightarrow {'}UD\ PDS\_STATE$

make_inference

$$\begin{aligned}
&((({'}IP \times {'}UD\ PDS_THM\ LIST) \times {'}UD\ PDS_STATE)\\
&\qquad \times TERM\ SET\\
&\qquad \times TERM)\ SET\\
&\quad \rightarrow ({'}IP \times {'}UD\ PDS_THM\ ADDR\ LIST) \times {'}UD\ PDS_STATE\\
&\quad \rightarrow {'}UD\ PDS_STATE
\end{aligned}$$

allowed

$$\begin{aligned}
&((({'}IP \times {'}UD\ PDS_THM\ LIST) \times {'}UD\ PDS_STATE)\\
&\qquad \times (TERM\ SET \times TERM)\\
&\qquad \times (CHAR\ LIST \times \mathbb{N})\ LIST\\
&\qquad \times (CHAR\ LIST \times TYPE)\ LIST\\
&\qquad \times CHAR\ LIST\ LIST)\ SET\\
&\quad \rightarrow ((({'}IP \times {'}UD\ PDS_THM\ LIST) \times {'}UD\ PDS_STATE)\\
&\qquad \times TERM\ SET\\
&\qquad \times TERM)\ SET\\
&\quad \rightarrow ({'}UD\ PDS_STATE \rightarrow {'}UD\ PDS_STATE)\\
&\quad \rightarrow BOOL
\end{aligned}$$

pds

$$\begin{aligned}
&((({'}IP \times {'}UD\ PDS_THM\ LIST) \times {'}UD\ PDS_STATE)\\
&\qquad \times (TERM\ SET \times TERM)\\
&\qquad \times (CHAR\ LIST \times \mathbb{N})\ LIST\\
&\qquad \times (CHAR\ LIST \times TYPE)\ LIST\\
&\qquad \times CHAR\ LIST\ LIST)\ SET\\
&\quad \rightarrow ((({'}IP \times {'}UD\ PDS_THM\ LIST) \times {'}UD\ PDS_STATE)\\
&\qquad \times TERM\ SET\\
&\qquad \times TERM)\ SET\\
&\quad \rightarrow (((({'}IP \times {'}UD\ PDS_THM\ LIST) \times {'}UD\ PDS_STATE)\\
&\qquad\quad \times (TERM\ SET \times TERM)\\
&\qquad\quad \times (CHAR\ LIST \times \mathbb{N})\ LIST\\
&\qquad\quad \times (CHAR\ LIST \times TYPE)\ LIST\\
&\qquad\quad \times CHAR\ LIST\ LIST)\ SET\\
&\qquad \rightarrow ((({'}IP \times {'}UD\ PDS_THM\ LIST) \times {'}UD\ PDS_STATE)\\
&\qquad\quad \times TERM\ SET\\
&\qquad\quad \times TERM)\ SET
\end{aligned}$$

$\rightarrow {'}IP \times {'}UD\ PDS\_THM\ ADDR\ LIST$
$\rightarrow {'}UD\ PDS\_STATE$
$\rightarrow {'}UD\ PDS\_STATE)$
$\rightarrow (({'}IP \times {'}UD\ PDS\_THM\ ADDR\ LIST) \times {'}UD\ PDS\_STATE$
$\rightarrow {'}UD\ PDS\_STATE$
$\times ({'}UD\ PDS\_THM\ ADDR \times {'}UD\ PDS\_THM)\ SET)$
$\times ({'}UD\ PDS\_STATE \rightarrow THEORY\_HIERARCHY)$

good_definer $((({'}IP \times {'}UD\ PDS\_THM\ LIST) \times {'}UD\ PDS\_STATE)$
$\times (TERM\ SET \times TERM)$
$\times (CHAR\ LIST \times \mathbb{N})\ LIST$
$\times (CHAR\ LIST \times TYPE)\ LIST$
$\times CHAR\ LIST\ LIST)\ SET$
$\rightarrow BOOL$

good_inferrer
${'}U$
$\rightarrow ((({'}IP \times {'}UD\ PDS\_THM\ LIST) \times {'}UD\ PDS\_STATE)$
$\times TERM\ SET$
$\times TERM)\ SET$
$\rightarrow BOOL$

good_interpreter
$(((({'}IP \times {'}UD\ PDS\_THM\ LIST) \times {'}UD\ PDS\_STATE)$
$\times (TERM\ SET \times TERM)$
$\times (CHAR\ LIST \times \mathbb{N})\ LIST$
$\times (CHAR\ LIST \times TYPE)\ LIST$
$\times CHAR\ LIST\ LIST)\ SET$
$\rightarrow ((({'}IP \times {'}UD\ PDS\_THM\ LIST) \times {'}UD\ PDS\_STATE)$
$\times TERM\ SET$
$\times TERM)\ SET$
$\rightarrow {'}IP \times {'}UD\ PDS\_THM\ ADDR\ LIST$
$\rightarrow {'}UD\ PDS\_STATE$
$\rightarrow {'}UD\ PDS\_STATE)$
$\rightarrow BOOL$

10.3 Types

${'}1$ **ADDR**
${'}1$ **THEORY_CONTENTS**
${'}1$ **THEORY_INFO**
${'}1$ **PDS_THM**
${'}1$ **PDS_STATE**

10.4 Type Abbreviations

${'}X$ **DICT** $(CHAR\ LIST \times {'}X)\ SET$
${'}X$ **STORE** $({'}X\ ADDR \times {'}X)\ SET$
STATUS $ONE + ONE + ONE + ONE$
${'}UD$ **HIERARCHY**
${'}UD\ THEORY\_INFO\ LIST$
$({'}IP, {'}UD)$ **SUBSYS_INPUT**
${'}IP \times {'}UD\ PDS\_THM\ LIST$
$({'}IP, {'}UD)$ **PDS_INPUT**
${'}IP \times {'}UD\ PDS\_THM\ ADDR\ LIST$
$({'}IP, {'}UD)$ **DEFINER**
$((({'}IP \times {'}UD\ PDS\_THM\ LIST) \times {'}UD\ PDS\_STATE)$
$\times (TERM\ SET \times TERM)$
$\times (CHAR\ LIST \times \mathbb{N})\ LIST$
$\times (CHAR\ LIST \times TYPE)\ LIST$

$\times$ *CHAR LIST LIST*) *SET*

('*IP*, '*UD*) **INFERRER**

$((('IP \times 'UD\ PDS\_THM\ LIST) \times 'UD\ PDS\_STATE)$
$\times\ TERM\ SET$
$\times\ TERM)\ SET$

('*IP*, '*UD*) **INTERPRETER**

$((('IP \times 'UD\ PDS\_THM\ LIST) \times 'UD\ PDS\_STATE)$
$\times\ (TERM\ SET \times TERM)$
$\times\ (CHAR\ LIST \times \mathbb{N})\ LIST$
$\times\ (CHAR\ LIST \times TYPE)\ LIST$
$\times\ CHAR\ LIST\ LIST)\ SET$
$\rightarrow ((('IP \times 'UD\ PDS\_THM\ LIST) \times 'UD\ PDS\_STATE)$
$\times\ TERM\ SET$
$\times\ TERM)\ SET$
$\rightarrow 'IP \times 'UD\ PDS\_THM\ ADDR\ LIST$
$\rightarrow 'UD\ PDS\_STATE$
$\rightarrow 'UD\ PDS\_STATE$

10.5 Fixity

Right Infix 300:

<−

10.6 Definitions

initial_dict $\vdash initial\_dict = \{\}$

enter $\vdash \forall\ key\ item\ dict$
$\bullet\ enter\ key\ item\ dict = dict \oplus \{(key, item)\}$

lookup $\vdash \forall\ key\ dict\ item$
$\bullet\ lookup\ key\ dict\ item \Leftrightarrow (key, item) \in dict$

delete $\vdash \forall\ key\ dict \bullet\ delete\ key\ dict = \{key\} \lhd dict$

block_delete $\vdash \forall\ a\ dict \bullet\ block\_delete\ a\ dict = dict \rhd a$

keys $\vdash keys = Dom$

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

<− $\vdash ConstSpec$
$(\lambda\ \$"<-'"$
$\bullet\ \forall\ addr\ value\ st$
$\bullet\ addr \in Dom\ st$
$\Rightarrow \$"<-'"\ addr\ value\ st$
$= st \oplus \{(addr, value)\})$
$\$<-$

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

new $\vdash \forall\ value\ st1\ st2\ addr$
$\bullet\ new\ value\ st1\ (st2, addr)$
$\Leftrightarrow \neg\ addr \in Dom\ st1 \wedge st2 = st1 \oplus \{(addr, value)\}$

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

tc_theorem_dict
tc_current_level
tc_deleted_levels
tc_user_data $\vdash \forall\ t\ x1\ x2\ x3\ x4\ x5\ x6\ x7\ x8\ x9\ x10$

$$
\begin{array}{l}
\bullet\ tc\_name \\
\quad (MkTHEORY\_CONTENTS \\
\qquad x1 \\ \qquad x2 \\ \qquad x3 \\ \qquad x4 \\ \qquad x5 \\ \qquad x6 \\ \qquad x7 \\ \qquad x8 \\ \qquad x9 \\ \qquad x10) \\
\quad = x1 \\
\land\ tc\_ty\_env \\
\quad (MkTHEORY\_CONTENTS \\
\qquad x1 \\ \qquad x2 \\ \qquad x3 \\ \qquad x4 \\ \qquad x5 \\ \qquad x6 \\ \qquad x7 \\ \qquad x8 \\ \qquad x9 \\ \qquad x10) \\
\quad = x2 \\
\land\ tc\_con\_env \\
\quad (MkTHEORY\_CONTENTS \\
\qquad x1 \\ \qquad x2 \\ \qquad x3 \\ \qquad x4 \\ \qquad x5 \\ \qquad x6 \\ \qquad x7 \\ \qquad x8 \\ \qquad x9 \\ \qquad x10) \\
\quad = x3 \\
\land\ tc\_parents \\
\quad (MkTHEORY\_CONTENTS \\
\qquad x1 \\ \qquad x2 \\ \qquad x3 \\ \qquad x4 \\ \qquad x5 \\ \qquad x6 \\ \qquad x7 \\ \qquad x8 \\ \qquad x9 \\ \qquad x10) \\
\quad = x4 \\
\land\ tc\_axiom\_dict \\
\quad (MkTHEORY\_CONTENTS
\end{array}
$$

$x1$
$x2$
$x3$
$x4$
$x5$
$x6$
$x7$
$x8$
$x9$
$x10)$
$= x5$
$\wedge\ tc\_definition\_dict$
$(MkTHEORY\_CONTENTS$
$x1$
$x2$
$x3$
$x4$
$x5$
$x6$
$x7$
$x8$
$x9$
$x10)$
$= x6$
$\wedge\ tc\_theorem\_dict$
$(MkTHEORY\_CONTENTS$
$x1$
$x2$
$x3$
$x4$
$x5$
$x6$
$x7$
$x8$
$x9$
$x10)$
$= x7$
$\wedge\ tc\_current\_level$
$(MkTHEORY\_CONTENTS$
$x1$
$x2$
$x3$
$x4$
$x5$
$x6$
$x7$
$x8$
$x9$
$x10)$
$= x8$
$\wedge\ tc\_deleted\_levels$
$(MkTHEORY\_CONTENTS$
$x1$
$x2$
$x3$
$x4$
$x5$
$x6$

$x7$
$x8$
$x9$
$x10)$
$= x9$
$\land\ tc\_user\_data$
$(MkTHEORY\_CONTENTS$
$x1$
$x2$
$x3$
$x4$
$x5$
$x6$
$x7$
$x8$
$x9$
$x10)$
$= x10$
$\land\ MkTHEORY\_CONTENTS$
$(tc\_name\ t)$
$(tc\_ty\_env\ t)$
$(tc\_con\_env\ t)$
$(tc\_parents\ t)$
$(tc\_axiom\_dict\ t)$
$(tc\_definition\_dict\ t)$
$(tc\_theorem\_dict\ t)$
$(tc\_current\_level\ t)$
$(tc\_deleted\_levels\ t)$
$(tc\_user\_data\ t)$
$= t$

TSNormal
TSLocked
TSAncestor
TSDeleted $\vdash ConstSpec$
$(\lambda\ (TSNormal', TSLocked', TSAncestor', TSDeleted')$
$\bullet\ [TSNormal'; TSLocked'; TSAncestor'; TSDeleted']$
$\in Distinct)$
$(TSNormal, TSLocked, TSAncestor, TSDeleted)$

THEORY_INFO $\vdash \exists\ f \bullet\ TypeDefn\ (\lambda\ x \bullet\ T)\ f$
MkTHEORY_INFO
ti_status
ti_inscope
ti_contents $\vdash \forall\ t\ x1\ x2\ x3$
$\bullet\ ti\_status\ (MkTHEORY\_INFO\ x1\ x2\ x3) = x1$
$\land\ (ti\_inscope\ (MkTHEORY\_INFO\ x1\ x2\ x3) \Leftrightarrow x2)$
$\land\ ti\_contents\ (MkTHEORY\_INFO\ x1\ x2\ x3) = x3$
$\land\ MkTHEORY\_INFO$
$(ti\_status\ t)$
$(ti\_inscope\ t)$
$(ti\_contents\ t)$
$= t$

PDS_THM $\vdash \exists\ f \bullet\ TypeDefn\ (\lambda\ x \bullet\ T)\ f$
MkPDS_THM
pt_theory
pt_level
pt_sequent $\vdash \forall\ t\ x1\ x2\ x3$
$\bullet\ pt\_theory\ (MkPDS\_THM\ x1\ x2\ x3) = x1$
$\land\ pt\_level\ (MkPDS\_THM\ x1\ x2\ x3) = x2$

```
                    ∧ pt_sequent (MkPDS_THM x1 x2 x3) = x3
                    ∧ MkPDS_THM
                        (pt_theory t)
                        (pt_level t)
                        (pt_sequent t)
                      = t
PDS_STATE         ⊢ ∃ f• TypeDefn (λ x• T) f
MkPDS_STATE
ps_current_theory
ps_current_hierarchy
ps_theory_store
ps_hierarchy_store
ps_theorem_store
                  ⊢ ∀ t x1 x2 x3 x4 x5
                    • ps_current_theory (MkPDS_STATE x1 x2 x3 x4 x5) = x1
                    ∧ ps_current_hierarchy
                        (MkPDS_STATE x1 x2 x3 x4 x5)
                      = x2
                    ∧ ps_theory_store (MkPDS_STATE x1 x2 x3 x4 x5)
                      = x3
                    ∧ ps_hierarchy_store (MkPDS_STATE x1 x2 x3 x4 x5)
                      = x4
                    ∧ ps_theorem_store (MkPDS_STATE x1 x2 x3 x4 x5)
                      = x5
                    ∧ MkPDS_STATE
                        (ps_current_theory t)
                        (ps_current_hierarchy t)
                        (ps_theory_store t)
                        (ps_hierarchy_store t)
                        (ps_theorem_store t)
                      = t
initial_theory
                  ⊢ ∀ ud
                    • initial_theory ud
                      = (let contents
                             = MkTHEORY_CONTENTS
                               "MIN"
                               initial_dict
                               initial_dict
                               []
                               initial_dict
                               initial_dict
                               initial_dict
                               0
                               {}
                               ud
                        in let (st, addr)
                               = (ϵ (st, addr)
                               • new contents initial_store (st, addr))
                          in (st, MkTHEORY_INFO TSNormal T addr))
initial_state
                  ⊢ ∀ ud
                    • initial_state ud
                      = (let (thy_st, thy_info) = initial_theory ud
                        in let (hier_st, hier_addr)
                               = (ϵ (st, addr)
                               • new
                                   [thy_info]
```

```
                              initial_store
                              (st, addr))
                         in MkPDS_STATE
                           (ti_contents thy_info)
                           hier_addr
                           thy_st
                           hier_st
                           initial_store)
```

theory_contents

```
⊢ ∀ state name thy_c
  • theory_contents state name thy_c
      ⇔ (let thy_st = ps_theory_store state
        in let hier_st = ps_hierarchy_store state
          in let cur_hier = ps_current_hierarchy state
            in let infos
                  = (ε x• fetch cur_hier hier_st x)
              in let thys
                    = Map
                      ((λ addr
                          • ε x• fetch addr thy_st x)
                          o ti_contents)
                      infos
                in ∃ thy
                • thy ∈ Elems thys
                    ∧ tc_name thy = name)
```

theory_names

```
⊢ ∀ state name
  • name ∈ theory_names state
      ⇔ (∃ thy_c• theory_contents state name thy_c)
```

theory_ancestors

```
⊢ ∀ state name
  • theory_ancestors state name
      = ⋂
        {P
          |(name ∈ theory_names state ⇒ name ∈ P)
              ∧ (∀ anc1 thy_c anc2
              • anc1 ∈ P
                    ∧ theory_contents state anc1 thy_c
                    ∧ anc2 ∈ Elems (tc_parents thy_c)
                ⇒ anc2 ∈ P)}
```

interpret_theory_contents

```
⊢ ∀ thy_cs
  • interpret_theory_contents thy_cs
      = (abs_theory
          ({(tyn, arity)
              |∃ thy_c lev
                • thy_c ∈ thy_cs
                    ∧ lookup
                      tyn
                      (tc_ty_env thy_c)
                      (arity, lev)},
            {(cn, ty)
              |∃ thy_c lev
                • thy_c ∈ thy_cs
                    ∧ lookup
                      cn
                      (tc_con_env thy_c)
                      (ty, lev)},
            {seq
```

$$
\begin{array}{l}
\qquad\qquad\qquad |\exists\ thy\_c\ thmn\ lev \\
\qquad\qquad\qquad\quad \bullet\ thy\_c \in thy\_cs \\
\qquad\qquad\qquad\qquad \land\ (lookup \\
\qquad\qquad\qquad\qquad\quad thmn \\
\qquad\qquad\qquad\qquad\quad (tc\_axiom\_dict\ thy\_c) \\
\qquad\qquad\qquad\qquad\quad (seq,\ lev) \\
\qquad\qquad\qquad\qquad\ \lor\ lookup \\
\qquad\qquad\qquad\qquad\quad thmn \\
\qquad\qquad\qquad\qquad\quad (tc\_definition\_dict\ thy\_c) \\
\qquad\qquad\qquad\qquad\quad (seq,\ lev))\}), \\
\qquad\qquad \{seq \\
\qquad\qquad\ |\exists\ thy\_c\ thmn\ lev \\
\qquad\qquad\quad \bullet\ thy\_c \in thy\_cs \\
\qquad\qquad\qquad \land\ lookup \\
\qquad\qquad\qquad\quad thmn \\
\qquad\qquad\qquad\quad (tc\_definition\_dict\ thy\_c) \\
\qquad\qquad\qquad\quad (seq,\ lev)\}, \\
\qquad\qquad \{seq \\
\qquad\qquad\ |\exists\ thy\_c\ thmn\ lev \\
\qquad\qquad\quad \bullet\ thy\_c \in thy\_cs \\
\qquad\qquad\qquad \land\ lookup \\
\qquad\qquad\qquad\quad thmn \\
\qquad\qquad\qquad\quad (tc\_theorem\_dict\ thy\_c) \\
\qquad\qquad\qquad\quad (seq,\ lev)\})
\end{array}
$$

interpret_state

$$
\begin{array}{l}
\vdash \forall\ state \\
\quad \bullet\ interpret\_state\ state \\
\qquad = mk\_theory\_hierarchy \\
\qquad\ (\lambda\ thyn \\
\qquad\quad \bullet\ interpret\_theory\_contents \\
\qquad\qquad \{tc \\
\qquad\qquad\ |\exists\ anc \\
\qquad\qquad\quad \bullet\ anc \in theory\_ancestors\ state\ thyn \\
\qquad\qquad\qquad \land\ theory\_contents\ state\ anc\ tc\})
\end{array}
$$

dest_state

$$
\begin{array}{l}
\vdash \forall\ state \\
\quad \bullet\ dest\_state\ state \\
\qquad = (ps\_current\_theory\ state, \\
\qquad\quad ps\_current\_hierarchy\ state, \\
\qquad\quad ps\_theory\_store\ state, \\
\qquad\quad ps\_hierarchy\_store\ state, \\
\qquad\quad ps\_theorem\_store\ state)
\end{array}
$$

dest_theory_contents

$$
\begin{array}{l}
\vdash \forall\ tc \\
\quad \bullet\ dest\_theory\_contents\ tc \\
\qquad = (tc\_name\ tc,\ tc\_ty\_env\ tc,\ tc\_con\_env\ tc, \\
\qquad\quad tc\_parents\ tc,\ tc\_axiom\_dict\ tc, \\
\qquad\quad tc\_definition\_dict\ tc,\ tc\_theorem\_dict\ tc, \\
\qquad\quad tc\_current\_level\ tc,\ tc\_deleted\_levels\ tc, \\
\qquad\quad tc\_user\_data\ tc)
\end{array}
$$

current_theory_contents

$$
\begin{array}{l}
\vdash \forall\ state \\
\quad \bullet\ current\_theory\_contents\ state \\
\qquad = (let\ (cur\_thy,\ \_1,\ thy\_st,\ \_2,\ \_3) \\
\qquad\qquad = dest\_state\ state \\
\qquad\quad in\ \epsilon\ tc \bullet\ fetch\ cur\_thy\ thy\_st\ tc)
\end{array}
$$

current_theory_name

$$
\begin{array}{l}
\vdash \forall\ state \\
\quad \bullet\ current\_theory\_name\ state
\end{array}
$$

$= tc\_name\ (current\_theory\_contents\ state)$

current_abstract_theory

$\vdash \forall\ state$
$\bullet\ current\_abstract\_theory\ state$
$= Fst$
$(interpret\_theory\_contents$
$\{tc$
$| \exists\ anc$
$\bullet\ anc$
$\in theory\_ancestors$
$state$
$(current\_theory\_name\ state)$
$\land theory\_contents\ state\ anc\ tc\})$

theory_info $\vdash \forall\ state\ name$
$\bullet\ theory\_info\ state\ name$
$= (let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ \_1)$
$= dest\_state\ state$
$in\ let\ hier = (\epsilon\ h \bullet\ fetch\ cur\_hier\ hier\_st\ h)$
$in\ \epsilon\ ti$
$\bullet\ tc\_name$
$(\epsilon\ tc$
$\bullet\ fetch\ (ti\_contents\ ti)\ thy\_st\ tc)$
$= name$
$\land \neg\ ti\_status\ ti = TSDeleted)$

current_theory_status

$\vdash \forall\ state$
$\bullet\ current\_theory\_status\ state$
$= ti\_status$
$(theory\_info\ state\ (current\_theory\_name\ state))$

check_thm $\vdash \forall\ state\ thm$
$\bullet\ check\_thm\ state\ thm$
$\Leftrightarrow (let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,\ \_1)$
$= dest\_state\ state$
$in\ let\ tc$
$= (\epsilon\ tc \bullet\ fetch\ (pt\_theory\ thm)\ thy\_st\ tc)$
$in\ let\ ti = theory\_info\ state\ (tc\_name\ tc)$
$in\ pt\_theory\ thm = ti\_contents\ ti$
$\land ti\_inscope\ ti$
$\land \neg\ pt\_level\ thm \in tc\_deleted\_levels\ tc)$

check_thm_address

$\vdash \forall\ state\ thm\_ad$
$\bullet\ check\_thm\_address\ state\ thm\_ad$
$\Leftrightarrow (let\ (\_1,\ \_2,\ \_3,\ \_4,\ thm\_st)$
$= dest\_state\ state$
$in\ \exists\ thm$
$\bullet\ fetch\ thm\_ad\ thm\_st\ thm$
$\land check\_thm\ state\ thm)$

fetch_thms $\vdash \forall\ state\ thm\_ads$
$\bullet\ fetch\_thms\ state\ thm\_ads$
$= (let\ (\_1,\ \_2,\ \_3,\ \_4,\ thm\_st)$
$= dest\_state\ state$
$in\ Map$
$(\lambda\ a \bullet\ \epsilon\ thm \bullet\ fetch\ a\ thm\_st\ thm)$
$thm\_ads)$

hierarchy_ancestor

$\vdash \forall\ state\ hier\_ad1\ hier\_ad2$
$\bullet\ hierarchy\_ancestor\ state\ hier\_ad1\ hier\_ad2$
$\Leftrightarrow (let\ (\_1,\ cur\_hier,\ \_2,\ hier\_st,\ \_3)$

$= dest\_state\ state$
$in\ \forall\ h1\ h2$
$\bullet\ fetch\ hier\_ad1\ hier\_st\ h1$
$\wedge\ fetch\ hier\_ad2\ hier\_st\ h2$
$\Rightarrow Elems\ (Map\ ti\_contents\ h1)$
$\subseteq Elems\ (Map\ ti\_contents\ h2))$

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

make_current $\vdash \forall\ thyn\ state$
$\bullet\ make\_current\ thyn\ state$
$= (let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,$
$thm\_st)$
$= dest\_state\ state$
$in\ let\ f1\ ti$
$= tc\_name$
$(\epsilon\ tc$
$\bullet\ fetch\ (ti\_contents\ ti)\ thy\_st\ tc)$
$in\ let\ f2\ ti$
$\Leftrightarrow f1\ ti \in theory\_ancestors\ state\ thyn$
$in\ let\ f3\ ti$
$= MkTHEORY\_INFO$
$(ti\_status\ ti)$
$(f2\ ti)$
$(ti\_contents\ ti)$
$in\ let\ hier'$
$= Map$
$f3$
$(\epsilon\ h\bullet\ fetch\ cur\_hier\ hier\_st\ h)$
$in\ let\ hier\_st'$
$= (cur\_hier <- hier')\ hier\_st$
$in\ let\ cur\_thy'$
$= ti\_contents$
$(theory\_info\ state\ thyn)$
$in\ MkPDS\_STATE$
cur_thy'
cur_hier
thy_st
$hier\_st'$
$thm\_st)$

freeze_hierarchy
$\vdash \forall\ state$
$\bullet\ freeze\_hierarchy\ state$
$= (let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,$
$thm\_st)$
$= dest\_state\ state$
$in\ let\ f1\ n$
$= (if\ n = TSDeleted$
$then\ n$
$else\ TSAncestor)$
$in\ let\ f2\ ti$
$= MkTHEORY\_INFO$
$(f1\ (ti\_status\ ti))$
$(ti\_inscope\ ti)$

```
                    (ti_contents ti)
              in let hier′
                    = Map
                      f2
                      (ε 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 = (ε 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′)
                  = (ε (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 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
```

```
                    ∨ ti_status (theory_info state thyn)
                      = TSDeleted
                  then state
                  else make_current thyn state)
empty_theory   ⊢ ∀ thyn pars ud
                 • empty_theory thyn pars ud
                     = MkTHEORY_CONTENTS
                       thyn
                       initial_dict
                       initial_dict
                       pars
                       initial_dict
                       initial_dict
                       initial_dict
                       0
                       {}
                       ud
arbitrary_ud   ⊢ T
delete_theory
               ⊢ ∀ thyn state
                 • delete_theory thyn state
                     = (if
                          ¬ thyn ∈ theory_names state
                            ∨ ¬ ti_status (theory_info state thyn)
                                = TSNormal
                            ∨ ti_inscope (theory_info state thyn)
                            ∨ (∃ 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 ti'
                                  = (if ti' = ti
                                    then
                                      MkTHEORY_INFO
                                        TSDeleted
                                        F
                                        (ti_contents ti)
                                    else ti)
                              in let hier'
                                    = Map
                                      f
                                      (ϵ h• fetch cur_hier hier_st h)
                                in let hier_st'
                                      = (cur_hier <− hier') hier_st
                                  in let thy
                                        = empty_theory
                                          thyn
                                          []
                                          arbitrary_ud
                                    in let thy_st'
                                          = (ti_contents ti <− thy)
                                            thy_st
                                      in MkPDS_STATE
```

```
                                    cur_thy
                                    cur_hier
                                    thy_st′
                                    hier_st′
                                    thm_st))
new_theory     ⊢ ∀ 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
                                         (ϵ 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_parent     ⊢ ∀ thyn state
                 • new_parent thyn state
                     = (if
                         ¬ thyn ∈ theory_names state
                           ∨ thyn
                             ∈ Elems
                               (tc_parents
                                   (current_theory_contents state))
                           ∨ (∃ ancn
                           • ancn
                                 ∈ theory_ancestors state thyn
                                   \ theory_ancestors
                                     state
                                     (current_theory_name state)
                               ∧ (let anc
                                     = (ϵ anc
                                     • theory_contents state ancn anc)
                                 in let cur_thy
                                       = current_abstract_theory state
                                   in (∃ ty nlev
                                     • ty ∈ Dom (types cur_thy)
                                         ∧ lookup
```

```
                    ty
                    (tc_ty_env anc)
                    nlev)
              ∨ (∃ con tylev
              • con ∈ Dom (constants cur_thy)
                  ∧ lookup
                    con
                    (tc_con_env anc)
                    tylev)))
then state
else
  (let (cur_thy, cur_hier, thy_st, hier_st,
          thm_st)
        = dest_state state
    in let cur_thyn = current_theory_name state
      in let f1 ti
            = tc_name
              (ε tc
                • fetch
                    (ti_contents ti)
                    thy_st
                    tc)
        in let f2 ti
              ⇔ f1 ti
                  ∈ theory_ancestors state thyn
                ∨ ti_inscope ti
          in let f3 ti
                = MkTHEORY_INFO
                  (ti_status ti)
                  (f2 ti)
                  (ti_contents ti)
            in let hier′
                  = Map
                    f3
                    (ε h
                      • fetch
                          cur_hier
                          hier_st
                          h)
              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 tc′
                          = MkTHEORY_CONTENTS
                            nm
                            t_e
                            c_e
                            (Cons thyn pars)
                            ax_d
                            def_d
                            thm_d
                            lev
                            x_levs
                            ud
```

```
                                    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

```
⊢ ∀ thyn copyn state
  • duplicate_theory thyn copyn state
    = (if
        ¬ thyn ∈ theory_names state
          ∨ copyn ∈ theory_names state
          ∨ thyn = "MIN"
      then state
      else
        (let (cur_thy, cur_hier, thy_st, hier_st,
                thm_st)
              = dest_state state
          in let tc
                = (ϵ tc
                • 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′
                    = 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)
                      = (ϵ (st, a)
                      • new tc′ thy_st (st, a))
                  in let ti
                        = MkTHEORY_INFO
                          TSNormal
                          F
                          addr
                    in let hier′
                          = Cons
                            ti
                            (ϵ 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))
```

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
                        (ϵ 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
```

```
                          TSNormal
                          (ti_inscope ti)
                          (ti_contents ti)
                      else ti)
                in let hier′
                      = Map
                        f
                        (ϵ 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′
```

```
                                hier_st
                                thm_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 = 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
                    = (ϵ lv• is_latest_level state lv)
                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'
$hier\_st$
$thm\_st))$

delete_thm $\vdash \forall\ key\ state$
$\bullet\ delete\_thm\ key\ state$
$= (let\ tc = current\_theory\_contents\ state$
$in\ if$
$\neg\ key \in keys\ (tc\_theorem\_dict\ tc)$
$\vee \neg\ current\_theory\_status\ state = TSNormal$
$then\ state$
$else$
$(let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,$
$thm\_st)$
$= dest\_state\ state$
$in\ let\ (nm,\ t\_e,\ c\_e,\ pars,\ ax\_d,\ def\_d,$
$thm\_d,\ lev,\ x\_levs,\ ud)$
$= dest\_theory\_contents\ tc$
$in\ let\ thm\_d' = delete\ key\ thm\_d$
$in\ let\ tc'$
$= MkTHEORY\_CONTENTS$
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'
$hier\_st$
$thm\_st))$

pds_new_axiom
$\vdash \forall\ tm\ key\ state$
$\bullet\ pds\_new\_axiom\ tm\ key\ state$
$= (let\ tc = current\_theory\_contents\ state$

```
in let seq = ({}, tm)
  in if
    key ∈ keys (tc_axiom_dict tc)
      ∨ ¬ seq
          ∈ sequents
            (current_abstract_theory state)
      ∨ ¬ 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)
                      = (ϵ 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

```
⊢ ∀ 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
                     (λ k
                       • 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)
                             = (ϵ sa
                             • new
```

$(pds\_mk\_thm\ state\ seq)$
thm_st
$sa)$
$in\ MkPDS\_STATE$
cur_thy
cur_hier
thy_st'
$hier\_st$
$thm\_st')))$

pds_new_type $\vdash \forall\ ty\ arity\ state$
$\bullet\ pds\_new\_type\ ty\ arity\ state$
$= (if$
ty
$\in Dom$
$(types\ (current\_abstract\_theory\ state))$
$then\ state$
$else$
$(let\ (cur\_thy,\ cur\_hier,\ thy\_st,\ hier\_st,$
$thm\_st)$
$= dest\_state\ state$
$in\ let\ tc = current\_theory\_contents\ state$
$in\ let\ (nm,\ t\_e,\ c\_e,\ pars,\ ax\_d,\ def\_d,$
$thm\_d,\ lev,\ x\_levs,\ ud)$
$= dest\_theory\_contents\ tc$
$in\ let\ lev' = lev + 1$
$in\ let\ t\_e'$
$= enter\ ty\ (arity,\ lev')\ t\_e$
$in\ let\ tc'$
$= MkTHEORY\_CONTENTS$
nm
t_e'
c_e
$pars$
ax_d
def_d
thm_d
lev'
x_levs
ud
$in\ let\ thy\_st'$
$= (cur\_thy <- tc')\ thy\_st$
$in\ MkPDS\_STATE$
cur_thy
cur_hier
thy_st'
$hier\_st$
$thm\_st))$

pds_new_constant
$\vdash \forall\ con\ ty\ state$
$\bullet\ pds\_new\_constant\ con\ ty\ state$
$= (if$
con
$\in Dom$
$(constants$
$(current\_abstract\_theory\ state))$
$\vee \neg\ ty$
$\in wf\_type$
$(types$

```
              (current_abstract_theory state))
        then state
        else
          (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 c_e′
                        = enter con (ty, lev′) 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 MkPDS_STATE
                          cur_thy
                          cur_hier
                          thy_st′
                          hier_st
                          thm_st))
```

make_inference

```
  ⊢ ∀ inferrer pars thm_ads state
    • make_inference inferrer ((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 inferrer
                  ∨ ¬ current_theory_status state
                      = TSNormal
              then state
              else
                (let seq
                      = inferrer @ ((pars, thms), state)
                  in let (cur_thy, cur_hier, thy_st,
                          hier_st, thm_st)
                        = dest_state state
                    in let (thm_st′, _1)
                          = (ε sa
                          • new
                              (pds_mk_thm state seq)
```

thm_st
$sa)$
$in\ MkPDS\_STATE$
cur_thy
cur_hier
thy_st
$hier\_st$
thm_st')))

allowed $\vdash \forall\ definer\ inferrer\ trans$
$\bullet\ allowed\ definer\ inferrer\ trans$
$\Leftrightarrow trans = freeze\_hierarchy$
$\lor\ trans = new\_hierarchy$
$\lor\ (\exists\ addr$
$\bullet\ trans = load\_hierarchy\ addr$
$\lor\ (\exists\ thyn$
$\bullet\ trans = open\_theory\ thyn$
$\lor\ (\exists\ thyn$
$\bullet\ trans = delete\_theory\ thyn$
$\lor\ (\exists\ thyn\ ud$
$\bullet\ trans = new\_theory\ thyn\ ud$
$\lor\ (\exists\ thyn$
$\bullet\ trans = new\_parent\ thyn$
$\lor\ (\exists\ thyn\ copyn$
$\bullet\ trans$
$= duplicate\_theory$
$thyn$
$copyn$
$\lor\ (\exists\ thyn$
$\bullet\ trans$
$= lock\_theory$
$thyn$
$\lor\ (\exists\ thyn$
$\bullet\ trans$
$= unlock\_theory$
$thyn$
$\lor\ (\exists\ key\ thm$
$\bullet\ trans$
$= save\_thm$
key
thm
$\lor\ trans = delete\_extension$
$\lor\ (\exists\ key$
$\bullet\ trans = delete\_thm\ key$
$\lor\ (\exists\ tm\ key$
$\bullet\ trans = pds\_new\_axiom\ tm\ key$
$\lor\ (\exists\ ty\ arity$
$\bullet\ trans = pds\_new\_type\ ty\ arity$
$\lor\ (\exists\ con\ ty$
$\bullet\ trans = pds\_new\_constant\ con\ ty$
$\lor\ (\exists\ pars\ thm\_ads$
$\bullet\ trans$
$= Curry$
$(make\_definition\ definer)$
$(pars,\ thm\_ads)$
$\lor\ (\exists\ pars\ thm\_ads$
$\bullet\ trans$
$= Curry$
$(make\_inference\ inferrer)$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad (pars,\\
\qquad\qquad\qquad\qquad\quad thm\_ads)))))))))))))))
\end{array}
$$

pds

$$
\begin{array}{l}
\vdash \forall\ definer\ inferrer\ interpreter\\
\quad \bullet\ pds\ definer\ inferrer\ interpreter\\
\qquad = ((\lambda\ ((pars,\ thm\_ads),\ state)\\
\qquad\quad \bullet\ (let\ state'\\
\qquad\qquad\quad = interpreter\\
\qquad\qquad\qquad definer\\
\qquad\qquad\qquad inferrer\\
\qquad\qquad\qquad (pars,\ thm\_ads)\\
\qquad\qquad\qquad state\\
\qquad\qquad in\ (state',\ ps\_theorem\_store\ state'))),\\
\qquad\quad interpret\_state)
\end{array}
$$

good_definer

$$
\begin{array}{l}
\vdash \forall\ definer\\
\quad \bullet\ good\_definer\ definer\\
\qquad \Leftrightarrow (\forall\ pars\ thms\ state\\
\qquad \bullet\ ((pars,\ thms),\ state) \in Dom\ definer\\
\qquad\quad \Rightarrow (let\ thy = current\_abstract\_theory\ state\\
\qquad\qquad in\ let\ (tyenv,\ conenv,\ axs)\\
\qquad\qquad\qquad = rep\_theory\ thy\\
\qquad\qquad\quad in\ let\ (seq,\ tys,\ cons,\ \_1)\\
\qquad\qquad\qquad\quad = definer\ @\ ((pars,\ thms),\ state)\\
\qquad\qquad\qquad in\ let\ tyenv'\\
\qquad\qquad\qquad\qquad = Fold\\
\qquad\qquad\qquad\qquad\quad (\lambda\ tn\ te \bullet\ te \cup \{tn\})\\
\qquad\qquad\qquad\qquad\quad tys\\
\qquad\qquad\qquad\qquad\quad tyenv\\
\qquad\qquad\qquad\quad in\ let\ conenv'\\
\qquad\qquad\qquad\qquad\quad = Fold\\
\qquad\qquad\qquad\qquad\qquad (\lambda\ st\ ce \bullet\ ce \cup \{st\})\\
\qquad\qquad\qquad\qquad\qquad cons\\
\qquad\qquad\qquad\qquad\qquad conenv\\
\qquad\qquad\qquad\qquad in\ let\ axs' = axs \cup \{seq\}\\
\qquad\qquad\qquad\qquad\quad in\ let\ thy'\\
\qquad\qquad\qquad\qquad\qquad\quad = abs\_theory\\
\qquad\qquad\qquad\qquad\qquad\qquad (tyenv',\ conenv',\ axs')\\
\qquad\qquad\qquad\qquad\qquad in\ thy\\
\qquad\qquad\qquad\qquad\qquad\quad \in definitional\_extension\\
\qquad\qquad\qquad\qquad\qquad\qquad thy'))
\end{array}
$$

good_inferrer

$$
\begin{array}{l}
\vdash \forall\ v\ inferrer\\
\quad \bullet\ good\_inferrer\ v\ inferrer\\
\qquad \Leftrightarrow (\forall\ pars\ thms\ state\\
\qquad \bullet\ ((pars,\ thms),\ state) \in Dom\ inferrer\\
\qquad\quad \Rightarrow (let\ thy = current\_abstract\_theory\ state\\
\qquad\qquad in\ let\ seq\\
\qquad\qquad\qquad = inferrer\ @\ ((pars,\ thms),\ state)\\
\qquad\qquad\quad in\ seq \in sequents\ thy\\
\qquad\qquad\qquad \wedge\ seq \in valid\ v\ thy))
\end{array}
$$

good_interpreter

$$
\begin{array}{l}
\vdash \forall\ interpreter\\
\quad \bullet\ good\_interpreter\ interpreter\\
\qquad \Leftrightarrow (\forall\ definer\ inferrer\ pars\ thm\_ads\\
\qquad \bullet\ allowed\\
\qquad\qquad definer\\
\qquad\qquad inferrer\\
\qquad\qquad (interpreter\\
\qquad\qquad\quad definer
\end{array}
$$

$inferrer$
$(pars,\ thm\_ads)))$

11 INDEX