HOL Formalised: Proof Development System

R.D. Arthan
Lemma 1 Ltd.
rda@lemma-one.com

25 October 1993
Revised 17 October 2002

Abstract

This is part of a suite of documents giving a formal specification of the HOL logic. An abstract model of a proof development system for HOL is defined and its critical properties are stated. Thus this document defines formal criteria for assessing a theorem-proving tool which purports to implement the HOL logic. These criteria cover both the inference and the definitional mechanisms provided by such a tool.

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

Reference: DS/FMU/IED/SPC004; issue 1.16

1 DOCUMENT CONTROL

1.1 Contents list

1.2 Document cross references

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

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

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

2 GENERAL

2.1 Scope

This document specifies some high level aspects of a proof development system for HOL. It is part of a suite of documents specifying HOL an overview of which may be found in [1].

2.2 Introduction

In [3] the rules of inference and definitional mechanisms which make up the HOL deductive system are defined. In this document we turn to specifying some high level aspects of the HOL proof development system. Many of the types and functions used to specify the logic may already be viewed as specifying corresponding parts of the implementation. We now wish to specify a generic type giving an abstract model of a HOL proof development system and to characterise the critical properties of such a system.

The designer of a proof development system is interested in ensuring that the theorems which the users compute are indeed derivable from the axioms of the theory involved. The LCF approach to this problem is to use a programming language supporting the abstract data type concept. Theorems are represented as elements of an abstract data type whose constructor functions consist precisely of the rules of inference (suitably parameterised so that they are partial functions rather than arbitrary relations).

In this way, the type system of the programming language helps to ensure that only valid theorems may be derived. In section 4 we define the action of the rules of inference on our type of theorems. These definitions can be viewed as a high level specification of the constructor functions of the abstract data type. Such functions are used to extend the set of theorems stored within a theory.

Finally, section 5 discusses an abstraction of the database side of the proof development system. The concept of a named hierarchy of theories is introduced and an abstraction of the state of an HOL proof development system is presented. Given this, we can define the generic type of an HOL system and so define general predicates on such systems expressing their critical properties. In producing a very high assurance implementation of HOL, this would give the starting point for some meaningful proof work: one might attempt to prove that these properties held for a high level design for a system defined in terms of the theorem proving mechanisms of section 4 and the (conservative) definitional mechanisms of [3]).

3 PREAMBLE

We introduce the new theory. Its parents are the theories *spc002* and *spc003* defined in [2] and [3].

SML

```
open_theory"spc002";
new_theory"spc004";
new_parent"spc003";
```

4 THE RULES OF INFERENCE REVISITED

In [1] the inference rules are defined as relations between sequents. We now wish to define inference rules as relations between theorems, since this gives a better model of what is done in a proof

development system. Thus, recalling that a theorem is represented by a pair (S, T) where T is a theory and S is a sequent, we must specify how the inference rules interact with the theory components. Essentially, we say that if, by rule $\mathbf{X}\_rule$, we may infer the sequent S from $S_1, S_2, \ldots$, then, by $\mathbf{X}$, we may infer (S, T) from $(S_1, T_1), (S_2, T_2), \ldots$ provided the theory T is an extension of each T_i.

Proof theoretically this is no different from a rule which insists that $T_i = T$ for all i. The more general formulation is meant to accord a little better with the thinking of the user of the proof development system and may allow more freedom in an implementation. An implementation need not exploit the full generality. For example, in the Cambridge HOL system all proofs are conducted in the context of a particular theory called the *current* theory. Thus the abstract data type representing theorems does not need a theory component and the current theory is, effectively, an implicit and unused parameter to the constructor functions of the abstract data type. The ProofPower system follows a different approach to storage of theories and does tag theorems with the theory to which they belong.

The definitions of the inference rules for theorems are derived directly and tediously from the corresponding rules for sequences. In each case, we simply change the sequent arguments to theorem arguments, and check that the corresponding rule for sequents holds for the sequent components of the theorems and that the theory component of the theorem inferred extends that of all the other theorem arguments.

HOL Constant

$$\mathbf{SUBST} : ((STRING \times TYPE) \to THM) \to TERM \to THM \to THM \to BOOL$$

$$\begin{array}{l}
\forall\ eqs\ tm\ old_thm\ new_thm \bullet \\
SUBST\ eqs\ tm\ old_thm\ new_thm = \\
let\ old_seq = thm_seq\ old_thm\ in\ let\ old_thy = thm_thy\ old_thm \\
in\ let\ new_seq = thm_seq\ new_thm\ in\ let\ new_thy = thm_thy\ new_thm \\
in \\
SUBST_rule\ (thm_seq\ o\ eqs)\ tm\ old_seq\ new_seq \wedge \\
new_thy\ extends\ old_thy \wedge \\
Ran(Graph(thm_thy\ o\ eqs)) \subseteq \{thy \mid new_thy\ extends\ thy\}
\end{array}$$

HOL Constant

$$\mathbf{ABS} : (STRING \times TYPE) \to THM \to THM \to BOOL$$

$$\begin{array}{l}
\forall\ vty\ old_thm\ new_thm \bullet \\
ABS\ vty\ old_thm\ new_thm = \\
let\ old_seq = thm_seq\ old_thm\ in\ let\ old_thy = thm_thy\ old_thm \\
in\ let\ new_seq = thm_seq\ new_thm\ in\ let\ new_thy = thm_thy\ new_thm \\
in \\
ABS_rule\ vty\ old_seq\ new_seq \wedge \\
new_thy\ extends\ old_thy
\end{array}$$

HOL Constant

INST_TYPE : $(STRING \to TYPE) \to THM \to THM \to BOOL$

$\forall\ tysubs\ old\_thm\ new\_thm \bullet$
$INST\_TYPE\ tysubs\ old\_thm\ new\_thm =$
$let\ old\_seq = thm\_seq\ old\_thm\ in\ let\ old\_thy = thm\_thy\ old\_thm$
$in\ let\ new\_seq = thm\_seq\ new\_thm\ in\ let\ new\_thy = thm\_thy\ new\_thm$
in
$INST\_TYPE\_rule\ tysubs\ old\_seq\ new\_seq \wedge$
$new\_thy\ extends\ old\_thy$

HOL Constant

DISCH : $TERM \to THM \to THM \to BOOL$

$\forall\ tm\ old\_thm\ new\_thm \bullet$
$DISCH\ tm\ old\_thm\ new\_thm =$
$let\ old\_seq = thm\_seq\ old\_thm\ in\ let\ old\_thy = thm\_thy\ old\_thm$
$in\ let\ new\_seq = thm\_seq\ new\_thm\ in\ let\ new\_thy = thm\_thy\ new\_thm$
in
$DISCH\_rule\ tm\ old\_seq\ new\_seq \wedge$
$new\_thy\ extends\ old\_thy$

HOL Constant

MP : $THM \to THM \to THM \to BOOL$

$\forall\ imp\_thm\ ant\_thm\ new\_thm \bullet$
$MP\ imp\_thm\ ant\_thm\ new\_thm =$
$let\ imp\_seq = thm\_seq\ imp\_thm\ in\ let\ imp\_thy = thm\_thy\ imp\_thm$
$in\ let\ ant\_seq = thm\_seq\ ant\_thm\ in\ let\ ant\_thy = thm\_thy\ ant\_thm$
$in\ let\ new\_seq = thm\_seq\ new\_thm\ in\ let\ new\_thy = thm\_thy\ new\_thm$
in
$MP\_rule\ imp\_seq\ ant\_seq\ new\_seq \wedge$
$new\_thy\ extends\ imp\_thy \wedge$
$new\_thy\ extends\ ant\_thy$

The axiom schemata for theorems are even more straightforward to define, since they hold in every normal theory.

HOL Constant

ASSUME : $TERM \to THM \to BOOL$

$\forall\ tm\ thm \bullet ASSUME\ tm\ thm = ASSUME\_axiom\ tm\ (thm\_seq\ thm)$

HOL Constant

REFL : $TERM \to THM \to BOOL$

$\forall\ tm\ thm \bullet REFL\ tm\ thm = ((thm\_seq\ thm) = REFL\_axiom\ tm)$

HOL Constant

BETA_CONV : $TERM \rightarrow THM \rightarrow BOOL$

$\forall\ tm\ thm \bullet BETA\_CONV\ tm\ thm = BETA\_CONV\_axiom\ tm\ (thm\_seq\ thm)$

5 THEORY HIERARCHIES AND THE HOL SYSTEM

HOL systems operate not with the totality of the type *THEORY*, but with subsets of it. The subsets are structured as hierarchies in which each theory has a name and a parenthood relation is given on the names. Moreover the representation of a theory provides for the storage of theorems which have been proved. Another important aspect is that the systems distinguish between definitional extensions[1] and axiomatic ones. We will view a theory hierarchy as a function from names to triples comprising a theory, a set of sequents giving the definitional axioms and a set of sequents giving the theorems which have been saved. We forget the parenthood relation since we can recover its ancestral from the extension relation on theories, and for our purposes the ancestral is all that is needed.

Our abstraction for an HOL system is a (total) function[2], which we think of as the state transition function for an automaton whose state can be interpreted as a theory hierarchy. We formulate three important properties of such functions:

(a)	a semantic property of the theorems in the system — do the transitions preserve truth? I.e. given an input state in the interpretation of which every theorem is true in every model of the corresponding theory, does the result state have the same property?
(b)	a syntactic property of the theorems in the system — do the transitions preserve derivability? I.e. given an input state in the interpretation of which every theorem is derivable from the axioms of the corresponding theory, does the output state have the same property?
(c)	a syntactic property of the theories in the system — are the definitional axioms in each theory in the interpretation of the output state of a transition the result of a definitional extension of some theory in the interpretation of the input state?

Neither *(a)* nor *(b)* prohibits arbitrary axiomatic extensions, this is the purpose of *(c)*. However *(c)* see says nothing about how the system treats the theorems in a theory hierarchy.

The reason for stating the three properties separately is to give a little latitude in the choice of a critical property to verify for an implementation. E.g. it would be appropriate to verify *(a)* and *(c)* for a system which was asserted to permit only the definitional extension mechanisms but which contained built-in inference rules for which it was felt that a semantic proof of correctness was significantly easier than a syntactic one.

(It is possible to give semantic analogues of *(c)*, but I am not sure how useful they are.)

[1] Recall that in the terminology introduced in [3] "definitional" extensions embrace the introduction of new types isomorphic to subsets of old and loose specification of new constants as well as simple definition by means of an equation.

[2] Arguably, a partial function or some similar, more realistic abstraction should be used.

5.1 Theory Hierarchies

In our abstraction of a proof development system a theory hierarchy is represented by the following type (we assume that the finite function in an implementation is interpreted as a total function by mapping unused names to some well-known theory (e.g. *MIN* or *INIT*):

SML

```
declare_type_abbrev("THEORY_HIERARCHY_REP", [],
                    ⌜:STRING → (THEORY × (SEQ SET) × (SEQ SET))⌝);
```

The second components of the triples in the above definition give the definitional axioms, the third components give the theorems which have been saved in a theory.

The theory hierarchies in a proof development system will be required to satisfy the following condition, which says that for each theory the corresponding definitional axioms are among its axioms and the corresponding stored theorems are well-formed with respect to its type and constant environment.

HOL Constant

```
        wf_hierarchy : THEORY_HIERARCHY_REP SET
─────────────────────────────────────────────────
        ∀hier•
        hier ∈ wf_hierarchy ⇔
                ∀thyn• Fst(Snd(hier thyn)) ⊆ axioms (Fst(hier thyn))
                ∧      Snd(Snd(hier thyn)) ⊆ sequents (Fst(hier thyn))
```

The critical part of a proof development system states will thus be interpreted as having the following type:

SML

```
type_spec {rep_fun="rep_theory_hierarchy", def_tm = ⌜
        THEORY_HIERARCHY ≃ mk_theory_hierarchy Of wf_hierarchy
⌝};
```

which we access with the following functions:

HOL Constant

```
        th_theory : THEORY_HIERARCHY → STRING → THEORY
─────────────────────────────────────────────────
        ∀ hier thyn•
        th_theory hier thyn = Fst((rep_theory_hierarchy hier) thyn)
```

HOL Constant

```
        th_theories : THEORY_HIERARCHY → (THEORY SET)
─────────────────────────────────────────────────
        ∀ hier thy•
        thy ∈ th_theories hier ⇔ ∃thyn•th_theory hier thyn = thy
```

HOL Constant

th_definitions : $THEORY\_HIERARCHY \rightarrow STRING \rightarrow (SEQ\ SET)$

$\forall\ hier\ thyn\ seq \bullet$
$\quad seq \in th\_definitions\ hier\ thyn$
$\Leftrightarrow \quad seq \in Fst(Snd((rep\_theory\_hierarchy\ hier)\ thyn))$

HOL Constant

th_theorems : $THEORY\_HIERARCHY \rightarrow STRING \rightarrow (SEQ\ SET)$

$\forall\ hier\ thyn\ seq \bullet$
$\quad seq \in th\_theorems\ hier\ thyn$
$\Leftrightarrow \quad seq \in Snd(Snd((rep\_theory\_hierarchy\ hier)\ thyn))$

5.2 Proof Development Systems

A HOL system may be interpreted as an instance of the following polymorphic type, in which the first component stands for the transition function for an automaton and the second for an interpretation of the states of the automaton as theory hierarchies.

SML

```
declare_type_abbrev("HOL_SYSTEM", ["'INPUT", "'OUTPUT", "'STATE"],
                    ⌜:  (('INPUT × 'STATE) → ('STATE × 'OUTPUT))
                     ×  ('STATE → THEORY_HIERARCHY)⌝);
```

The above definition does not capture any conditions on the "theorems" stored in the theories in the system other than their well-formedness as sequents. We can now consider the properties of such systems which we might wish to verify to give some confidence that the "theorems" really are theorems of the logic.

5.2.1 Property (a)

This is the condition based on the semantics. To describe it, let us say that a hierarchy is *valid* with respect to a type, $'U$, if all of its "theorems" are satisfied in any model of the theory with universe $'U$. We could then assert that we are interested in systems which preserve validity of hierarchies. Thus valid hierarchies for the type $'U$ are given by:

HOL Constant

valid_hierarchy : $'U \rightarrow THEORY\_HIERARCHY\ SET$

$\forall\ (v{:}'U)\ hier \bullet$
$hier \in valid\_hierarchy\ v \Leftrightarrow$
$\forall thyn \bullet\ th\_theorems\ hier\ thyn \subseteq valid\ v\ (th\_theory\ hier\ thyn)$

(Note that the apparently unused first parameter of *valid_hierarchy* ensures that $'U$ appears in the type of *valid_hierarchy*, as required to satisfy the restrictions on type variables imposed by *new_specification*.)

The validity preserving HOL systems for the type $'U$ are given by:

HOL Constant

validity_preserving : $'U \to ('INPUT, 'OUTPUT, 'STATE)HOL\_SYSTEM\ SET$

$\forall\ (v{:}'U)\ tr\_f\ int\bullet$
$(tr\_f,\ int) \in validity\_preserving\ v \Leftrightarrow$
$\forall input\ st\bullet$
$int\ st \in valid\_hierarchy\ v \Rightarrow int(Fst(tr\_f(input,\ st))) \in valid\_hierarchy\ v$

Thus the proposition for a given system *sys* would be the conjecture:

$?\vdash \forall(v{:}'U)\bullet validity\_preserving\ v\ sys$

I.e. property *(a)* is:

$\lambda sys{:}('INPUT, 'OUTPUT, 'STATE)HOL\_SYSTEM\bullet\forall(v{:}'U)\bullet validity\_preserving\ v\ sys$

Note that it is inadequate just to prove an instance of the above conjecture. E.g. since there are no models of any theory with universe a one-point type, the instance:

$?\vdash \forall(v{:}one)\bullet validity\_preserving\ v\ sys$

is trivially true for any v. However, using a model of a (provably) consistent theory such as *MIN*, it is possibly to construct systems which are not validity preserving.

(This is an example of the need for the restrictions imposed on type variables in the definition of *new_definition* and *new_specification* in [2]. The conjecture

$?\vdash xx = \lambda sys{:}('INPUT, 'OUTPUT, 'STATE)HOL\_SYSTEM\bullet\forall(v{:}'U)\bullet$
$validity\_preserving\ v\ sys$

is equal to $\lambda sys\bullet T$ at some instances of $'U$ but not at others, and so cannot be the defining theorem of a new constant xx).

5.3 Property (b)

This is the condition based on derivability. We say that a hierarchy is *derivable* if every theorem saved in it is derivable from the axioms of the corresponding theory. (One may think of the hierarchy as being derived from one with no saved theorems by a sequence of proof steps.)

HOL Constant

derivable_hierarchy : $THEORY\_HIERARCHY\ SET$

$\forall\ hier\bullet$
$hier \in derivable\_hierarchy \Leftrightarrow$
$\forall thyn\ seq\bullet\ seq \in th\_theorems\ hier\ thyn$
$\Rightarrow derivable\_from\ seq\ (axioms(th\_theory\ hier\ thyn))$

Property *(b)* is then the condition on systems that they preserve derivability of hierarchies:

HOL Constant

derivability_preserving : $('INPUT, 'OUTPUT, 'STATE)HOL\_SYSTEM\ SET$

$\forall\ tr\_f\ int\bullet$
$(tr\_f,\ int) \in derivability\_preserving \Leftrightarrow$
$\forall input\ st\bullet$
$int\ st \in derivable\_hierarchy \Rightarrow int(Fst(tr\_f(input,\ st))) \in derivable\_hierarchy$

5.4 Property (c)

We will say that a HOL system is *standard* if it only extends the definitional axioms in a theory hierarchy by means of the definitional extension mechanisms. Note that this does not prohibit axiomatic extensions, it just ensures that the definitions components in the theory hierarchy give a proper record of which extensions are definitional and which are axiomatic. We state this property as follows:

HOL Constant

standard : $('INPUT, 'OUTPUT, 'STATE)HOL\_SYSTEM\ SET$

$\forall\ tr\_f\ int\bullet$
$(tr\_f,\ int) \in standard \Leftrightarrow$
$\forall input\ st\ new\_thyn\bullet\exists old\_thyn\ thy\ tms\bullet$
$let\ old\_thy = th\_theory\ (int\ st)\ old\_thyn$
$in\ let\ old\_defs = th\_definitions\ (int\ st)\ old\_thyn$
$in\ let\ new\_thy = th\_theory(int(Fst(tr\_f(input,\ st))))new\_thyn$
$in\ let\ new\_defs = th\_definitions(int(Fst(tr\_f(input,\ st))))new\_thyn$
in
$\quad old\_thy \in definitional\_extension\ thy$
$\wedge\quad new\_axioms\ tms\ thy\ new\_thy$
$\wedge\quad new\_defs = old\_defs \cup (axioms\ thy \setminus axioms\ old\_thy)$

That is to say, the intermediate theory, *thy*, is a definitional extension of *old_thy*, *new_thy* is obtained from *thy* by adding axioms, and *new_defs* consists of *old_defs* together with the definitional axioms introduced by the extension of *old_thy* to *thy*.

6 INDEX OF DEFINED TERMS