HOL Formalised: Semantics

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

25th October 1993
Revised 8 March 2014

Abstract

This is part of a suite of documents giving a formal specification of the HOL logic. It gives the semantics of the HOL language by defining the notion of a model of an HOL theory.

The semantics is given by first formalising the notion of what we call a "universe", i.e., a model of set theory within HOL. Such a universe is a type provided with a binary relation which obeys the axioms expected of the set-theoretic membership operation. Polymorphism is used to present the notion of a model conservatively, rather than axiomatically. The definition of a model of an HOL theory within some universe is then given in the usual denotational style. The document concludes with a brief and informal discussion of the consistency and independence of the postulates for set theory as defined here.

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

Reference: DS/FMU/IED/SPC002; issue *Revision : 2.17*

1 DOCUMENT CONTROL

1.1 Contents list

1.2 Document cross references

[1] William S. Hatcher. *The Logical Foundations of Mathematics.* Pergamon, 1982.

[2] Kenneth Kunen. *Set Theory: An Introduction to Independence Proofs.* North Holland, 1980.

[3] J. Lambek and P.J. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge University Press, 1986.

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

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

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

2 GENERAL

2.1 Scope

This document specifies the semantics of the HOL language. It is part of a suite of documents constituting a formal specification of the HOL logic, an overview of which may be found in [4].

2.2 Introduction

A set-theoretic semantics devised by A. Pitts for HOL is given in [6]. The treatment given is not fully formal; however, it is "possible in principle to give a completely formal version within ZFC set theory". We wish to formalise the semantics in HOL.

A number of approaches have been considered to specifying the semantics of HOL within HOL itself. At one extreme, analogues of the axioms of ZFC can be introduced to give a theory in which we could hope to construct a model of HOL and use it to prove the consistency of all HOL theories which do not use axiomatic extensions. At the other extreme we might try a category-theoretic approach (see [3] or [1]), say by using definitional extensions to define the notion of an interpretation of a typed λ-calculus in a cartesian closed category and to specify how this notion applies to HOL. The main decisions to be made are:

- How general a class of structures do we wish to consider as models for HOL?
- Should we use axiomatic extensions?

Our present approach tries to be no more general than it needs to be. Since we have no immediate interest in completeness results or comparisons with other type theories, extra generality would probably only be an obstacle to assessing the correctness of our treatment and might well make it more difficult to prove the soundness of the inference rules. Thus we have opted for a set-theoretic treatment.

We do not wish to use axiomatic extensions. To avoid them we formulate the semantics as a specification of what a model should be, rather than as construction of a particular model. This makes little difference to the utility of the semantics since all it amounts to is that we make explicit our assumption of the existence of an appropriate universe of sets in which terms take their values.

2.3 Initialisation

SML

```
open_theory"spc001";
new_theory"spc002";
```

3 MODELS FOR SET THEORY IN HOL

3.1 Postulates for Elementary Set Theory

We wish to define a notion of a model for set theory within HOL. We will formulate this as a property of a membership relation, $mem :' U \to ' U \to bool$. The type $'U$ over which mem is defined corresponds to the universe in the treatment in [6].

In the following sections we define predicates which assist in defining our postulates for a set theory.

3.1.1 Extensionality

The predicate for extensionality is straightforward:

HOL Constant

$$\textbf{extensional} : ('U \to 'U \to BOOL) \to BOOL$$

$$\forall mem \bullet extensional\ mem \Leftrightarrow$$
$$\forall x\ y{:}'U \bullet x = y \Leftrightarrow \forall a{:}'U \bullet mem\ a\ x \Leftrightarrow mem\ a\ y$$

Note that we use the ordinary metalanguage equality as the equality relation in $'U$. Since the extensionality property implies that equality for sets is an equivalence relation, this imposes no loss of generality (since we could always work with the equivalence classes).

3.1.2 Separation

For us the postulate of separation will assert the existence of a subset operator assigning to each set x and each property P the subset of x in which P holds. Note that, since P is not constrained to be a property which can be expressed in first-order set theory, this postulate is, in general, stronger than that usual in first-order treatments of set theory.

HOL Constant

$$\textbf{is\_separation} : ('U \to 'U \to BOOL) \to ('U \to ('U \to BOOL) \to 'U) \to BOOL$$

$$\forall mem\ sub \bullet is\_separation\ mem\ sub \Leftrightarrow$$
$$\forall x\ P \bullet \forall a \bullet mem\ a\ (sub\ x\ P) \Leftrightarrow mem\ a\ x \wedge P\ a$$

Note that our universe is necessarily non-empty (since metalanguage types are non-empty), so extensionality and the existence of a subset operator with the above property will imply the existence of a unique set with no elements (namely $sub\, x\, (\lambda a.F)$ where x is any term of type $'U$).

3.1.3 Power Sets

This postulate will assert the existence of an operator assigning to each set the set of all its subsets.

HOL Constant

$$\textbf{is\_power} : ('U \to 'U \to BOOL) \to ('U \to 'U) \to BOOL$$

$$\forall mem\ power \bullet is\_power\ mem\ power \Leftrightarrow$$
$$\forall x \bullet \forall a \bullet mem\ a\ (power\ x) \Leftrightarrow \forall b \bullet mem\ b\ a \Rightarrow mem\ b\ x$$

If x is an element of $'U$ let us write $extent\ mem\ x$ for the metalanguage predicate $\lambda a \bullet mem\ a\ x$ and assume mem satisfies the postulates of separation and power sets, then, viewing metalanguage predicates as sets, $extent\ mem\ (power\ x)$ will be in 1-1 correspondence with $\lambda P \bullet P \subseteq extent\ mem\ x$. This will mean later that our semantics uses only the so-called standard models, in which both object language and metalanguage agree about the cardinality of object language function spaces.

3.1.4 Union Sets

This postulate asserts the existence of an operator assigning to each set the union of all its elements.

HOL Constant

is_union : $('U \to 'U \to BOOL) \to ('U \to 'U) \to BOOL$

$\forall mem\ union \bullet is\_union\ mem\ union \Leftrightarrow$
$\quad \forall x \bullet \forall a \bullet mem\ a\ (union\ x) \Leftrightarrow \exists b \bullet mem\ a\ b \wedge mem\ b\ x$

3.1.5 Unordered Pairs

This postulate asserts the existence of an operator assigning to any two sets a set whose elements are precisely those sets.

HOL Constant

is_pair : $('U \to 'U \to BOOL) \to ('U \to 'U \to 'U) \to BOOL$

$\forall mem\ pair \bullet is\_pair\ mem\ pair \Leftrightarrow$
$\quad \forall x\ y \bullet \forall a \bullet mem\ a\ (pair\ x\ y) \Leftrightarrow a = x \vee a = y$

3.2 Set Theories

We will say that a relation, mem, is a set theory if it admits a subset operator, a power set operator etc. satisfying the properties defined in the previous section.

HOL Constant

is_set_theory : $('U \to 'U \to BOOL) \to BOOL$

$\forall mem \bullet is\_set\_theory\ mem \Leftrightarrow$
$\quad extensional\ mem$
$\wedge \quad (\exists sub \bullet is\_separation\ mem\ sub)$
$\wedge \quad (\exists power \bullet is\_power\ mem\ power)$
$\wedge \quad (\exists union \bullet is\_union\ mem\ union)$
$\wedge \quad (\exists pair \bullet is\_pair\ mem\ pair)$

This notion of a set theory is quite a weak one. An example could be constructed without using axiomatic extensions by using a countably infinite type to model the set of hereditarily finite sets in a classical set theory (see appendix A below).

I would conjecture that one would only need to assert axiomatically the existence of a set theory (in the above sense) together with an axiom of infinity to give an extension of the HOL system strong enough to prove the consistency of the theory *INIT* and its definitional extensions.

3.3 Operations on Set Theories

It is pleasant in the sequel to use notations similar to the usual set-theoretic ones, at the very least by making membership an infix operator. Towards this end, we define polymorphic constants which will act as membership relation and subset, power set, union and pair operators in any type which permits a set theory. The names of these constants are given a subscript to distinguish them from constants used in the metalanguage.

3.3.1 Primitive Operations

First we fix a choice of membership relation, then we define the other operators with respect to that choice.

SML

```
declare_infix(1000, "∈s");
```

HOL Constant

$$\$\in_{\mathbf{s}} : 'U \to 'U \to BOOL$$

$$(\exists mem : 'U \to 'U \to BOOL \bullet is\_set\_theory\ mem) \Rightarrow is\_set\_theory\ (\$\in_s : 'U \to 'U \to BOOL)$$

HOL Constant

$$\mathbf{Sub_s} : ('U \to BOOL) \to 'U$$

$$(\exists mem : 'U \to 'U \to BOOL \bullet is\_set\_theory\ mem) \Rightarrow is\_separation\ (\$\in_s)\ (Sub_s : ('U \to BOOL) \to 'U)$$

HOL Constant

$$\mathbb{P}_{\mathbf{s}} : 'U \to 'U$$

$$(\exists mem : 'U \to 'U \to BOOL \bullet is\_set\_theory\ mem) \Rightarrow is\_power\ (\$\in_s)\ (\mathbb{P}_s : 'U \to 'U)$$

HOL Constant

$$\bigcup_{\mathbf{s}} : 'U \to 'U$$

$$(\exists mem : 'U \to 'U \to BOOL \bullet is\_set\_theory\ mem) \Rightarrow is\_union\ (\$\in_s)\ (\bigcup_s : 'U \to 'U)$$

We will use the symbol $\oplus_s$ for the infix operator which makes an unordered pair out of its operands:

SML

```
declare_infix(1000, "⊕s");
```

HOL Constant

$\$\oplus_{\mathbf{s}} \quad : 'U \to 'U \to 'U$

$(\exists mem{:}'U \to 'U \to BOOL \bullet is\_set\_theory\ mem) \Rightarrow$
$\quad is\_pair\ (\$\in_s)\ (\$\oplus_s{:}'U \to 'U \to 'U)$

3.3.2 Derived Operations

As already mentioned, since the type $'U$ cannot be empty we may construct an empty set using the subset operator. This set will be denoted $\varnothing_s$:

HOL Constant

$\varnothing_{\mathbf{s}} \quad : 'U$

$\varnothing_s = Sub_s\ (\epsilon x \bullet T)\ (\lambda a \bullet F)$

$Unit_s\ x$ will denote the singleton set $\{x\}$:

HOL Constant

$\mathbf{Unit_s} \ : 'U \to 'U$

$\forall x{:}'U \bullet\ Unit_s\ x = x \oplus_s x$

1_s denotes the set $\{\varnothing_s\}$:

HOL Constant

$\mathbf{1_s} \quad : 'U$

$1_s = Unit_s\ \varnothing_s$

2_s denotes the set $\{\varnothing_s, 1_s\}$:

HOL Constant

$\mathbf{2_s} \quad : 'U$

$2_s = \varnothing_s \oplus_s 1_s$

$x \mapsto_s y$ will denote the ordered pair with first component x and second component y, that is to say $\{\{x\}, \{x, y\}\}$:

SML

```
declare_infix(1000, "↦ₛ");
```

HOL Constant

$\$\mapsto_{\mathbf{s}} \quad : 'U \to 'U \to 'U$

$\forall x\ y{:}'U \bullet\ x \mapsto_s y = Unit_s\ x \oplus_s (x \oplus_s y)$

$\cup_s$ will be the infix binary union operator:

SML

$declare\_infix(1000,\ "\cup_s");$

HOL Constant

$$\$\cup_{\mathbf{s}} \quad : 'U \to 'U \to 'U$$

$$\forall x\ y{:}'U \bullet\ x \cup_s y = \bigcup_s (x \oplus_s y)$$

$\times_s$ will be the infix binary product operator. Since $\{\{x\}, \{x, y\}\} \subseteq \mathbb{P}(\mathbb{P}(x \cup y))$, this may be defined using the subset operator as follows.

SML

$declare\_infix(1000,\ "\times_s");$

HOL Constant

$$\$\times_{\mathbf{s}} \quad : 'U \to 'U \to 'U$$

$$\forall x\ y{:}'U \bullet\ x \times_s y = Sub_s\ (\mathbb{P}_s(\mathbb{P}_s(x \cup_s y)))\ (\lambda a \bullet \exists b\ c \bullet b \in_s x \wedge c \in_s y \wedge a = b \mapsto_s c)$$

For the infix relation-space operator we use the name $\leftrightarrow_s$):

SML

$declare\_infix(1000,\ "\leftrightarrow_s");$

HOL Constant

$$\$\leftrightarrow_{\mathbf{s}} \quad : 'U \to 'U \to 'U$$

$$\forall x\ y{:}'U \bullet\ x \leftrightarrow_s y = \mathbb{P}_s\ (x \times_s y)$$

$\to_s$ will be the infix total function-space operator:

SML

$declare\_infix(1000,\ "\to_s");$

HOL Constant

$$\$\to_{\mathbf{s}} \quad : 'U \to 'U \to 'U$$

$$\forall x\ y{:}'U \bullet\ x \to_s y = Sub_s\ (x \leftrightarrow_s y)\ (\lambda f \bullet \forall a \bullet (a \in_s x) \Rightarrow \exists_1 b \bullet (a \mapsto_s b) \in_s f)$$

The infix operator $@_s$ is used to denote application of a function to an argument. (Note that x here may not be a function and, even if it is, y may not be in its domain. These properties will have to be proved as necessary.)

SML

$declare\_infix(1000,\ "@_s");$

HOL Constant

$\$@_{\mathbf{s}} \quad : 'U \to 'U \to 'U$

$\forall x\ y{:}'U \bullet\ x\ @_s\ y = \epsilon a \bullet\ (y \mapsto_s a) \in_s x$

The following function is used to define the semantics of the boolean type constructor. $Bool_s$ is a function taking the empty list to 2_s and any other list to the empty set.

HOL Constant

$\mathbf{Bool_s} : 'U\ LIST \to 'U$

$(Bool_s\ [] = 2_s)$
$\wedge \quad (\forall x\ t \bullet\ Bool_s\ (Cons\ x\ t) = \varnothing_s)$

The following function is used to define the semantics of the function space type constructor. Fun_s is a function which takes a two element list, $[dom, rng]$, of non-empty elements of $'U$ to the set of functions from Dom to rng and which takes any other list to the empty set.

HOL Constant

$\mathbf{Fun_s} : 'U\ LIST \to 'U$

$\forall args \bullet Fun_s\ args =$
$\quad let\ dom = Hd\ args$
$\quad in\ let\ rng = Hd\ (Tl\ args)$
$\quad in \quad if\ args = [dom; rng] \wedge \neg dom = \varnothing_s \wedge \neg rng = \varnothing_s$
$\quad\quad then\ dom \to_s rng$
$\quad\quad else\ \varnothing_s$

The following function is used to define the semantics of λ-abstraction. Note that there is no guarantee that the result is a total function and this will have to be proved where necessary.

HOL Constant

$\mathbf{Abs_s} : ('U \to 'U) \to 'U \to 'U \to 'U$

$\forall f\ dom\ rng \bullet Abs_s\ f\ dom\ rng = Sub_s\ (dom \times_s rng)\ (\lambda x \bullet \exists a \bullet x = a \mapsto_s f\ a)$

The following function will represent the polymorphic equality of HOL:

HOL Constant

$\mathbf{EqRel_s} \quad : 'U \to 'U$

$\forall x{:}'U \bullet EqRel_s\ x = Sub_s\ (x \times_s x \times_s 2_s)$
$\quad\quad (\lambda a \bullet \exists b\ c \bullet a = (b \mapsto_s c \mapsto_s if\ b = c\ then\ 1_s\ else\ \varnothing_s))$

The following set will represent the graph of the implication function:

HOL Constant

$$\mathbf{ImpRel_s} \quad : 'U$$

$$ImpRel_s =$$
$$\quad Unit_s(1_s \mapsto_s 1_s \mapsto_s 1_s)$$
$$\cup_s \quad Unit_s(1_s \mapsto_s \varnothing_s \mapsto_s \varnothing_s)$$
$$\cup_s \quad Unit_s(\varnothing_s \mapsto_s 1_s \mapsto_s 1_s)$$
$$\cup_s \quad Unit_s(\varnothing_s \mapsto_s \varnothing_s \mapsto_s 1_s)$$

4 SEMANTICS FOR HOL

In this section we give the semantics of HOL with respect to some universe $'U$ by specifying the notion of a model of a theory. The main work in doing this is defining the value of a term with respect to a function assigning values to the variables and constants in it (and similarly for types).

Because of our use of HOL rather than set theory as a metalanguage, the approach and hence the terminology used here is rather different from that used in the HOL manual, [6]. An approximate translation is given in the following table:

This Document	HOL Manual
(Well-formed) type constant assignment	(Standard) model of a type structure
Type variable assignment	Element of $\mathcal{U}^n$
Polymorphic value	Function from $\mathcal{U}^n$ to $\mathcal{U}$
(Well-formed) constant assignment	Second component of a (standard) model of a signature
(Well-formed) interpretation	(Standard) model of a signature
Variable assignment	Element of $\mathcal{U}^n$

4.1 Semantics of Types

4.1.1 Type Constant Assignment

The value we assign to a type constant is in effect a function from n-tuples of non-empty elements of the universe $'U$ to non-empty elements of $'U$, where n is the arity of the type constant. Since we do not have dependent types we represent such a thing as a function of type $('U\ list \rightarrow * U)$ which sends a list of length other than n to the empty set. We make the following type abbreviation[1] to facilitate this:

SML

$$declare\_type\_abbrev(\texttt{"}TY\_CON\_ASSIGNMENT\texttt{"}, [\texttt{"}'U\texttt{"}], \ulcorner{:}STRING \rightarrow ('U\ LIST \rightarrow 'U)\urcorner);$$

We say that a type constant assignment is well-formed with respect to a type environment if the arities of the values assigned to the type constant names agree with those in the type environment and if the boolean and function space types are assigned appropriately.

HOL Constant

$$\begin{array}{l}
\quad \textbf{is_wf_ty_con_assignment}: TY_ENV \rightarrow 'U\ TY_CON_ASSIGNMENT \rightarrow BOOL \\
\hline
\quad \forall tyenv\ tyconass \bullet\ is_wf_ty_con_assignment\ tyenv\ tyconass \Leftrightarrow \\
\qquad (\forall tycon\ args \bullet \\
\qquad\quad (\neg tyconass\ tycon\ args = \varnothing_s) \\
\qquad \Leftrightarrow \quad (tycon \mapsto Length\ args \in tyenv \wedge \forall a \bullet a \in Elems\ args \Rightarrow \neg a = \varnothing_s)) \\
\quad \wedge \quad tyconass\ \texttt{"}bool\texttt{"} = Bool_s \\
\quad \wedge \quad tyconass\ \texttt{"}\rightarrow\texttt{"} = Fun_s
\end{array}$$

[1] ProofPower-HOL does not provide special syntax for type abbreviations. To introduce a type abbreviation one uses the ML function *declare_type_abbrev* with parameters indicating the name, formal parameters and definition of the type abbreviation.

4.1.2 Type Variable Assignments

A type variable assignment is just a total function from type variables to the universe, $'U$. We make the following type abbreviation for this notion.

SML

$declare\_type\_abbrev("TY\_VAR\_ASSIGNMENT", ["'U"], \ulcorner{:}STRING \to 'U\urcorner);$

A type variable assignment will be well-formed if all the values in its range are non-empty

HOL Constant

$$is\_wf\_ty\_var\_assignment : 'U\ TY\_VAR\_ASSIGNMENT \to BOOL$$

$$\forall tyvarass \bullet is\_wf\_ty\_var\_assignment\ tyvarass \Leftrightarrow \forall tyv \bullet \neg tyvarass\ tyv = \varnothing_s$$

4.1.3 Polymorphic Values

Types and terms have polymorphic values. We represent these as functions from type variable assignments to $'U$. This corresponds to the use of certain dependent products in the treatment in [6].

SML

$declare\_type\_abbrev("POLY\_VALUE", ["'U"], \ulcorner{:}'U\ TY\_VAR\_ASSIGNMENT \to 'U\urcorner);$

A set X of type variables supports a polymorphic value v if v is independent of the values assigned to type variables not in X.

SML

$declare\_infix(1000, "supports");$

HOL Constant

$$\$\textbf{supports}: STRING\ SET \to 'U\ POLY\_VALUE \to BOOL$$

$$\forall X\ v \bullet X\ supports\ v \Leftrightarrow (\forall a1\ a2 \bullet (\forall x \bullet x \in X \Rightarrow a1\ x = a2\ x) \Rightarrow v\ a1 = v\ a2)$$

4.1.4 Semantics of Types

The value of a type with respect to a type constant assignment is a function mapping type variable assignments onto polymorphic values:

HOL Constant

$$\textbf{type\_value} : 'U\ TY\_CON\_ASSIGNMENT \to TYPE \to 'U\ POLY\_VALUE$$

$$\forall tyconass\ s\ tyl \bullet$$
$$type\_value\ tyconass\ (mk\_var\_type\ s) = (\lambda tyvarass \bullet tyvarass\ s)$$
$$\wedge\quad type\_value\ tyconass\ (mk\_type(s, tyl)) =$$
$$(\lambda tyvarass \bullet tyconass\ s\ (Map(\lambda ty \bullet type\_value\ tyconass\ ty\ tyvarass)\ tyl))$$

4.2 Semantics of Terms

4.2.1 Constant Assignments

A constant assignment is a function assigning to each constant name a polymorphic value.

SML

```
declare_type_abbrev("CON_ASSIGNMENT", ["'U"], ⌜:(STRING → 'U POLY_VALUE)⌝);
```

We call a constant assignment well-formed with respect to a constant environment if it sends implication and equality to appropriate polymorphic values and if the set of type variables in the type associated with each constant in the environment supports the polymorphic value associated with the constant.

HOL Constant

```
        is_wf_con_assignment: CON_ENV → 'U CON_ASSIGNMENT → BOOL
├──────────────────────────────────────────────────────────────
        ∀conenv conass• is_wf_con_assignment conenv conass ⇔
                (∀tyvarass•
                        conass "=" tyvarass = EqRel_s (tyvarass "*")
                ∧       conass "⇒" tyvarass = ImpRel_s)
        ∧       (∀con ty• con ↦ ty ∈ conenv ⇒ type_ty_vars ty supports (conass con))
```

Here we rely on the fact that in the theory *MIN* equality is defined to have type $*\to *\to bool$.

4.2.2 Interpretations

An interpretation is a pair consisting of a type constant assignment and a constant assignment:

SML

```
declare_type_abbrev("INTERPRETATION", ["'U"],
                ⌜:'U TY_CON_ASSIGNMENT × 'U CON_ASSIGNMENT⌝);
```

An interpretation is considered to be well-formed with respect to a theory if the type constant assignment is well-formed with respect to the type environment of the theory and if the constant assignment is well-formed and respects the type constant assignment in an appropriate sense:

HOL Constant

```
        is_wf_interpretation        : THEORY → 'U INTERPRETATION → BOOL
├──────────────────────────────────────────────────────────────
        ∀thy tyconass conass•
                is_wf_interpretation thy (tyconass, conass)
        ⇔ (     is_wf_ty_con_assignment (types thy) tyconass
          ∧     is_wf_con_assignment (constants thy) conass
          ∧     ∀s ty• s ↦ ty ∈ constants thy ⇒
                        ∀tyvarass•is_wf_ty_var_assignment tyvarass ⇒
                                conass s tyvarass ∈_s type_value tyconass ty tyvarass)
```

(Here *constants thy s ty* is the assertion that a constant named s with type ty has been defined in the theory *thy*.)

4.2.3 Variable Assignments

A variable assignment is a function sending name-type pairs to polymorphic values.

SML

```
declare_type_abbrev("VAR_ASSIGNMENT", ["'U"],
                    ⌜:(STRING × TYPE) → 'U POLY_VALUE⌝);
```

A variable assignment is considered to be well-formed with respect to a type constant assignment if the following condition holds:

HOL Constant

```
        is_wf_var_assignment : 'U TY_CON_ASSIGNMENT →
                               'U VAR_ASSIGNMENT → BOOL
├──────────────────────────────────────────────────────
        ∀tyconass varass•
                is_wf_var_assignment tyconass varass
        ⇔       ∀s ty tyvarass•is_wf_ty_var_assignment tyvarass ⇒
                        varass (s, ty) tyvarass ∈s type_value tyconass ty tyvarass
```

4.2.4 Instantiation of Polymorphic Values

Given a theory, the name of a constant and the type of an instance of a constant, the declared type may be matched with the instance type and the result used to instantiate the polymorphic value assigned to the constant appropriately. The function *instance* that does this is only used in contexts where the constant has an assigned value and the instance type is an instance of the declared type.

HOL Constant

```
        instance : THEORY → 'U INTERPRETATION →
                           STRING × TYPE →'U POLY_VALUE
├──────────────────────────────────────────────────────
        ∀thy tyconass conass s instty f declty•
                is_wf_con_assignment (constants thy) conass ⇒
                s ↦ declty ∈ constants thy ∧ inst_type f declty = instty ⇒
                        instance thy (tyconass, conass) (s, instty)
                =       (λtyvarass• conass s (λtyvar• type_value tyconass (f tyvar) tyvarass))
```

I am indebted to Ramana Kumar for pointing out that instantiation of polymorphic values was not addressed in earlier versions of this document.

4.2.5 Semantics of Terms

The value of a term with respect to a type assignment and a constant assignment is a function mapping variable assignments to polymorphic values. If the term is a variable then its value is given by the variable assignment. If the term is a constant then the value is given by instantiating the value give by the constant assignment according as determined by its declared type and the type of this instance. If it is a combination, $f\ a$, then the application operator $@_s$ is used to apply the value of f to that of a. If the term is an abstraction, $\lambda v \bullet b$, then we form a metalanguage λ-abstraction

which sends an element, x, of the universe to the value taken by b when v takes the polymorphic value which is identically x. We then use Abs_s to construct a set in $'U$ from this metalanguage λ-abstraction.

HOL Constant

```
term_value : THEORY → 'U INTERPRETATION →
             TERM → 'U VAR_ASSIGNMENT → 'U POLY_VALUE
```

```
∀thy tyconass conass tm varass sty f a v b•
      (term_value thy (tyconass, conass) (mk_var sty) varass
      =     (λtyvarass•varass sty tyvarass))
∧     (term_value thy (tyconass, conass)(mk_const sty) varass
      =     instance thy (tyconass, conass) sty)
∧     (has_mk_comb(f, a)tm ⇒
            term_value thy (tyconass, conass) tm varass
      =     let vf = term_value thy (tyconass, conass) f
            in let va = term_value thy (tyconass, conass) a
            in (λtyvarass•vf varass tyvarass @_s va varass tyvarass))
∧     (has_mk_abs(v, b)tm ∧ mk_var sty = v ⇒
            term_value thy (tyconass, conass) tm varass
      =     λtyvarass•
            let dom = type_value tyconass (type_of_term v) tyvarass
            in let rng = type_value tyconass (type_of_term b) tyvarass
            in let fnc = (λx:'U•
                  term_value thy
                  (tyconass, conass) b
                  (λsty'•if sty' = sty then λtyvarass'•x else varass sty')
                  tyvarass)
            in Abs_s fnc dom rng)
```

4.3 Models of Theories

4.3.1 Satisfaction

Using *term_value* and interpreting boolean terms which evaluate to $\lambda x \bullet 1_s$ as true, we can define the notion of a sequence being satisfied by an interpretation. We define this notion as an infix relation between interpretations and sequents.

SML

```
declare_infix(1000, "satisfies");
```

HOL Constant

$satisfies : $THEORY \times {'U}\ INTERPRETATION \to SEQ \to BOOL$

$\forall thy\ int\ seq \bullet (thy, int)\ satisfies\ seq \Leftrightarrow$
$\quad \forall varass\ tyvarass \bullet$
$\quad let\ val\_of = \lambda tm \bullet term\_value\ thy\ int\ tm\ varass\ tyvarass$
$\quad and\ (tyconass, \_) = int$
$\quad in \quad is\_wf\_ty\_var\_assignment\ tyvarass$
$\quad \wedge \quad is\_wf\_var\_assignment\ tyconass\ varass$
$\quad \wedge \quad (\forall asm \bullet asm \in hyp\ seq \Rightarrow val\_of\ asm = 1_s)$
$\quad \Rightarrow \quad val\_of\ (concl\ seq) = 1_s$

4.3.2 Models

A model for a theory is an interpretation which is well-formed with respect to the theory and which satisfies all of its axioms:

HOL Constant

is_model : $THEORY \to {'U}\ INTERPRETATION\ SET$

$\forall thy\ int \bullet$
$\quad int \in is\_model\ thy \Leftrightarrow$
$\quad (is\_wf\_interpretation\ thy\ int \wedge$
$\quad \forall seq \bullet seq \in axioms\ thy \Rightarrow (thy, int)\ satisfies\ seq)$

4.3.3 Validity

A sequent is valid with respect to a theory if it is satisfied by any model of that theory. Because of the restriction that any type variable in the defining property of a constant must also appear in the type of the constant, we have to give *valid* an additional apparently unused parameter. (The reasons for this are further discussed in the description of *new_specification* in [5].)

HOL Constant

valid : ${'U} \to THEORY \to SEQ\ SET$

$\forall v\ thy\ seq \bullet$
$\quad seq \in valid\ v\ thy \Leftrightarrow$
$\quad \forall int{:}{'U}\ INTERPRETATION \bullet int \in is\_model\ thy \Rightarrow (thy, int)\ satisfies\ seq$

A CONSISTENCY AND INDEPENDENCE

In this appendix we briefly discuss the consistency and mutual independence of the postulates for a set theory given in section 3.2 above. We make use of informal set-theoretic notions.

My original arguments for the independence of union and pairing were incorrect. I am much indebted to Prof. Robert M. Solovay for pointing this out and supplying the arguments given below.

A.1 Consistency

Since we have not included an axiom of infinity, a model for the postulates may be constructed using a model of the hereditarily finite sets[2] in classical set theory.

It is easy to see that the hereditarily finite sets would supply a model for our postulates if we could represent them as an HOL type and define the membership relation for them in terms of that representation. This is straightforward: e.g. take the type, $\mathbb{N}$, of natural numbers under the relation, *mem*, informally described by:

Informal Discussion

> $mem\ i\ j \Leftrightarrow$ *the* $i-th$ *coefficient in the binary expansion of* j *is 1*

Thus our definition of set theory is consistent in the sense that some instance of the predicate *is_set_theory* is satisfiable.

Note that if one wishes to introduce an axiom of infinity and attempt to construct a model for HOL on that basis, then one can presumably construct a model just using the separation and power set axioms. This should follow from the fact that the cumulative hierarchy in `ZF` is constructed using only the power set and union operators, since a model for HOL only requires sets formed before stage $\omega + \omega$ and the axiom of infinity gets one beyond stage ω immediately.

A.2 Independence

The 5 postulates section 3.2 are independent. For example, we can construct a model that satisfies all the postulates except extensionality, so that extensionality is not a logical consequence of the other postulates. The following paragraphs sketch such a construction for each postulate in turn:

Extensionality To see that extensionality is independent of the other postulates, we adjoin a new empty individual, X say, to a model of all the postulates. Of course, this actually introduces an infinite family of sets constructed using X and the various set forming operations. The resulting system of sets satisfies all of the postulates except extensionality.

Separation The identity relation on a one-element set supplies a model of all of the postulates apart from separation.

Power Sets The hereditarily countable sets supply a model of all of the postulates apart from the existence of power sets (see [2] for more information).

[2] A set, A, is hereditarily finite if its transitive closure is finite. Here the transitive closure $tr\_cl\ A$, of A is defined by $tr\_cl\ A = A \cup (\bigcup A) \cup (\bigcup(\bigcup A)) \cup (\bigcup(\bigcup(\bigcup A))) \cup \ldots$. Less formally, a set is hereditarily finite if it is finite, all its elements are finite, all their elements are finite and so on.

Union Sets In ZFC, if $\kappa = |X|$ is the cardinal of a set X, we write, as usual, 2^κ for the cardinal of the powerset of X. Define β_x for $x \in \omega \cup \{\omega\}$, by

$$\begin{aligned} \beta_0 &= |\omega| \\ \beta_{n+1} &= 2^{\beta_n}, n \in \omega \\ \beta_\omega &= \mathsf{sup}_{n\in\omega}|\beta_n| \end{aligned}$$

One may verify that the set M of all sets which are hereditarily of cardinality less than β_ω satisfies all the postulates except the existence of unions. If one takes $U = \{\beta_n | n \in \omega\}$, then certainly $U \in M$, but $|\bigcup U| = \beta_\omega \notin M$, so M is not closed under unions.

Pairs To see the independence of the pairing postulate, consider a model of the system of axioms derived from ZFC by replacing the axiom of foundation by an axiom asserting the existence of two sets A and B satisfying $\{a\} = a$ and $\{b\} = b$ and such that any descending chain with respect to the membership relation stabilises at either a or b. I.e., if $x_1, x_2, \ldots$ is such that $x_{i+1} \in x_i$ for all i, then there is a j such that either $\forall i > j \bullet x_i = a$ or $\forall i > j \bullet x_i = b$

Now define take M to comprise all sets X such that at least one of a and b does not belong to the transitive closure of X. One may check that M satisfies all the postulates except pairing. However, both a and b are in M, but the pair $\{a, b\}$ is not, so M is not closed under formation of pairs.

B INDEX OF DEFINED TERMS