HOL Formalised:
Deductive 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. It defines the primitive inference rules, including conservative extension mechanisms. Related notions such as derivability are also defined.

The treatment of the HOL deductive system formally defined here is closely based on the semi-formal treatment in the documentation for the Cambridge HOL system.

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

1.1 Contents list

1 DOCUMENT CONTROL 1
1.1 Contents list ................................................................. 1
1.2 Document cross references .................................................... 2

2 GENERAL 3
2.1 Scope .................................................................................. 3
2.2 Introduction ........................................................................ 3

3 PREAMBLE 3

4 THE RULES OF INFERENCE 3
4.1 Free Variables ................................................................. 4
4.2 Object Language Constructs .................................................. 4
4.3 Substitution of Equals ........................................................... 5
4.3.1 Substitution .................................................................... 6
4.3.2 α-conversion .................................................................. 7
4.3.3 The Inference Rule SUBST .................................................. 7
4.4 Abstraction: ABS ............................................................... 8
4.5 Type Instantiation ............................................................... 8
4.5.1 Instantiation of Terms ....................................................... 9
4.5.2 The Inference Rule INST_TYPE ........................................... 11
4.6 Discharging an Assumption: DISCH ........................................ 12
4.7 Modus Ponens: MP ............................................................. 13

5 THE AXIOM SCHEMATA 13
5.1 The Axiom Schema ASSUME .................................................. 13
5.2 The Axiom Schema REFL ....................................................... 13
5.3 The Axiom Schema BETA_CONV ............................................ 14

6 DERIVABILITY 14

7 NORMAL THEORIES 15
7.1 Object Language Constructs .................................................. 15
7.2 Normal Theories ............................................................... 16

8 THEOREMS 17

9 CONSISTENCY AND CONSERVATIVE EXTENSION 18

10 DEFINITIONAL EXTENSIONS 19
10.1 Object Language Constructs .................................................. 19
10.1.1 Truth .......................................................................... 20
10.1.2 Universal Quantification ................................................... 20
10.1.3 Existential Quantification .................................................. 21
10.1.4 Falsity .......................................................................... 21
10.1.5 Negation ........................................................................ 22
10.1.6 Conjunction .................................................................. 22
10.1.7 Disjunction ................................................................... 23
10.1.8 ONE_ONE .................................................................... 24
11 THE THEORY INIT
  11.1 The Axioms ......................................................... 31
    11.1.1 BOOL_CASES_AX ............................................. 31
    11.1.2 IMP_ANTISYM_AX ........................................... 31
    11.1.3 ETA_AX ....................................................... 31
    11.1.4 SELECT_AX .................................................. 32
    11.1.5 INFINITY_AX ............................................... 32
  11.2 The Theory ....................................................... 32
  11.3 DEFINITIONAL EXTENSIONS ...................................... 32

12 INDEX OF DEFINED TERMS ........................................... 34

1.2 Document cross references


2 GENERAL

2.1 Scope

This document specifies the HOL deductive system. Some high level aspects of the implementation of the proof development system are also discussed. It is part of a suite of documents specifying the HOL logic, an overview of which may be found in [1].

2.2 Introduction

In [1] a brief theoretical discussion of the definition of deductive systems is given. In this document we fill in the details for HOL.

The first task is to define the rules of inference. HOL has five rules of inference: \textit{ABS}, \textit{DISCH}, \textit{INST\_TYPE}, \textit{MP}, \textit{SUBST} (defined in section 4 below) and three axiom schemata: \textit{ASSUME}, \textit{BETA\_CONV} and \textit{REFL} (defined in section 5). We follow [3] in treating the axiom schemata just like unary rules of inference. Such rules are a convenient home for infinite families of axioms that we wish to have in every theory.

With the rules of inference in hand, we define derivability in section 6. We then define the type of theorems of HOL as those pairs $(s, T)$ where $T$ is a theory and $s$ is a sequent in the language of $T$ derivable from the axioms of $T$.

Section 9, defines the type of all theorems and specifies the notions of consistency and conservative extension.

Mechanisms for extending theories by making definitions are of great practical importance, particularly those which preserve consistency. Section 10 discusses the means by which theories may be extended in the HOL system. Of particular importance are certain mechanisms for introducing new constants and types.

In section 11 we define the individual axioms of the HOL logic. The resulting theory is of special interest, as are what we call its definitional extensions, which we define in section 11.3: they are all consistent and have a common standard set-theoretic model; their theorems comprise what are normally taken to be the theorems of HOL by those who shun axiomatic extensions.

3 PREAMBLE

We introduce the new theory. Its parent is the theory \textit{spc001} which contains definitions concerned with the HOL language.

\sml
open \textit{theory}"\textit{spc001}"
new \textit{theory}"\textit{spc003}"

4 THE RULES OF INFERENCE

In this section we treat the syntax manipulating functions required to define the various rules of inference. We consider each inference rule in turn. In the HOL system the inference rules are functions which take theorems (and other things) as arguments and return theorems. Since we
cannot define the type of theorems until we have defined the inference rules we define the rules as
functions taking sequents (and other things) as arguments and returning sequents.

4.1 Free Variables

*freevars_list* returns the free variables of a term listed in order of first appearance (from left to right
in the usual concrete syntax).

HOL Constant

| freevars_list: TERM \rightarrow ((STRING \times TYPE)LIST) |
| \forall s : STRING; ty : TYPE; tm f a vty b : TERM \rightarrow |
| freevars_list (mk_var(s, ty)) = [(s, ty)] |
| \land |
| freevars_list (mk_const(s, ty)) = [] |
| \land |
| (has_mk_comb(f, a) tm \Rightarrow freevars_list tm = freevars_list f \setminus freevars_list a) |
| \land |
| ((has_mk_abs(vty, b) tm \land mk_var(s, ty) = vty) \Rightarrow |
| freevars_list tm = freevars_list b \setminus \{f\}) |

*freevars_set* returns the set of free variables of a term. We use it in cases where the order of appearance
of the free variables in the term is immaterial.

HOL Constant

| freevars_set: TERM \rightarrow (STRING \times TYPE) SET |
| \forall tm : TERM \bullet freevars_set tm = Elems(freevars_list tm) |

4.2 Object Language Constructs

To define the rules of inference we need to form certain object language types and terms. We have
already defined the function space type constructor. The other definitions needed are given in this
section.

We need to form instances of the polymorphic constant “=":

HOL Constant

| Equality : TYPE \rightarrow TERM |
| \forall ty \bullet Equality ty = mk_const("="), Fun ty (Fun ty Bool) |

The following is our analogue of the derived constructor function for equations in the HOL system.
HOL Constant

\[ \text{has\_mk\_eq} : (\text{TERM} \times \text{TERM}) \rightarrow \text{TERM} \rightarrow \text{BOOL} \]

\[ \forall \ lhs\ rhs\ tm \bullet \text{has\_mk\_eq}(lhs, rhs) \ tm \Leftrightarrow \]
\[ \exists \ tm2 \bullet \]
\[ \text{has\_mk\_comb}(\text{Equality}(\text{type\_of\_term\ lhs}, \ lhs), \ lhs) \ tm2 \]
\[ \land \text{has\_mk\_comb}(tm2, rhs) \ tm \]

We also need to form implications. The following functions are analogous to those treating equality above.

HOL Constant

\[ \text{Implication} : \text{TERM} \]

\[ \text{Implication} = \text{mk\_const}("\Rightarrow", \text{Fun\ Bool\ (Fun\ Bool\ Bool)}) \]

HOL Constant

\[ \text{has\_mk\_imp} : (\text{TERM} \times \text{TERM}) \rightarrow \text{TERM} \rightarrow \text{BOOL} \]

\[ \forall \ lhs\ rhs\ tm \bullet \text{has\_mk\_imp}(lhs, rhs) \ tm \Leftrightarrow \]
\[ \exists \ tm2 \bullet \]
\[ \text{has\_mk\_comb}(\text{Implication}, \ lhs) \ tm2 \]
\[ \land \text{has\_mk\_comb}(tm2, rhs) \ tm \]

### 4.3 Substitution of Equals

In this section we define the inference rule \textit{SUBST}.

In essence, \textit{SUBST} says that given a theorem whose conclusion is an equation, \( A = B \), where \( A \) and \( B \) are arbitrary terms of the same type, and given any other theorem with conclusion \( C \), we may obtain a new theorem by substituting \( B \) for any subterm of \( C \) which is identical with \( A \). This is subject to the proviso that no variable capture problems arise, i.e. no free variables of \( B \) should become bound in the conclusion of the new theorem. (The assumption set of the consequent theorem is the union of the assumption sets of the antecedent theorems.)

The inference rule is, in fact, slightly more general. It allows one to use a whole set of theorems whose conclusions are equations to perform (simultaneous) substitutions for many subterms of \( C \). Moreover, it is implemented as a functional relation, effectively by renaming any bound variables of \( C \) which would give rise to the capture problem.

The inference rule is parametrised by a template term and a set of some of its free variables, one for each equation. The actual statement of the rule is, essentially, that, if the result of substituting the left hand sides of the equations for the corresponding variables in the template term is equal to \( C \) (modulo renaming bound variables), then we may infer the result of substituting the right hand sides of the equations for the corresponding template variables in the template term (providing we rename bound variables to avoid the capture problem).

The notions we must formalise are therefore: (i) substituting terms for free variables in a term according to a given mapping of variables to terms renaming bound variables as necessary to avoid variable capture; (ii) testing equivalence of terms \textit{modulo} renaming of bound variables (aka. \( \alpha \)-conversion).
4.3.1 Substitution

We will need to choose new names for variables. More precisely, given a variable and a set of same
we will wish to rename the variable, when necessary, to ensure that the result does not lie in the set.
In practice in an implementation we would insist that the new name be derived from the old one in
a specified way.

HOL Constant

\textbf{variant} : ((STRING $\times$ TYPE) SET) $\rightarrow$ (STRING $\times$ TYPE) $\rightarrow$ STRING

\forall \ vs \ v \ ty \bullet \\
\hspace{1cm} if \ -(v, ty) \in \ vs \hspace{1cm} then \ variant \ vs \ (v, ty) = v \\
\hspace{1cm} else \ -(\text{variant} \ vs \ (v, ty), ty) \in \ vs

Now we can define \textit{subst}. Given a function \(R\) associating free variables with terms, \(\text{subst} \ R \ t1\) is the
term resulting from replacing every free variable \(\text{mk\_var}(s, ty)\) in \(t1\) by \(R(\text{mk\_var}(s, ty))\) with bound
variables renamed as necessary to avoid capture. Variables which are not to be changed correspond
to pairs \((s, ty)\) with \(R(s, ty) = \text{mk\_var}(s, ty)\).

Note \(R\) here is intended to respect types, in the sense that \(\forall \text{ty} \bullet \text{type\_of\_term}(R(s, ty)) = ty\), but
this is not checked here (since it is convenient for \textit{subst} to be a total function). This property should
be checked whenever \textit{subst} is used.

The only difficult case in \textit{subst} is when the second argument is an abstraction. In this case we calculate
the variables which must not get captured (this is the value \textit{new\_frees} below) and use \textit{variant} to
give an alternative name for the bound variable if necessary. We then perform the substitution on
the body using a function, \(RR\), which is \(R\) modified to send the old bound variable to the new one.

HOL Constant

\textbf{subst} : ((STRING $\times$ TYPE) $\rightarrow$ TERM) $\rightarrow$ TERM $\rightarrow$ TERM

\forall \ R : (STRING $\times$ TYPE) $\rightarrow$ TERM; \(tm : \text{TERM}\);  \\
\hspace{1cm} s : \text{STRING}; \ ty : \text{TYPE}; \ vty : \text{TERM};  \\
\hspace{1cm} f : \text{TERM}; \ a : \text{TERM}; \ b : \text{TERM}  \\
\bullet  \\
\hspace{1cm} \text{subst} \ R \ (\text{mk\_var}(s, ty)) = R(s, ty)  \\
\wedge  \\
\hspace{1cm} \text{subst} \ R \ (\text{mk\_const}(s, ty)) = \text{mk\_const}(s, ty)  \\
\wedge  \\
\hspace{1cm} (\text{has\_mk\_comb}(f, a) \ tm \Rightarrow  \\
\hspace{2cm} (\text{subst} \ R \ tm = \epsilon t \bullet \text{has\_mk\_comb}(\text{subst} \ R \ f, \text{subst} \ R \ a) t))  \\
\wedge  \\
\hspace{1cm} ((\text{has\_mk\_abs}(vty, b) \ tm \wedge \text{mk\_var}(s, ty) = vty) \Rightarrow  \\
\hspace{2cm} (\text{subst} \ R \ tm =  \\
\hspace{3cm} \text{let} \ \text{new\_frees} = \bigcup (\text{Graph} \ (\text{freevars\_set} \ o \ R) \ \text{Image}  \\
\hspace{4cm} (\text{freevars\_set} \ b \setminus \{(s, ty)\}))  \\
\hspace{3cm} \text{in} \ \text{let} \ s' = \text{variant} \ \text{new\_frees} \ (s, ty)  \\
\hspace{3cm} \text{in} \ \text{let} \ RR \ x = \text{if} \ x = (s, ty) \text{ then} \text{mk\_var} \ (s', ty) \ \text{else} \ R \ x)
The special case of substitution where we simply wish to rename a variable is needed in the definition of our \(\alpha\)-conversion test and elsewhere. The following function `rename` is used for this purpose. `rename(v, ty)w e` is the result of changing the name in every free occurrence of the variable with name \(v\), and type \(ty\), in the term \(e\), to \(w\), renaming any bound variables as necessary.

\[
\text{rename} : (STRING \times TYPE) \rightarrow STRING \rightarrow TERM \rightarrow TERM
\]

\[
\forall v : STRING; ty : TYPE; w : STRING
\]

\[
\bullet
\]

\[
\text{rename} (v, ty) w = \\
\text{subst} (\lambda x \bullet \text{if } x = (v, ty) \text{ then mk}_{\text{var}}(w, ty) \text{ else mk}_{\text{var}} x)
\]

### 4.3.2 \(\alpha\)-conversion

Our \(\alpha\)-conversion test is as follows:

\[
\text{aconv} : TERM \rightarrow TERM \rightarrow BOOL
\]

\[
\forall t1 \ t2 : TERM
\]

\[
\text{aconv} t1 \ t2 \iff \ \\
(t1 = t2)
\]

\[
\lor \ (\exists t1f \ t1a \ t2f \ t2a \bullet \\
\text{has}_{\text{mk}}_{\text{comb}}(t1f, t1a)t1 \\
\land \text{has}_{\text{mk}}_{\text{comb}}(t2f, t2a)t2 \\
\land \text{aconv} t1f \ t2f \land \text{aconv} t1a \ t2a)
\]

\[
\lor \ (\exists v1 \ v2 \ ty \ v1ty \ v2ty \ v1 \ v2 \bullet \\
\text{has}_{\text{mk}}_{\text{abs}}(v1ty, b1)t1 \land \text{has}_{\text{mk}}_{\text{abs}}(v2ty, b2)t2 \\
\land \text{mk}_{\text{var}}(v1, ty) = v1ty \land \text{mk}_{\text{var}}(v2, ty) = v2ty \\
\land \text{aconv} b1 (\text{rename} (v2, ty) v1 b2) \\
\land ((v1 = v2) \lor (\neg(v1, ty) \in \text{freevars}_\text{set} b2))
\]

### 4.3.3 The Inference Rule \texttt{SUBST}

We can now define the inference rule. Its first argument gives the correspondence between the template variables and equation theorems. We could take this argument to behave as \texttt{REFL axiom o mk var} on variables which are not template variables. Note that, to allow implementation as a partial function, we test up to \(\alpha\)-convertibility on the first sequent argument only. Note also that the
way that the first argument to subst is constructed by dismantling equations ensures that it respects types.

HOL Constant

\( \textbf{SUBST\_rule} : ((\text{STRING} \times \text{TYPE}) \rightarrow \text{SEQ}) \rightarrow \text{TERM} \rightarrow \text{SEQ} \rightarrow \text{SEQ} \rightarrow \text{BOOL} \)

\( \forall \text{eqs} \ \text{tm} \ \text{old\_asms} \ \text{old\_conc} \ \text{new\_asms} \ \text{new\_conc} \bullet \)

\( \text{SUBST\_rule} \ \text{eqs} \ \text{tm} \ (\text{old\_asms}, \ \text{old\_conc}) \ (\text{new\_asms}, \ \text{new\_conc}) \Leftrightarrow \)

\( (\forall v \ \text{ty} \bullet \)

\( \exists \text{lhs} \ \text{rhs} \bullet \)

\( \text{has\_mk\_eq}(\text{lhs}, \ \text{rhs})(\text{concl}(\text{eqs}(v, \ \text{ty}))) \land \)

\( \text{(type\_of\_term} \ \text{lhs} = \text{ty}) \land \)

\( (\text{aconv} \ \text{old\_conc} \ (\text{subst}(\lambda(v,\text{ty})\bullet\text{lhs}\bullet\exists\text{rhs}\bullet\text{has\_mk\_eq}(\text{lhs}, \ \text{rhs})(\text{concl}(\text{eqs}(v,\text{ty}))))\text{tm})) \land \)

\( (\text{new\_conc} = \text{subst}(\lambda(v,\text{ty})\bullet\text{rhs}\bullet\exists\text{lhs}\bullet\text{has\_mk\_eq}(\text{lhs}, \ \text{rhs})(\text{concl}(\text{eqs}(v,\text{ty}))))\text{tm}) \land \)

\( (\text{new\_asms} = \text{old\_asms} \cup \bigcup \{ \text{asms} | \exists vty \bullet \text{asms} = (\text{hyp} \ (\text{eqs} \ vty))\}) \)

4.4 Abstraction: ABS

Again \( \text{ABS} \) is a partial function which we specify as a relation:

HOL Constant

\( \textbf{ABS\_rule} : (\text{STRING} \times \text{TYPE}) \rightarrow \text{SEQ} \rightarrow \text{SEQ} \rightarrow \text{BOOL} \)

\( \forall \text{vty} \ \text{old\_asms} \ \text{old\_conc} \ \text{new\_asms} \ \text{new\_conc} \bullet \)

\( \exists \ \text{old\_lhs} \ \text{old\_rhs} \ \text{new\_lhs} \ \text{new\_rhs} \ v \bullet \)

\( \text{has\_mk\_eq}(\text{old\_lhs}, \ \text{old\_rhs})\text{old\_conc} \land \)

\( \text{has\_mk\_eq}(\text{new\_lhs}, \ \text{new\_rhs})\text{new\_conc} \land \)

\( \text{mk\_var} \ \text{vty} = v \land \)

\( \text{has\_mk\_abs}(v, \ \text{old\_lhs}) \ \text{new\_lhs} \land \)

\( \text{has\_mk\_abs}(v, \ \text{old\_rhs}) \ \text{new\_rhs} \land \)

\( (\neg vty \in \bigcup(\text{Graph freevars\_set Image old\_asms})) \land \)

\( (\text{new\_asms} = \text{old\_asms}) \)

4.5 Type Instantiation

The ability to prove and use general (polymporphic) theorems is one of the great strengths of the HOL system. The feature in the inference system which gives this strength is the inference rule \( \text{INST\_TYPE} \) which allows us to instantiate the type variables in the conclusion of a polymorphic theorem.
In essence, the inference rule says that, given a theorem with conclusion, \( \mathcal{A} \), say, we may infer the theorem which has the same assumption set and whose conclusion results from instantiating every type in \( \mathcal{A} \) according to a given mapping of type variables to types. This is subject to two provisos: (i) no type variable may be changed which appears in the assumption set for the theorem; (ii) no two variables in the assumptions or conclusion of the antecedent theorem, which are different, by virtue of their type, should become identified in the consequent theorem as a result of the transformation.

The first proviso is, we believe, only enforced to preserve a convention of natural deduction systems, whereby inference rules involve only simple set operations on the assumption sets. It would seem to be quite in order for the first proviso to be dropped provided we insisted that the type instantiation be applied to every term in the sequent (we have, of course, not done this).

The second proviso cannot be avoided. Consider for example: \( \lambda(x : *\star)\bullet \lambda(x : *\star)\bullet(x : *\star) \). If the types in this were instantiated according to \( \{ : *\star \mapsto : *, : *\mapsto : * \} \), then from:

\[
\Gamma \vdash \forall(y : *\star)(z : *)\bullet(\lambda(x : *\star)\bullet\lambda(x : *)(x : *\star))yz = y
\]

we could infer that:

\[
\Gamma \vdash \forall(y : *)(z : *)(\lambda(x : *)(x : *)(x : *))yz = y
\]

whence, by \( \beta \)-conversions:

\[
\Gamma \vdash \forall(y : *)(z : *)zz = y.
\]

This leads to a contradiction whenever : * is instantiated to a type with more than one inhabitant.

To permit an implementation which is convenient to use, the inference rule is actually formulated without the second proviso. Instead, variables (both free and bound, in general) in the conclusion of the consequent theorem, which would violate the rule are renamed to avoid the problem. It is valid to rename free variables in these circumstances, given the first proviso, since the variables in question cannot occur free in the assumption set. Note that it would be invalid to rename free variables in \( \mathcal{A} \) which are not changed by the type instantiation (since these may appear free in the assumption set).

Formalising these notions is a little tricky. We present here a highly unconstructive specification, reminiscent of \( \alpha \)-conversion. The notion to be formalised is the predicate on pairs of terms which says whether one is a type instance of another according to a given mapping of type variables to types and with respect to a set of variables with which clashes must not occur (this will be the set of free variables of the assumptions in practice).

It is entertaining and instructive to consider algorithms meeting these specifications.

### 4.5.1 Instantiation of Terms

Instantiation of terms is a little tricky. The following two functions should be viewed as local to the function \( \text{inst} \). \( \text{inst\_loc1} \) is very similar to an \( \alpha \)-convertibility test. Indeed \( \text{aconv} \) could have been defined as \( \text{inst\_loc1} I \). The first \( \text{TERM} \) argument of \( \text{inst\_loc1} \) and \( \text{inst\_loc2} \) gives the terms whose types are being instantiated (i.e. it is the “more polymorphic” term).

\( \text{inst\_loc1} \) checks that one term, \( tm2 \), is a type instance of \( tm1 \), according to a mapping from type variable names to types given by \( \text{tysubs} \), under the assumption that the free variable names agree, i.e. that the first occurrence of each variable which may need renaming will be its binding occurrence in a \( \lambda \) – abstraction.
\hspace*{1cm}

**inst_loc1**: \((\text{STRING} \rightarrow \text{TYPE}) \rightarrow \text{TERM} \rightarrow \text{TERM} \rightarrow \text{BOOL}\)

\[
\forall \quad \text{tysubs} : \text{STRING} \rightarrow \text{TYPE};
\text{tm1} \text{ tm2} : \text{TERM} \bullet
\quad \text{inst_loc1} \text{ tysubs} \text{ tm1} \text{ tm2} \iff
\quad (\exists s \ ty1 \ ty2 \ mk_X \bullet
\quad \quad \quad ((\text{mk}_X = \text{mk}_\text{var}) \lor (\text{mk}_X = \text{mk}_\text{const}))
\quad \land \quad \text{mk}_X(s, ty1) = \text{tm1} \land \text{mk}_X(s, ty2) = \text{tm2}
\quad \land \quad (ty2 = \text{inst\_type} \text{ tysubs} \text{ ty1}))
\quad \lor
\quad (\exists \text{tm1f} \text{ tm1a} \text{ tm2f} \text{ tm2a} \bullet
\quad \quad \quad \text{has}_\text{mk}_\text{comb}(\text{tm1f}, \text{tm1a})\text{tm1} \land \text{has}_\text{mk}_\text{comb}(\text{tm2f}, \text{tm2a})\text{tm2}
\quad \land \quad \text{inst_loc1} \text{ tysubs} \text{ tm1f} \text{ tm2f} \land \text{inst_loc1} \text{ tysubs} \text{ tm1a} \text{ tm2a})
\quad \lor
\quad (\exists v1 \text{ v2} \text{ ty1} \text{ ty2} \text{ b1} \text{ b2} \text{ v1ty1} \text{ v2ty2} \bullet
\quad \quad \quad \text{mk}_\text{var}(v1, ty1) = v1ty1 \land \text{has}_\text{mk}_\text{abs}(v1ty1, b1)\text{tm1}
\quad \land \quad \text{mk}_\text{var}(v2, ty2) = v2ty2 \land \text{has}_\text{mk}_\text{abs}(v2ty2, b2)\text{tm2}
\quad \land \quad \text{inst_loc1} \text{ tysubs} (\text{rename} (v1, ty1) v2 b1) b2
\quad \land \quad (ty2 = \text{inst\_type} \text{ tysubs} \text{ ty1})
\quad \land \quad \neg (\exists ty3 \text{ v2ty3} \bullet
\quad \quad \quad \text{mk}_\text{var}(v2, ty3) = v2ty3
\quad \land \quad ((v2, ty3) \in \text{freevars\_set} \text{ b1})
\quad \land \quad (ty2 = \text{inst\_type} \text{ tysubs} \text{ ty3})
\quad \land \quad (\neg v2ty3 = v1ty1))
\]

**inst_loc2** uses **inst_loc1** to check that a term \(\text{tm2}\) is a type instance of the result of renaming free variables of a term \(\text{tm2}\) according to a mapping given by a list of pairs. It also checks that the type of the second variable in each pair in the list is a type instance of the type of the first variable in the pair, and that the second variable in each pair is not in the set, \(\text{avoid}\), unless both names and types agree for that pair. In the application of **inst_loc2** in **inst** the list of pairs is obtained by combining the free variable lists of the two terms side by side. The set \(\text{avoid}\) is a set of variables (coming from the assumptions of a sequent) whose free occurrences must not change as a result of the type instantiation.

\hspace*{1cm}

**inst_loc2**: \(((\text{STRING} \times \text{TYPE}) \text{ SET}) \rightarrow \text{TERM} \rightarrow \text{TERM} \rightarrow \text{BOOL}\)

\[
\forall \text{avoid} : (\text{STRING} \times \text{TYPE}) \text{ SET};
\text{tysubs} : \text{STRING} \rightarrow \text{TYPE};
\text{v1} : \text{STRING}; \text{ty1} : \text{TYPE};
\text{v2} : \text{STRING}; \text{ty2} : \text{TYPE};
\text{rest} : (((\text{STRING} \times \text{TYPE}) \times (\text{STRING} \times \text{TYPE})) \text{ LIST}) \rightarrow \text{TERM} \rightarrow \text{TERM} \rightarrow \text{BOOL}
\text{tm1} \text{ tm2} : \text{TERM} \bullet


\[(\text{inst\_loc2 avoid tysubs } \equiv \text{tm1 tm2} \leftrightarrow \text{inst\_loc1 tysubs tm1 tm2})\]

\[
\land (\text{inst\_loc2 avoid tysubs (Cons ((v1, ty1), (v2, ty2)) rest) tm1 tm2} \leftrightarrow

(((v2, ty2) \notin \text{avoid}) \Rightarrow ((v1, ty1) = (v2, ty2)))
\land (\text{ty2 = inst\_type tysubs ty1})
\land \text{inst\_loc2 avoid tysubs rest}

((\text{rename (v1, ty1) v2 tm1 tm2})
\]

With the above preliminaries we can now define inst. Note that the condition that the free variable lists of the two terms have the same length is required to ensure that \text{inst\_loc2} examines each free variable of each term.

\text{HOL Constant}

\[
\text{inst} : (\text{STRING } \times \text{ TYPE} \set) \rightarrow

(\text{STRING } \rightarrow \text{ TYPE}) \rightarrow \text{ TERM } \rightarrow \text{ TERM}
\]

\[
\forall \text{avoid} : (\text{STRING } \times \text{ TYPE} \set);
\text{tysubs : STRING } \rightarrow \text{ TYPE}; \text{ tm1 : TERM }\bullet
\text{let tm2 = inst avoid tysubs tm1}
\text{in let fl1 = freevars_list tm1}
\text{in let fl2 = freevars_list tm2}
\text{in}

((\text{Length fl1 = Length fl2})
\land \text{inst\_loc2 avoid tysubs (Combine fl1 fl2) tm1 tm2})
\]

\[4.5.2\text{ The Inference Rule INST\_TYPE}\]

Given inst, we need a few simple auxiliaries before we can define the inference rule INST\_TYPE.

We need to detect the type variables in a term. We use some auxiliary functions to do this: \text{type\_tyvars} detects the type variables in a type.

\text{HOL Constant}

\[
\text{type\_tyvars : TYPE } \rightarrow (\text{STRING} \set)
\]

\[
(\forall s\bullet \text{type\_tyvars (mk\_var\_type s)} = \{s\})
\land (\forall s\ tl\bullet \text{type\_tyvars (mk\_type(s, tl))} =

\bigcup (\text{Elems (Map type\_tyvars tl)})
\]

\text{term\_types} detects the types in a term.
term_types : TERM → (TYPE SET)

\forall tm : TERM; s: STRING; ty : TYPE;
f : TERM; a : TERM; v: TERM; b: TERM
term_types (mk_var(s, ty)) = \{ty\}
∧
term_types (mk_const(s, ty)) = \{ty\}
∧
(has_mk_comb(f, a) tm ⇒ (term_types tm = term_types f ∪ term_types a))
∧
(has_mk_abs(v, b) tm ⇒ (term_types tm = term_types v ∪ term_types b))

term_tyvars detects all the type variables in a term using the previous two functions.

term_tyvars : TERM → (STRING SET)

\forall tm • term_tyvars tm = \bigcup\{(Graph type_tyvars Image (term_types tm))\}

INST_TYPE_rule is now readily defined:

INST_TYPE_rule : (STRING → TYPE) → SEQ → SEQ → BOOL

\forall tysubs old_asms old_conc new_seq
INST_TYPE_rule tysubs (old_asms, old_conc) new_seq ⇔
(∀ tyv •
 (tyv ∈ \bigcup\{(Graph term_tyvars Image old_asms)\}) ⇒
 (tysubs tyv = mk_var_type tyv))
∧
let asms_frees = \bigcup\{(Graph freevars_set Image old_asms)\}
in
 new_seq = (old_asms, inst asms_frees tysubs old_conc)

4.6 Discharging an Assumption: DISCH

DISCH is, in essence, the usual rule of natural deduction which allows one to infer from a proof of B on the assumption A, that A⇒B on no assumption. The actual rule is suitably generalised to cover sequents and their assumption sets. It is not required that A be in the assumption set, and the logic would probably not be complete otherwise.
\textbf{Theorem 4.7: Modus Ponens: MP}

This is the usual rule: from \( A \Rightarrow B \) and \( A \), infer \( B \). This generalises to sequents by taking the union of the assumption sets.

\textbf{Theorem 5.1: The Axiom Schema ASSUME}

\textsc{Assume} allows us to infer for any boolean term \( A \), that \( A \) holds on the assumptions \( \{A\} \). This is straightforward to formalise. We must check that the term being assumed is of the right type.

\textbf{Theorem 5.2: The Axiom Schema REFL}

\textsc{Ref} says that for any term \( A \), we may infer that \( A = A \) without assumptions.
5.3 The Axiom Schema $\text{BETA}\_\text{CONV}$

$\text{BETA}\_\text{CONV}$ says that, without any assumptions, any $\beta$-redex is equal to its $\beta$-reduction. This is straightforward to define, given the apparatus we used to define $\text{SUBST}$. Note that the way we construct the first argument to $\text{subst}$ by dismantling a combination ensures that it respects types.

**HOL Constant**

$\text{BETA}\_\text{CONV\_axiom} : \text{TERM} \to \text{SEQ} \to \text{BOOL}$

\[
\forall \text{tm} \, \text{new\_seq}\bullet \\
\text{BETA}\_\text{CONV\_axiom} \, \text{tm} \, \text{new\_seq} \iff \\
\exists \, v \, ty \, vty \, b \, abs \, a \bullet \\
\text{mk\_var}(v, ty) = vty \land \\
\text{has\_mk\_abs}(vty, b)abs \land \\
\text{has\_mk\_comb}(abs, a)tm \land \\
(\text{new\_seq} = \\
\text{let subs: (STRING \times TYPE) \to TERM} = \\
(\lambda (vx, tyx)\bullet \text{if} \, vx = v \land tyx = ty \, \text{then} \, a \, \text{else} \, \text{mk\_var}(vx, tyx)) \\
in \\
\{\}, (et\bullet \text{has\_mk\_eq}(tm, \text{subst} \, subs \, b)t)))
\]

6 DERIVABILITY

In this section we will define derivability. This is a relation between sets of sequents and sequents. As usual, we first define direct derivability. We include instances of the axiom schemata as valid direct derivations from no premisses. This is merely for convenience, we could equally well include all instances of the axiom schemata as axioms in every theory when theories are defined.

**HOL Constant**

$\text{directly\_derivable\_from} : \text{SEQ} \to (\text{SEQ SET}) \to \text{BOOL}$

\[
\forall \text{seq} \, \text{seqs} \bullet \\
\text{directly\_derivable\_from} \, \text{seq} \, \text{seqs} \iff \\
(\exists \, \text{eqs} \, \text{tm} \, \text{old\_seq} \bullet \\
\text{Ran} \, (\text{Graph} \, \text{eqs}) \subseteq \text{seqs} \land \text{old\_seq} \in \text{seqs} \land \text{SUBST\_rule} \, \text{eqs} \, \text{tm} \, \text{old\_seq} \, \text{seq}) \\
\lor \\
(\exists \, vty \, \text{old\_seq} \bullet \text{old\_seq} \in \text{seqs} \land \text{ABS\_rule} \, vty \, \text{old\_seq} \, \text{seq}) \\
\lor \\
(\exists \, \text{tysubs} \, \text{old\_seq} \bullet \text{old\_seq} \in \text{seqs} \land \text{INST\_TYPE\_rule} \, \text{tysubs} \, \text{old\_seq} \, \text{seq}) \\
\lor \\
(\exists \, \text{tm} \, \text{old\_seq} \bullet \text{old\_seq} \in \text{seqs} \land \text{DISCH\_rule} \, \text{tm} \, \text{old\_seq} \, \text{seq}) \\
\lor \\
(\exists \, \text{imp\_seq} \, \text{ant\_seq} \bullet \\
\text{imp\_seq} \in \text{seqs} \land \text{ant\_seq} \in \text{seqs} \land \text{MP\_rule} \, \text{imp\_seq} \, \text{ant\_seq} \, \text{seq}) \\
\lor \\
(\exists \, \text{tm} \bullet \text{ASSUME\_axiom} \, \text{tm} \, \text{seq})
\]
Proofs will just be lists of sequents. Any non-empty list is a valid proof (of the sequent at its head) on the premisses given by those elements of the list which are not directly derivable from elements later in the list. There is little point in making the relevant type definition for a syntactic class of proofs in this sense, since they contain so little information. We simply define the function which extracts the set of premisses.

\[ \forall \ seq \ rest \bullet \ \premisses \ [\] = \{\} \]
\[ \wedge \ \premisses \ (\text{Cons} \ seq \ rest) = \]
\[ \begin{cases} \text{if directly\_derivable\_from} \ seq \ (\text{Elems} \ rest) \\ \text{then} \ premisses \ rest \\ \text{else} \ \{\seq\} \cup \premisses \ rest \end{cases} \]

HOL Constant
\[ \text{premisses} : \ (\text{SEQ \ LIST}) \rightarrow (\text{SEQ \ SET}) \]

HOL Constant
\[ \text{derivable\_from} : \text{SEQ} \rightarrow (\text{SEQ \ SET}) \rightarrow \text{BOOL} \]

7 NORMAL THEORIES

In [1] a type \textit{THEORY} is defined to represent the idea of a theory comprising signatures governing the formation of types and terms and a set of axioms. However the type \textit{THEORY} is too general for our present purposes, since we have formulated rules of inference on the assumption that the nullary type “\texttt{bool}” and the constants “\texttt{=}” and “\texttt{⇒}” are available. In this section we define a predicate \textit{normal\_theory} which selects the theories in which the inference rules are intended to be valid. (The normal theories correspond to those whose type structures and signatures are standard in the terminology of [3]. Unfortunately the term \textit{standard theory} is used for a stronger notion in [3].)

7.1 Object Language Constructs

To define the type of all well-formed HOL theories we need two further object language constructs: the choice function “\texttt{ε}” and the type of individuals “\texttt{ind}”. These are required since we will follow [3] in insisting on the presence of the equality, implication and choice functions in each theory. It
is noteworthy however that neither the rules of inference nor the standard conservative extension mechanisms require choice or the individuals; they are only used in the axioms given in section 11.

\[ \text{HOL Constant} \]
\[
\text{Star : TYPE}
\]
\[
\text{Star} = \text{mk\_var\_type } "\,*" 
\]

\[ \text{HOL Constant} \]
\[
\text{Choice : TERM}
\]
\[
\text{Choice} = \text{mk\_const}(("ε", \text{Fun (Fun Star Bool) Star})) 
\]

\[ \text{HOL Constant} \]
\[
\text{Ind : TYPE}
\]
\[
\text{Ind} = \text{mk\_type}("ind", []) 
\]

7.2 Normal Theories

We now wish to define the predicate \textit{normal\_theory}. It is natural to say that the normal theories are those which extend the minimal normal theory which contains only \texttt{"bool"}, \texttt{"="} etc. Thus we must define this minimal normal theory and also the notion of extension of theories.

\textit{MIN} is the minimal normal theory. It is represented by the triple \textit{MIN\_REP}:

\[ \text{HOL Constant} \]
\[
\text{MIN\_REP : TY\_ENV} \times \text{CON\_ENV} \times \text{SEQS}
\]
\[
\text{MIN\_REP} = ( 
\{("bool", 0); ("→", 2 ); ("ind", 0 )\},
\{("="; \text{Fun Star (Fun Star Bool)});
("⇒", \text{Fun Bool (Fun Bool Bool)});
("ε", \text{Fun (Fun Star Bool) Star}),(} 
\}
\)

\[ \text{HOL Constant} \]
\[
\text{MIN : THEORY}
\]
\[
\text{MIN} = \text{abs\_theory MIN\_REP} 
\]

Extension for objects of type \textit{THEORY} is the following binary relation:

\[ \text{SML} \]
\[
declare\_infix(200, "extends"); 
\]
$\text{extends} : \text{THEORY} \to \text{THEORY} \to \text{BOOL}$

\[
\forall \ thy1 \ thy2. \\
\thy1 \text{ extends } \thy2 \iff \\
(\text{types } \thy2 \subseteq \text{types } \thy1) \land \\
(\text{constants } \thy2 \subseteq \text{constants } \thy1) \land \\
(\text{axioms } \thy2 \subseteq \text{axioms } \thy1)
\]

The normal theories are those which extend the minimal theory \(\text{MIN}\). Note that we do not exclude inconsistent theories here. (This corresponds to the possibility of introducing inconsistent axioms in the HOL system).

\[
\text{is_normal_theory} : \text{THEORY} \to \text{SET}
\]

\[
\forall \thy. \thy \in \text{is_normal_theory} = \thy \text{ extends } \text{MIN}
\]

8 \ THEOREMS

We can, at last, define the type of all HOL theorems. A theorem will consist of a sequent and a theory. The type is the subtype of the type of all such pairs in which the sequent is well-formed with respect to the type and constant environments of the theory and in which the sequent may be derived from the axioms of the theory.

\[
\text{is_thm} : (\text{SEQ} \times \text{THEORY}) \to \text{SET}
\]

\[
\forall \seq \thy. \\
(\seq, \thy) \in \text{is_thm} \iff \\
\thy \in \text{is_normal_theory} \land \\
\seq \in \text{sequents } \thy \land \\
\text{derivable\_from } \seq \ (\text{axioms } \thy)
\]

Note that if \((\seq, \thy)\) is a theorem in this sense, the derivation of \(\seq\) from the axioms of \(\thy\) may involve sequents which are not well-formed with respect to \(\thy\) (i.e. which contain type operators or constants which are not in \(\thy\)). This is allowed since it simplifies the definition of derivability and makes no difference to the set of theorems in a given theory (this is essentially the fact that the extension mechanisms \text{new\_type} and \text{new\_constant} are conservative).

Proving that \(\exists \thy. \thy \in \text{is_thm}\) involves rather more work than has been involved in previous type definitions. (A witness is easy to supply, e.g. \(\text{REFL}\_\text{axiom} (\text{mk\_var}(\text{'x, Star})), \text{MIN}\) would do. However, to show that it is a witness we need to compute \text{sequents } \text{MIN}\) and to do this we must show that \text{MIN\_REP} is indeed the representative of a theory and checking the conditions on the two environments is rather long-winded). For the time being we therefore defer this proof task and use \text{type\_spec} to define the type, \(\text{THM}\), of theorems.
The components of a theorem are extracted using the following functions:

\[
\begin{align*}
\text{thm\_seq} & : \text{THM} \rightarrow \text{SEQ} \\
\forall \text{thm} \quad \text{thm\_seq thm} & = \text{Fst}(\text{rep\_thm thm})
\end{align*}
\]

\[
\begin{align*}
\text{thm\_thy} & : \text{THM} \rightarrow \text{THEROY} \\
\forall \text{thm} \quad \text{thm\_thy thm} & = \text{Snd}(\text{rep\_thm thm})
\end{align*}
\]

9 CONSISTENCY AND CONSERVATIVE EXTENSION

A theory is consistent if not every sequent which is well-formed in it can be derived from the axioms:

\[
\begin{align*}
\forall \text{thy} \quad \text{thy} \in \text{consistent\_theory} & \iff \\
& \exists \text{seq} \quad \\
& (\text{seq} \in \text{sequents thy}) \\
& \land \\
& \neg(\text{derivable\_from seq (axioms thy)})
\end{align*}
\]

An extension of a theory is conservative if no sequent of the smaller theory is provable in the larger but not in the smaller.

\[
\text{declare\_infix(200, "conservatively\_extends")};
\]
\textbf{HOL Constant}

\begin{align*}
\textsf{conservatively\_extends} & : \text{THEORY} \to \text{THEORY} \to \text{BOOL} \\
\forall \ thy1 \ thy2. & \\
thy1 \text{ conservatively\_extends} \ thy2 & \iff \\
(\thy1 \text{ extends} \ thy2) & \land \\
(\forall \ seq. & \\
(seq \in \text{sequents} \ thy2) & \Rightarrow \\
(\text{derivable\_from} \ seq \ (\text{axioms} \ thy1)) & \Rightarrow \\
(\text{derivable\_from} \ seq \ (\text{axioms} \ thy2)) & 
\end{align*}

\section{DEFINITIONAL EXTENSIONS}

\subsection{Object Language Constructs}

A theory \textit{LOG} in which more of the standard logical apparatus is available will be needed to define some of the definitional extension mechanisms. For example, \textit{new\_type\_definition} works with a theorem whose conclusion must be an existentially quantified term of a particular form. To define \textit{LOG} we need some more object language types and terms and these are defined in this section. (It is convenient to leave the definition of \textit{LOG} itself until we have defined \textit{new\_definition}.)

The formulation of the various logical connectives follows the HOL manual, [3].

It is helpful now to have the following term constructor functions. Note that we are now using total functions to approximate partial ones; we must, therefore, be careful only to apply them to appropriate arguments.

\begin{itemize}
\item \textbf{HOL Constant}
\end{itemize}

\begin{align*}
\text{mk\_comb} & : (\text{TERM} \times \text{TERM}) \to \text{TERM} \\
\text{mk\_comb} & = \$e \ o \ has\_mk\_comb \\
\text{mk\_abs} & : (\text{TERM} \times \text{TERM}) \to \text{TERM} \\
\text{mk\_abs} & = \$e \ o \ has\_mk\_abs \\
\text{mk\_eq} & : (\text{TERM} \times \text{TERM}) \to \text{TERM} \\
\text{mk\_eq} & = \$e \ o \ has\_mk\_eq \\
\text{mk\_imp} & : (\text{TERM} \times \text{TERM}) \to \text{TERM} \\
\text{mk\_imp} & = \$e \ o \ has\_mk\_imp
\end{align*}
We can now define the object language constructs needed. (These could be defined via our explicit representations of types and terms using strings. This has not been done since the explicit concrete syntax used is very hard to read.)

10.1.1 Truth

The constant $T : \text{bool}$ is defined by the following equation:

$$T = (\lambda(x : \text{bool}) \cdot x) = (\lambda(x : \text{bool}) \cdot x)$$

HOL Constant

<table>
<thead>
<tr>
<th>Truth : TERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth = mk_const(&quot;T&quot;, \text{Bool})</td>
</tr>
</tbody>
</table>

HOL Constant

<table>
<thead>
<tr>
<th>Truth_def : TERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth_def =</td>
</tr>
<tr>
<td>let $x = \text{mk_var}(&quot;x&quot;, \text{Bool})$</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>$\text{mk_eq}(\text{mk_abs}(x, x), \text{mk_abs}(x, x))$</td>
</tr>
</tbody>
</table>

10.1.2 Universal Quantification

The constant $\forall : (\ast \rightarrow \text{bool}) \rightarrow \text{bool}$ is defined by the following equation:

$$\forall = (\lambda(P : \ast \rightarrow \text{bool}) \cdot P = (\lambda(x : \ast) \cdot T)$$

HOL Constant

<table>
<thead>
<tr>
<th>Forall : TYPE $\rightarrow$ TERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forall $ty = \text{mk_const}(&quot;\forall&quot;, \text{Fun (Fun ty Bool) Bool})$</td>
</tr>
</tbody>
</table>

HOL Constant

<table>
<thead>
<tr>
<th>Forall_def : TERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forall_def =</td>
</tr>
<tr>
<td>let $P = \text{mk_var}(&quot;P&quot;, \text{Fun Star Bool})$</td>
</tr>
<tr>
<td>in let $x = \text{mk_var}(&quot;x&quot;, \text{Star})$</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>$\text{mk_abs}(P, \text{mk_eq}(P, \text{mk_abs}(x, \text{Truth})))$</td>
</tr>
</tbody>
</table>

HOL Constant

<table>
<thead>
<tr>
<th>mk_forall : (TERM $\times$ TERM) $\rightarrow$ TERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall tm1 ; tm2 \cdot \text{mk_forall}(tm1, tm2) =$</td>
</tr>
<tr>
<td>$\text{mk_comb}($Forall (type_of_term tm1), $\text{mk_abs}(tm1, tm2))$</td>
</tr>
</tbody>
</table>
10.1.3 Existential Quantification

The constant \( \exists : (\ast \rightarrow \text{bool}) \rightarrow \text{bool} \) is defined by the following equation, which defines \( \exists \) in terms of the choice function \( \epsilon : (\ast \rightarrow \text{bool}) \rightarrow \ast \):

\[
\exists = \lambda(P : \ast \rightarrow \text{bool}) \cdot P(\epsilon P)
\]

(This may be a little perplexing at first sight. In the intended interpretations, given a predicate \( P : \ast \rightarrow \text{bool} \), if there is some \( x : \ast \) for which \( P \) is true (i.e. for which \( Px = T \)), then \( \epsilon P \) is such an \( x \). I.e. taking as known the intuitive notion of “whether or not something with a given property exists”, \( \epsilon \) chooses something with a given property if such a thing exists. The above definition can be viewed as taking as known the informal notion of “choosing something with a given property” and defining \( \exists \) to determine whether or not something with a given property exists by attempting to choose something with the given property and checking whether the attempt succeeded.)

HOL Constant

\begin{align*}
\text{Exists} & : \text{TYPE} \rightarrow \text{TERM} \\
\forall \text{ ty} \cdot \text{Exists ty} = \text{mk\_const}("\exists", \text{Fun (Fun ty Bool) Bool})
\end{align*}

HOL Constant

\begin{align*}
\text{Exists\_def} & : \text{TERM} \\
\text{Exists\_def} &= \text{let } P = \text{mk\_var}("P", \text{Fun Star Bool}) \\
& \text{in let } P\text{choiceP} = \text{mk\_comb}(P, \text{mk\_comb}(\text{Choice, P})) \\
& \text{in} \\
& \text{mk\_abs}(P, P\text{choiceP})
\end{align*}

HOL Constant

\begin{align*}
\text{has\_mk\_exists} & : (\text{TERM} \times \text{TERM}) \rightarrow \text{TERM} \rightarrow \text{BOOL} \\
\forall \text{ tm1 tm2 tm3} \cdot \text{has\_mk\_exists}(\text{tm1, tm2}) \text{ tm3} = \\
\text{has\_mk\_comb}(&\text{Exists (type\_of\_term tm1), mk\_abs(tm1, tm2))tm3}
\end{align*}

HOL Constant

\begin{align*}
\text{mk\_exists} & : (\text{TERM} \times \text{TERM}) \rightarrow \text{TERM} \\
\forall \text{ tm1 tm2} \cdot \text{mk\_exists}(\text{tm1, tm2}) = \\
\text{mk\_comb}(&\text{Exists (type\_of\_term tm1), mk\_abs(tm1, tm2)})
\end{align*}

10.1.4 Falsity

The constant \( F : \text{bool} \) is defined by the following equation:

\[
F = \forall(x : \text{bool}) \cdot x
\]
(Again this may seem perplexing. The type bool is intended to contain the truth values. The above definition says that false is the truth value of the proposition that every truth value is true!)

\[
\text{Falsity} = \text{mk}_\text{const}("F", \text{Bool})
\]

\[
\text{Falsity} = \text{mk}_\text{const}("F", \text{Bool})
\]

10.1.5 Negation

The constant \( \neg : \text{bool} \to \text{bool} \) is defined by the following equation:

\[
\neg = \lambda (b : \text{bool}) \cdot b \Rightarrow \text{F}
\]

\[
\text{Negation} = \text{mk}_\text{const}("\neg", \text{Fun Bool Bool})
\]

\[
\text{Negation} = \text{mk}_\text{const}("\neg", \text{Fun Bool Bool})
\]

10.1.6 Conjunction

The constant \( \land : \text{bool} \to \text{bool} \to \text{bool} \) is defined by the following equation:

\[
\land = \lambda (b1 : \text{bool}) \cdot \lambda (b2 : \text{bool}) \cdot (b1 \Rightarrow (b2 \Rightarrow b)) \Rightarrow b
\]

(I assume, but do not know, that the above formulation has some practical advantage in the present context over the more obvious definition in terms of \( \neg \) and \( \Rightarrow \).

The name of the constant is a slash, \( \div \), followed by a backslash, \( \backslash \). The backslash character must be escaped by another backslash character within an HOL string.
HOL Constant

\[
\text{Conjunction} : \text{TERM}
\]

\[
\text{Conjunction} = \text{mk}_\text{const}("\land\", \text{Fun Bool (Fun Bool Bool)})
\]

HOL Constant

\[
\text{Conjunction}_\text{def} : \text{TERM}
\]

\[
\text{Conjunction}_\text{def} = \\
\text{let } b = \text{mk}_\text{var}("b", \text{Bool}) \\
in \text{let } b1 = \text{mk}_\text{var}("b1", \text{Bool}) \\
in \text{let } b2 = \text{mk}_\text{var}("b2", \text{Bool}) \\
in \\
\text{mk}_\text{abs}(b1, \text{mk}_\text{abs}(b2, \text{mk}_\text{forall}(b, \text{mk}_\text{imp}(\text{mk}_\text{imp}(b1, \text{mk}_\text{imp}(b2, b)), b))))
\]

A derived constructor function for conjunctions is useful.

HOL Constant

\[
\text{mk}_\text{conj} : (\text{TERM }\times\text{ TERM}) \rightarrow \text{TERM}
\]

\[
\forall \text{tm1 tm2} \bullet \\
\text{mk}_\text{conj}(\text{tm1}, \text{tm2}) = \text{mk}_\text{comb}(\text{mk}_\text{comb}(\text{Conjunction}, \text{tm1}),\text{tm2})
\]

\[10.1.7\] Disjunction

The constant $\lor : \text{bool} \rightarrow \text{bool} \rightarrow \text{bool}$ is defined by the following equation:

\[
\mathbf{\lor} = \lambda b1 \bullet \lambda b2 \bullet \forall b ((\mathbf{b1} \Rightarrow b) \Rightarrow (\mathbf{b2} \Rightarrow b)) \Rightarrow b
\]

(As for conjunction I assume this has some advantage over a definition from the propositional calculus.)

The name of the constant is a backslash, \\, followed by a slash, /. The backslash character must be escaped by another backslash character within an HOL string.

HOL Constant

\[
\text{Disjunction} : \text{TERM}
\]

\[
\text{Disjunction} = \text{mk}_\text{const}("\lor\", \text{Fun Bool (Fun Bool Bool)})
\]
Disjunction_def : TERM

Disjunction_def = let b = mk_var("b", Bool) in let b1 = mk_var("b1", Bool) in let b2 = mk_var("b2", Bool) in mk_abs(b1, mk_abs(b2, mk_forall(b, mk_imp(mk_imp(b1, b), mk_imp(mk_imp(b2, b), b)))))

A derived constructor function for disjunctions is useful later.

\[ \text{mk\_disj} : (\text{TERM} \times \text{TERM}) \to \text{TERM} \]

\[ \forall tm1 \; tm2 \bullet \text{mk\_disj}(tm1, \; tm2) = \text{mk\_comb}(\text{mk\_comb}(\text{Disjunction}, \; tm1), \; tm2) \]

### 10.1.8 ONE_ONE

The definition of Type_Definition below requires the notion of a one-to-one function. The constant \( \text{ONE\_ONE} \) is defined by the following equation:

\[ \text{ONE\_ONE} = \lambda (f : \ast \to \ast) \bullet (\forall (x1 : \ast) \bullet (\forall (x2 : \ast) \bullet (f \; x1 = f \; x2) \Rightarrow (x1 = x2)) \]

\[ \text{StarStar} : \text{TYPE} \]

\[ \text{StarStar} = \text{mk\_var\_type} \; "\ast\ast" \]

\[ \text{One\_One} : \text{TERM} \]

\[ \text{One\_One} = \text{mk\_const}(\"\text{ONE\_ONE}\", \; \text{Fun}(\text{Fun \; Star \; StarStar}) \; \text{Bool}) \]

\[ \text{One\_One\_def} : \text{TERM} \]

\[ \text{One\_One\_def} = \text{let } f = \text{mk\_var}(\"f\", \text{Fun \; Star \; StarStar}) \text{ in let } x1 = \text{mk\_var}(\"x1\", \text{Star}) \text{ in let } x2 = \text{mk\_var}(\"x2\", \text{Star}) \text{ in mk\_abs}(f, \; \text{mk\_forall}(x1, \; \text{mk\_forall}(x2, \text{mk\_imp(mk\_eq(mk\_comb(f, \; x1), \; mk\_comb(f, \; x2)), \text{mk\_eq(x1, \; x2))))})) \]
The axiom of infinity requires the notion of an onto function. The constant \textit{ONTO} is defined by the following equation:

\[ \text{ONTO} = \lambda(f : * \to **) \forall(y : **) \exists(x : *) y = f x \]

The name is all upper case to avoid conflict with the actual constant \textit{Onto} used in the metalanguage system.

\textbf{10.1.10 Type Definition}

\textit{Type Definition} may be new to some readers. It is a term asserting that a function represents one type as a subtype of another. It is used in defining \textit{new_type_definition}. It has type \((** \to \text{bool}) \to (** \to *) \to \text{bool}\) and is defined by the following equation:

\[ \text{Type Definition} = \lambda(P: ** \to \text{bool}) \forall(rep : ** \to **) \text{ONET ONE rep} \]

\[ \land \forall(x: **) P x = \exists(y: *) x = rep y \]

It is useful later to have a version of \textit{Type Definition} parameterised over the types involved.
The first two definitional extension mechanisms, \textit{new\_type} and \textit{new\_constant} are conservative, but not very powerful.

\textit{new\_type} is used to declare a name to be used as a type constructor. No axioms about the type are introduced so that only instances of polymorphic functions may be applied to it. The only constraint is that the name should not be a type constructor in the theory to be extended.

To see, syntactically, that \textit{new\_type} is conservative observe that, given a proof in which the new type does not appear in the conclusion, distinct applications of the new type operator could be replaced by distinct type variables not used elsewhere in the proof. The result would be a proof in the unextended theory with the same conclusion as the original proof.

\textit{new\_constant} is used to declare a name to be used as a constant of a given type. No axioms about the constant are introduced so that it behaves as a value which we cannot determine. The only constraint is that the name should not be a constant in the theory to be extended and that the type of the constant should be well-formed.

\textit{new\_type} : \( \mathbb{N} \rightarrow \text{STRING} \rightarrow \text{THEORY} \rightarrow \text{THEORY} \rightarrow \text{BOOL} \)

\textit{new\_constant} : \( \text{STRING} \rightarrow \text{TYPE} \rightarrow \text{THEORY} \rightarrow \text{THEORY} \rightarrow \text{BOOL} \)
Again it is easy to see syntactically that this is conservative. Simply replace distinct instances of the new constant in a proof by distinct variables not used elsewhere in the proof to obtain a proof in the unextended theory.

10.3 new_axiom

new_axiom is both powerful and dangerous! It allows a sequent with no hypotheses and a given conclusion to be taken as an axiom. The only constraint is that the sequent be well-formed with respect to the environments of the theory being extended.

It is convenient, for technical reasons, in [2] to have the more general operation of adding a set of new axioms. We therefore define new_axiom in terms of the more general new_axioms.

HOL Constant
\[
\text{new_axioms} : (\text{TERM SET}) \rightarrow \text{THEORY} \rightarrow \text{THEORY} \rightarrow \text{BOOL}
\]

\[
\forall \ tms \ thy1 \ thy2 \bullet \\
\text{new_axioms} \ tms \ thy1 \ thy2 = \\
\text{let seqs} = \{(x, tm) \mid x = \{\} \land tm \in \text{tms}\} \\
in \\
\text{seqs} \subseteq \text{sequents thy1} \land \\
\text{types thy2} = \text{types thy1} \land \\
\text{constants thy2} = \text{constants thy1} \land \\
\text{axioms thy2} = \text{axioms thy1} \cup \text{seqs}
\]

HOL Constant
\[
\text{new_axiom} : \text{TERM} \rightarrow \text{THEORY} \rightarrow \text{THEORY} \rightarrow \text{BOOL}
\]

\[
\forall \ tm \ thy1 \ thy2 \bullet \\
\text{new_axiom} \ tm \ thy1 \ thy2 = \text{new_axioms} \ \{tm\} \ thy1 \ thy2
\]

10.4 new_definition

new_definition is useful and conservative. It allows the simultaneous introduction of a new constant and an axiom asserting that the new constant is equal to a given term. The constraints imposed are (a) the name must satisfy the check made in new_constant, (b) the term must be closed and (c) the term must contain no bound variables whose types contain type variables which do not appear in the type of the new constant. Condition (c) ensures that different type instances of the term result in different instances of the constant; this avoids a possible inconsistency (see [2] for an example which arises in the course of this specification).
HOL Constant

new_definition \( : \) STRING -> TERM -> THEORY -> THEORY -> BOOL

\( \forall \) name tm thy1 thy2 •

new_definition name tm thy1 thy2 ↔

let ty = type_of_term tm in

\( \exists \) thy1a •

new_constant name ty thy1 thy1a \&

freevars_set tm = \{\} \&

term_tyvars tm \subseteq type_tyvars ty \&

new_axiom (mk_eq(mk_const(name, ty), tm)) thy1a thy2

10.5 new_specification

new_specification allows the simultaneous introduction of a set of new constants satisfying a given predicate provided that a theorem asserting the existence of some set of values satisfying the constants is given. An axiom asserting the predicate for the new constants is introduced. Like new_definition, new_specification is useful and conservative.

The constraints imposed are analogous to those imposed in new_definition: (a) the constant names must be pairwise distinct and different from any constant name in the theory being extended, (b) the predicate must have no free variables apart from those corresponding to the new constants, (c) any type variable contained in a bound variable of the predicate must appear as a type variable of each of the new constants. Also, of course, the theorem must have the right form.

Since we now need to work with existential quantifiers it is necessary to introduce the theory LOG. We impose the restriction that new_specification may only be used to extend theories which extend LOG.

HOL Constant

LOG : THEORY

\( \exists \) thy1 thy2 thy3 thy4 thy5 thy6 thy7 thy8 thy9•

let Name = λcon•s•ty•mk_const(s, ty) = con in

(new_definition (Name Truth) Truth_def MIN thy1 ∧

new_definition (Name (Forall Star)) Forall_def thy1 thy2 ∧

new_definition (Name (Exists Star)) Exists_def thy2 thy3 ∧

new_definition (Name Falsity) Falsity_def thy3 thy4 ∧

new_definition (Name Negation) Negation_def thy4 thy5 ∧

new_definition (Name Conjunction) Conjunction_def thy5 thy6 ∧

new_definition (Name Disjunction) Disjunction_def thy6 thy7 ∧

new_definition (Name One_One) One_One_def thy7 thy8 ∧

new_definition (Name ONTO) ONTO_def thy8 thy9 ∧

new_definition (Name (Type_Definition Star StarStar)) Type_Definition_def thy9 LOG)
To define \textit{new specification} we need the relation \textit{has\_list\_mk\_exists}, and the relation \textit{new\_constants} which is like \textit{new\_constant} but handles a set of new constants.

\begin{itemize}
  \item \textbf{HOL Constant}
    \begin{align*}
      \textit{has\_list\_mk\_exists} & : (\text{TERM LIST}) \to \text{TERM} \to \text{TERM} \to \text{BOOL} \\
      (\forall \text{tm1 tm2} \bullet \text{has\_list\_mk\_exists} [] \text{tm1 tm2} \iff \text{tm1} = \text{tm2}) \\
      \wedge \\
      (\forall \text{v rest tm1 tm2} \bullet \\
      \text{has\_list\_mk\_exists}\ (\text{Cons v rest}) \text{tm1 tm2} \iff \\
      \exists \text{rem} \bullet \text{has\_mk\_exists}(\text{v, rem}) \text{tm2} \wedge \\
      \text{has\_list\_mk\_exists}\ \text{rest rem tm1})
    \end{align*}
  \\
  \item \textbf{HOL Constant}
    \begin{align*}
      \textit{new\_constants} & : ((\text{STRING} \times \text{TYPE}) \text{SET}) \to \text{THEORY} \to \text{THEORY} \to \text{BOOL} \\
      \forall \text{cons thy1 thy2} \bullet \\
      \text{new\_constants}\ \text{cons thy1 thy2} \iff \\
      \text{Dom cons} \cap \text{Dom (constants thy1)} = \{\} \wedge \\
      \text{Ran cons} \subseteq \text{wf\_type(types thy1)} \wedge \\
      \text{constants thy2} = \text{constants thy1} \cup \text{cons} \wedge \\
      \text{types thy2} = \text{types thy1} \wedge \\
      \text{axioms thy2} = \text{axioms thy1}
    \end{align*}
\end{itemize}

We can now define \textit{new specification}.

\begin{itemize}
  \item \textbf{HOL Constant}
    \begin{align*}
      \textit{new\_specification} & : ((\text{STRING} \times (\text{STRING} \times \text{TYPE})) \text{LIST}) \to \\
      \text{TERM} \to \text{THM} \to \text{THEORY} \to \text{THEORY} \to \text{BOOL} \\
      \forall \text{pairs tm thm thy1 thy2} \bullet \\
      \text{new\_specification}\ \text{pairs tm thm thy1 thy2} = \\
      \text{let conl} = \text{Fst}(\text{Split pairs}) \\
      \text{in let varl} = \text{Map mk\_var (Snd(Split pairs))} \\
      \text{in let tyl} = \text{Map Snd (Snd(Split pairs))} \\
      \text{in let subs = \lambda (s, ty) \bullet} \\
      \quad \text{if} \quad \exists \text{c} \bullet (\text{c}, (s, ty)) \in \text{Elems pairs} \\
      \quad \text{then mk\_const}((\text{c}\bullet\text{c}, (s, ty)) \in \text{Elems pairs}, ty) \\
      \quad \text{else mk\_var(s, ty)} \\
      \text{in let axiom = subst subs tm} \\
      \text{in (\exists conc\bullet} \\
      \text{has\_list\_mk\_exists varl tm conc} \wedge \text{thy1 extends LOG} \\
      \wedge (\text{freevars\_set conc} = \{\}) \\
      \wedge \text{conl} \in \text{Distinct}
    \end{align*}
\end{itemize}
\& \texttt{varl} \in \text{Distinct} \\
\& \texttt{thm\_seq \_thm} = (\{\}, \texttt{conc}) \\
\& \texttt{thy1} \text{ \texttt{extends}\_thm \_thm} \\
\& (\forall \texttt{ty} \bullet \texttt{ty} \in \text{Elems\_ty}l \Rightarrow \texttt{term\_tyvars\_conc} \subseteq \texttt{type\_tyvars\_ty}) \\
\& (\exists \texttt{thyl1a} \bullet \\
\quad \texttt{new\_constants\ (Elems\ (Combine\ \texttt{cond\_ty}l))\ \texttt{thy1}\ \texttt{thy1a} \land \\
\quad \texttt{new\_axiom\ axiom\ \texttt{thy1a}}\ \texttt{thy2}) \\

\subsection*{10.6 \texttt{new\_type\_definition}}

\texttt{new\_type\_definition} allows the introduction of a new type in one-to-one correspondence with the subset of an existing type satisfying a given predicate, given a theorem asserting that the subset is not empty. A new axiom asserting the existence of a representation function for the new type is introduced. Like \texttt{new\_definition}, \texttt{new\_type\_definition} is useful and conservative.

For simplicity, we have made the list of type variable names to be used as the parameters of the type being defined, a parameter to \texttt{new\_type}. The constraints imposed are (a) that the list of type parameter names contain no repeats, (b) the theorem must have the right form and (c) all type variables contained in the predicate must be contained in the list of type parameters names. Condition (c) ensures that different type instances of the new axiom involve different type instances of the new type.

\begin{haskell}
\textbf{HOL Constant} \texttt{new\_type\_definition} : \\
\texttt{STRING \to (STRING\ LIST) \to THM \to THEORY \to THEORY \to BOOL} \\
\forall \texttt{name\ \_typars\ \_thm\ \_thy1\ \_thy2} \bullet \\
\texttt{new\_type\_definition\ \texttt{name}\ \_typars\ \_thm\ \_thy1\ \_thy2} \leftrightarrow \\
\exists \texttt{p\ \_xty\ \_x\ \_ty\ \_px\ \_thy1a\ \_axiom} \bullet \\
\texttt{let\ \texttt{newty} = \texttt{mk\_type(name, Map\ \texttt{mk\_var\_type\ \_typars})}} \\
in\ \texttt{let\ \texttt{f} = \texttt{mk\_var("f", Fun\ \texttt{newty}\ \_ty)}} \\
in\ \texttt{\_thy1\ \texttt{extends\ \_LOG}} \\
\& \texttt{\_hyp\ (\texttt{thm\_seq\ \_thm}) = \{\}} \\
\& \texttt{\_has\_\_mk\_exists\ (\texttt{xty}, \_px)\ (\texttt{concl\ (\texttt{thm\_seq\ \_thm})})} \\
\& \texttt{\_mk\_var\ (\_x, \_ty) = \_xty} \\
\& \texttt{\_has\_\_mk\_comb\ (\_p, \_xty)\ \_px} \\
\& \texttt{\_freevars\_set\ \_p = \{\}} \\
\& \texttt{\_term\_tyvars\ \_p \subseteq \texttt{Elems\ \_typars}} \\
\& \texttt{\_typars \in \text{Distinct}} \\
\& \texttt{\_has\_\_mk\_exists(f, \_mk\_comb(mk\_comb(\_Type\_Definition\ \texttt{newty}\ \_ty, \_p), \_f))\ \_axiom} \\
\& \texttt{\_new\_\_type\ (\#\ \_typars)\ \_name\ \_thy1\ \_thy1a} \\
\& \texttt{\_new\_\_axiom\ \_axiom\ \_thy1a\ \_thy2} 
\end{haskell}
11  THE THEORY INIT

By extending the theory LOG with five axioms we will arrive at the theory INIT. In a typical HOL
proof development system all theories will be extensions of this theory.

11.1  The Axioms

11.1.1  BOOL_CASES_AX

This is the law of the excluded middle:

\[ \forall (b:\text{bool}) \bullet (b = T) \lor (b = F) \]

HOL Constant

\[
\text{BOOL_CASES_AX} : \text{TERM}
\]

- \[ \text{let } b = \text{mk\_var("b", } \text{Bool}) \]
- \[ \text{in mk\_forall}(b, \text{mk\_disj(mk\_eq}(b, \text{Truth}), \text{mk\_eq}(b, \text{Falsity}))) \]

11.1.2  IMP_ANTISYM_AX

This says that implication is an antisymmetric relation:

\[ \forall (b1:\text{bool}) \bullet \forall (b2:\text{bool}) \bullet (b1 \Rightarrow b2) \Rightarrow (b2 \Rightarrow b1) \Rightarrow (b1=b2) \]

HOL Constant

\[
\text{IMP_ANTISYM_AX} : \text{TERM}
\]

- \[ \text{let } b1 = \text{mk\_var("b1", } \text{Bool}) \]
- \[ \text{in let } b2 = \text{mk\_var("b2", } \text{Bool}) \]
- \[ \text{in mk\_forall}(b1, \text{mk\_forall}(b2, \text{mk\_imp(mk\_imp}(b1, b2), \text{mk\_imp}(b2, b1)), \text{mk\_eq}(b1, b2)))) \]

11.1.3  ETA_AX

This says that an \(\eta\)-redex is equal to its \(\eta\)-reduction.

\[ \forall (f:* \rightarrow **) \bullet \lambda(x:*) \bullet f \ x = f \]

HOL Constant

\[
\text{ETA_AX} : \text{TERM}
\]

- \[ \text{let } f = \text{mk\_var("f1", } \text{Fun Star StarStar}) \]
- \[ \text{in let } x = \text{mk\_var("x", } \text{Star}) \]
- \[ \text{in mk\_forall}(f, \text{mk\_eq}(\text{mk\_abs}(x, \text{mk\_comb}(f, x)), f)) \]
11.1.4 SELECT_AX

This is the defining property of the choice function $\epsilon$.

$$\text{SELECT_AX} \vdash \forall (P:* \rightarrow \text{bool}) \bullet \forall (x:* \bullet P \ x \ \Rightarrow \ P(\epsilon \ P))$$

HOL Constant

\[
\text{SELECT_AX} : \text{TERM}
\]

\[
\text{SELECT_AX} = \\
\text{let } P = \text{mk_var("P", Fun Star Bool)} \\
in \text{let } x = \text{mk_var("x", Star)} \\
in \text{mk forall}(P, \text{mk forall}(x, \\
\text{mk imp}(\text{mk comb}(P, x), \text{mk comb}(P, \text{mk comb}(\text{Choice}, P))))))
\]

11.1.5 INFINITY_AX

This is the axiom of infinity. It asserts that the type $\text{ind}$ is in one-to-one correspondence with a proper subset of itself:

$$\text{INFINITY_AX} \vdash \exists (f: \text{ind} \rightarrow \text{ind}) \bullet \text{ONE ONE} \ f \ \land \ \neg \text{ONT0} \ f$$

HOL Constant

\[
\text{INFINITY_AX} : \text{TERM}
\]

\[
\text{INFINITY_AX} = \\
\text{let } f = \text{mk_var("f", Fun Ind Ind)} \\
in \text{mk conj}(\text{mk comb(One One, f)}, \text{mk comb(Negation, \text{mk comb(ONT0, f)})})
\]

11.2 The Theory

HOL Constant

\[
\text{INIT} : \text{THEORY}
\]

\[
\exists \ thy1 \ thy2 \ thy3 \ thy4 \ thy5 \ thy6 \ \bullet \\
\text{new axiom BOOL CASES AX LOG thy1} \\
\land \ \text{new axiom IMP ANTISYM AX thy1 thy2} \\
\land \ \text{new axiom ETA AX thy2 thy3} \\
\land \ \text{new axiom SELECT AX thy4 thy5} \\
\land \ \text{new type 0 (Fst(dest type Ind)) thy5 thy6} \\
\land \ \text{new axiom INFINITY AX thy6 INIT}
\]

11.3 DEFINITIONAL EXTENSIONS

We will say that a theory $\text{thy1}$ is a definitional extension of a theory $\text{thy2}$ if one may go from $\text{thy2}$ to $\text{thy1}$ by some sequence of extensions by the functions $\text{new type}$, $\text{new constant}$, $\text{new definition}$,
new\_specification and new\_type\_definition. It is stressed that definitional extensions in this sense comprise significantly more than just extension by adjoining a defining equation for a new constant.

HOL Constant

definitional\_extension \text{ : THEORY $\rightarrow$ THEORY SET}

\[
\forall \text{thy} \cdot \text{definitional\_extension thy} = \bigcap \{ \text{thyset} \mid \\
\text{thy} \in \text{thyset} \\
\land ( \forall \text{thy1 thy2 arity name} \cdot \\
\text{thy1} \in \text{thyset} \land \\
\text{new\_type arity name thy1 thy2 } \Rightarrow \text{thy2} \in \text{thyset}) \\
) \land ( \forall \text{thy1 thy2 name type} \cdot \\
\text{thy1} \in \text{thyset} \land \\
\text{new\_constant name type thy1 thy2 } \Rightarrow \text{thy2} \in \text{thyset}) \\
) \land ( \forall \text{thy1 thy2 name tm} \cdot \\
\text{thy1} \in \text{thyset} \land \\
\text{new\_definition name tm thy1 thy2 } \Rightarrow \text{thy2} \in \text{thyset}) \\
) \land ( \forall \text{thy1 thy2 pairs tm thm} \cdot \\
\text{thy1} \in \text{thyset} \land \\
\text{new\_specification pairs tm thm thy1 thy2 } \Rightarrow \text{thy2} \in \text{thyset}) \\
) \land ( \forall \text{thy1 thy2 name typars thm} \cdot \\
\text{thy1} \in \text{thyset} \land \\
\text{new\_type\_definition name typars thm thy1 thy2 } \Rightarrow \text{thy2} \in \text{thyset})
\]

Of particular importance are theories which may be obtained from INIT by definitional extension. These theories are of interest since, we assert, they form a sound formalism in which much of the practical machine-checked proof work one might wish to do can be carried out.
## 12 INDEX OF DEFINED TERMS

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS_rule</td>
<td>8</td>
</tr>
<tr>
<td>aconv</td>
<td>7</td>
</tr>
<tr>
<td>ASSUME_axiom</td>
<td>13</td>
</tr>
<tr>
<td>BETA_CONV_axiom</td>
<td>14</td>
</tr>
<tr>
<td>BOOL_CASES_AX</td>
<td>31</td>
</tr>
<tr>
<td>Choice</td>
<td>16</td>
</tr>
<tr>
<td>Conjunction_def</td>
<td>23</td>
</tr>
<tr>
<td>Conjunction</td>
<td>23</td>
</tr>
<tr>
<td>conservatively_extends</td>
<td>19</td>
</tr>
<tr>
<td>consistent_theory</td>
<td>18</td>
</tr>
<tr>
<td>definitional_extension</td>
<td>33</td>
</tr>
<tr>
<td>derivable_from</td>
<td>15</td>
</tr>
<tr>
<td>directly_derivable_from</td>
<td>14</td>
</tr>
<tr>
<td>DISCH_rule</td>
<td>13</td>
</tr>
<tr>
<td>Disjunction_def</td>
<td>24</td>
</tr>
<tr>
<td>Disjunction</td>
<td>23</td>
</tr>
<tr>
<td>Equality</td>
<td>4</td>
</tr>
<tr>
<td>ETA_AX</td>
<td>31</td>
</tr>
<tr>
<td>Exists_def</td>
<td>21</td>
</tr>
<tr>
<td>Exists</td>
<td>21</td>
</tr>
<tr>
<td>extends</td>
<td>17</td>
</tr>
<tr>
<td>Falsity_def</td>
<td>22</td>
</tr>
<tr>
<td>Falsity</td>
<td>22</td>
</tr>
<tr>
<td>Forall_def</td>
<td>20</td>
</tr>
<tr>
<td>Forall</td>
<td>20</td>
</tr>
<tr>
<td>freevars_list</td>
<td>4</td>
</tr>
<tr>
<td>freevars_set</td>
<td>4</td>
</tr>
<tr>
<td>has_list_mk_exists</td>
<td>29</td>
</tr>
<tr>
<td>has_mk_eq</td>
<td>5</td>
</tr>
<tr>
<td>has_mk_exists</td>
<td>21</td>
</tr>
<tr>
<td>has_mk_imp</td>
<td>5</td>
</tr>
<tr>
<td>Implication</td>
<td>5</td>
</tr>
<tr>
<td>IMP_ANTISYM_AX</td>
<td>31</td>
</tr>
<tr>
<td>Ind</td>
<td>16</td>
</tr>
<tr>
<td>INFINITY_AX</td>
<td>32</td>
</tr>
<tr>
<td>INIT</td>
<td>32</td>
</tr>
<tr>
<td>inst_loc1</td>
<td>10</td>
</tr>
<tr>
<td>inst_loc2</td>
<td>10</td>
</tr>
<tr>
<td>INST_TYPE_rule</td>
<td>12</td>
</tr>
<tr>
<td>inst</td>
<td>11</td>
</tr>
<tr>
<td>is_normal_theory</td>
<td>17</td>
</tr>
<tr>
<td>is_thm</td>
<td>17</td>
</tr>
<tr>
<td>LOG</td>
<td>28</td>
</tr>
<tr>
<td>MIN_REP</td>
<td>16</td>
</tr>
<tr>
<td>MIN</td>
<td>16</td>
</tr>
<tr>
<td>mk_abs</td>
<td>19</td>
</tr>
<tr>
<td>mk_comb</td>
<td>19</td>
</tr>
<tr>
<td>mk_conj</td>
<td>23</td>
</tr>
<tr>
<td>mk_disj</td>
<td>24</td>
</tr>
<tr>
<td>mk_eq</td>
<td>19</td>
</tr>
<tr>
<td>mk_exists</td>
<td>21</td>
</tr>
<tr>
<td>mk_forall</td>
<td>20</td>
</tr>
<tr>
<td>mk_imp</td>
<td>19</td>
</tr>
<tr>
<td>mk_thm</td>
<td>18</td>
</tr>
<tr>
<td>MP_rule</td>
<td>13</td>
</tr>
<tr>
<td>Negation</td>
<td>22</td>
</tr>
<tr>
<td>Negation_def</td>
<td>22</td>
</tr>
<tr>
<td>new_axioms</td>
<td>27</td>
</tr>
<tr>
<td>new_axiom</td>
<td>27</td>
</tr>
<tr>
<td>new_axioms</td>
<td>27</td>
</tr>
<tr>
<td>new_constants</td>
<td>29</td>
</tr>
<tr>
<td>new_constant</td>
<td>26</td>
</tr>
<tr>
<td>new_definition</td>
<td>28</td>
</tr>
<tr>
<td>new_specification</td>
<td>29</td>
</tr>
<tr>
<td>new_type_definition</td>
<td>30</td>
</tr>
<tr>
<td>new_type</td>
<td>26</td>
</tr>
<tr>
<td>ONTO</td>
<td>25</td>
</tr>
<tr>
<td>ONTO</td>
<td>25</td>
</tr>
<tr>
<td>premisses</td>
<td>15</td>
</tr>
<tr>
<td>REFL_axiom</td>
<td>13</td>
</tr>
<tr>
<td>rename</td>
<td>7</td>
</tr>
<tr>
<td>SELECT_AX</td>
<td>32</td>
</tr>
<tr>
<td>spc003</td>
<td>3</td>
</tr>
<tr>
<td>StarStar</td>
<td>24</td>
</tr>
<tr>
<td>Star</td>
<td>16</td>
</tr>
<tr>
<td>subst</td>
<td>6</td>
</tr>
<tr>
<td>term_types</td>
<td>12</td>
</tr>
<tr>
<td>term_tyvars</td>
<td>12</td>
</tr>
<tr>
<td>thm_seq</td>
<td>18</td>
</tr>
<tr>
<td>thm_thy</td>
<td>18</td>
</tr>
<tr>
<td>THM</td>
<td>18</td>
</tr>
<tr>
<td>Truth_def</td>
<td>20</td>
</tr>
<tr>
<td>Truth</td>
<td>20</td>
</tr>
<tr>
<td>Type_Definition_def</td>
<td>26</td>
</tr>
<tr>
<td>Type_Definition</td>
<td>25</td>
</tr>
<tr>
<td>variant</td>
<td>6</td>
</tr>
</tbody>
</table>