Chapter 9

Syntax and Semantics

9.1 Introduction

This chapter describes the syntax and set-theoretic semantics of the logic supported by the HOL system, which is a variant of Church's simple theory of types [3] and will henceforth be called the HOL logic, or just HOL. The meta-language for this description will be English, enhanced with various mathematical notations and conventions. The object language of this description is the HOL logic. Note that there is a 'meta-language', in a different sense, associated with the HOL logic, namely the programming language ML. This is the language used to manipulate the HOL logic by users of the system, and is described in detail in Part I of *DESCRIPTION*. It is hoped that because of context, no confusion results from these two uses of the word 'meta-language'. When ML is described in Part I, ML is the object language under consideration—and English is again the meta-language!

The HOL syntax contains syntactic categories of types and terms whose elements are intended to denote respectively certain sets and elements of sets. This set theoretic interpretation will be developed along side the description of the HOL syntax, and in the next chapter the HOL proof system will be shown to be sound for reasoning about properties of the set theoretic model.[1] This model is given in terms of a fixed set of sets $\mathcal{U}$, which will be called the *universe* and which is assumed to have the following properties.

Inhab Each element of $\mathcal{U}$ is a non-empty set.

Sub If $X \in \mathcal{U}$ and $\emptyset \neq Y \subseteq X$, then $Y \in \mathcal{U}$.

Prod If $X \in \mathcal{U}$ and $Y \in \mathcal{U}$, then $X \times Y \in \mathcal{U}$. The set $X \times Y$ is the cartesian product, consisting of ordered pairs (x, y) with $x \in X$ and $y \in Y$, with the usual set-theoretic coding of ordered pairs, viz. $(x, y) = \{\{x\}, \{x, y\}\}$.

Pow If $X \in \mathcal{U}$, then the powerset $P(X) = \{Y : Y \subseteq X\}$ is also an element of $\mathcal{U}$.

Infty $\mathcal{U}$ contains a distinguished infinite set I.

[1] There are other, 'non-standard' models of HOL, which will not concern us here.

Choice There is a distinguished element $\mathrm{ch} \in \prod_{X \in \mathcal{U}} X$. The elements of the product $\prod_{X \in \mathcal{U}} X$ are (dependently typed) functions: thus for all $X \in \mathcal{U}$, X is non-empty by **Inhab** and $\mathrm{ch}(X) \in X$ witnesses this.

There are some consequences of these assumptions which will be needed. In set theory functions are identified with their graphs, which are certain sets of ordered pairs. Thus the set $X \to Y$ of all functions from a set X to a set Y is a subset of $P(X \times Y)$; and it is a non-empty set when Y is non-empty. So **Sub**, **Prod** and **Pow** together imply that $\mathcal{U}$ also satisfies

Fun If $X \in \mathcal{U}$ and $Y \in \mathcal{U}$, then $X \to Y \in \mathcal{U}$.

By iterating **Prod**, one has that the cartesian product of any finite, non-zero number of sets in $\mathcal{U}$ is again in $\mathcal{U}$. $\mathcal{U}$ also contains the cartesian product of no sets, which is to say that it contains a one-element set (by virtue of **Sub** applied to any set in $\mathcal{U}$—**Infty** guarantees there is one); for definiteness, a particular one-element set will be singled out.

Unit $\mathcal{U}$ contains a distinguished one-element set $1 = \{0\}$.

Similarly, because of **Sub** and **Infty**, $\mathcal{U}$ contains two-element sets, one of which will be singled out.

Bool $\mathcal{U}$ contains a distinguished two-element set $2 = \{0, 1\}$.

The above assumptions on $\mathcal{U}$ are weaker than those imposed on a universe of sets by the axioms of Zermelo-Fraenkel set theory with the Axiom of Choice (ZFC), principally because $\mathcal{U}$ is not required to satisfy any form of the Axiom of Replacement. Indeed, it is possible to prove the existence of a set $\mathcal{U}$ with the above properties from the axioms of ZFC. (For example one could take $\mathcal{U}$ to consist of all non-empty sets in the von Neumann cumulative hierarchy formed before stage $\omega + \omega$.) Thus, as with many other pieces of mathematics, it is possible in principal to give a completely formal version within ZFC set theory of the semantics of the HOL logic to be given below.

9.2 Types

The types of the HOL logic are expressions that denote sets (in the universe $\mathcal{U}$). Following tradition, σ, possibly decorated with subscripts or primes, is used to range over arbitrary types.

There are four kinds of types in the HOL logic. These can be described informally by the following BNF grammar, in which α ranges over type variables, c ranges over atomic types and op ranges over type operators.

$$\sigma \quad ::= \quad \underbrace{\alpha}_{\text{type variables}} \quad | \quad \underbrace{c}_{\text{atomic types}} \quad | \quad \underbrace{(\sigma_1, \ldots, \sigma_n)op}_{\text{compound types}} \quad | \quad \underbrace{\sigma_1 \rightarrow \sigma_2}_{\substack{\text{function types} \\ (\text{domain } \sigma_1, \text{ range } \sigma_2)}}$$

In more detail, the four kinds of types are as follows.

1. **Type variables:** these stand for arbitrary sets in the universe. In Church's original formulation of simple type theory, type variables are part of the meta-language and are used to range over object language types. Proofs containing type variables were understood as proof schemes (i.e. families of proofs). To support such proof schemes *within* the HOL logic, type variables have been added to the object language type system.[2]

2. **Atomic types:** these denote fixed sets in the universe. Each theory determines a particular collection of atomic types. For example, the standard atomic types *bool* and *ind* denote, respectively, the distinguished two-element set 2 and the distinguished infinite set I.

3. **Compound types:** These have the form $(\sigma_1, \ldots, \sigma_n)op$, where σ_1, ..., σ_n are the argument types and *op* is a *type operator* of arity n. Type operators denote operations for constructing sets. The type $(\sigma_1, \ldots, \sigma_n)op$ denotes the set resulting from applying the operation denoted by *op* to the sets denoted by σ_1, ..., σ_n. For example, *list* is a type operator with arity 1. It denotes the operation of forming all finite lists of elements from a given set. Another example is the type operator *prod* of arity 2 which denotes the cartesian product operation. The type $(\sigma_1, \sigma_2)prod$ is written as $\sigma_1 \times \sigma_2$.

4. **Function types:** If σ_1 and σ_2 are types, then $\sigma_1 \rightarrow \sigma_2$ is the function type with *domain* σ_1 and *range* σ_2. It denotes the set of all (total) functions from the set denoted by its domain to the set denoted by its range. (In the literature $\sigma_1 \rightarrow \sigma_2$ is written without the arrow and backwards—i.e. as $\sigma_2\sigma_1$.) Note that syntactically $\rightarrow$ is simply a distinguished type operator of arity 2 written with infix notation. It is singled out in the definition of HOL types because it will always denote the same operation in any model of a HOL theory—in contrast to the other type operators which may be interpreted differently in different models. (See Section 9.2.2.)

It turns out to be convenient to identify atomic types with compound types constructed with 0-ary type operators. For example, the atomic type *bool* of truth-values can be regarded as being an abbreviation for $()bool$. This identification will be made in the

[2]This technique was invented by Robin Milner for the object logic PPλ of his LCF system.

technical details that follow, but in the informal presentation atomic types will continue to be distinguished from compound types, and $()c$ will still be written as c.

9.2.1 Type structures

The term 'type constant' is used to cover both atomic types and type operators. It is assumed that an infinite set TyNames of the *names of type constants* is given. The greek letter ν is used to range over arbitrary members of TyNames, c will continue to be used to range over the names of atomic types (i.e. 0-ary type constants), and op is used to range over the names of type operators (i.e. n-ary type constants, where $n > 0$).

It is assumed that an infinite set TyVars of *type variables* is given. Greek letters $\alpha, \beta, \ldots$, possibly with subscripts or primes, are used to range over Tyvars. The sets TyNames and TyVars are assumed disjoint.

A *type structure* is a set Ω of type constants. A *type constant* is a pair (ν, n) where $\nu \in$ TyNames is the name of the constant and n is its arity. Thus $\Omega \subseteq$ TyNames $\times$ N (where N is the set of natural numbers). It is assumed that no two distinct type constants have the same name, i.e. whenever $(\nu, n_1) \in \Omega$ and $(\nu, n_2) \in \Omega$, then $n_1 = n_2$.

The set Types_Ω of types over a structure Ω can now be defined as the smallest set such that:

- TyVars $\subseteq \mathsf{Types}_\Omega$.
- If $(\nu, 0) \in \Omega$ then $()\nu \in \mathsf{Types}_\Omega$.
- If $(\nu, n) \in \Omega$ and $\sigma_i \in \mathsf{Types}_\Omega$ for $1 \leq i \leq n$, then $(\sigma_1, \ \ldots \ , \sigma_n)\nu \in \mathsf{Types}_\Omega$.
- If $\sigma_1 \in \mathsf{Types}_\Omega$ and $\sigma_2 \in \mathsf{Types}_\Omega$ then $\sigma_1 \to \sigma_2 \in \mathsf{Types}_\Omega$.

The type operator $\to$ is assumed to associate to the right, so that

$$\sigma_1 \to \sigma_2 \to \ldots \to \sigma_n \to \sigma$$

abbreviates

$$\sigma_1 \to (\sigma_2 \to \ldots \to (\sigma_n \to \sigma) \ldots)$$

The notation $tyvars(\sigma)$ is used to denote the set of type variables occurring in σ.

9.2.2 Semantics of types

A *model* M of a type structure Ω is specified by giving for each type constant (ν, n) an n-ary function

$$M(\nu) : \mathcal{U}^n \longrightarrow \mathcal{U}$$

Thus given sets $X_1, \ldots, X_n$ in the universe $\mathcal{U}$, $M(\nu)(X_1, \ldots, X_n)$ is also a set in the universe. In case $n = 0$, this amounts to specifying an element $M(\nu) \in \mathcal{U}$ for the atomic type ν.

Types containing no type variables are called *monomorphic*, whereas those that do contain type variables are called *polymorphic*. What is the meaning of a polymorphic type? One can only say what set a polymorphic type denotes once one has instantiated its type variables to particular sets. So its overall meaning is not a single set, but is rather a set-valued function, $\mathcal{U}^n \longrightarrow \mathcal{U}$, assigning a set for each particular assignment of sets to the relevant type variables. The arity n corresponds to the number of type variables involved. It is convenient in this connection to be able to consider a type variable to be involved in the semantics of a type σ whether or not it actually occurs in σ, leading to the notion of a type-in-context.

A *type context*, αs, is simply a finite (possibly empty) list of *distinct* type variables $\alpha_1, \ldots, \alpha_n$. A *type-in-context* is a pair, written $\alpha s.\sigma$, where αs is a type context, σ is a type (over some given type structure) and all the type variables occurring in σ appear somewhere in the list αs. The list αs may also contain type variables which do not occur in σ.

For each σ there are minimal contexts αs for which $\alpha s.\sigma$ is a type-in-context, which only differ by the order in which the type variables of σ are listed in αs. In order to select one such context, let us assume that TyVars comes with a fixed total order and define the *canonical* context of the type σ to consist of exactly the type variables it contains, listed in order.[3]

Let M be a model of a type structure Ω. For each type-in-context $\alpha s.\sigma$ over Ω, define a function

$$[\![\alpha s.\sigma]\!]_M : \mathcal{U}^n \longrightarrow \mathcal{U}$$

(where n is the length of the context) by induction on the structure of σ as follows.

- If σ is a type variable, it must be α_i for some unique $i = 1, \ldots, n$ and then $[\![\alpha s.\sigma]\!]_M$ is the ith projection function, which sends $(X_1, \ldots, X_n) \in \mathcal{U}^n$ to $X_i \in \mathcal{U}$.
- If σ is a function type $\sigma_1 \to \sigma_2$, then $[\![\alpha s.\sigma]\!]_M$ sends $Xs \in \mathcal{U}^n$ to the set of all functions from $[\![\alpha s.\sigma_1]\!]_M(Xs)$ to $[\![\alpha s.\sigma_2]\!]_M(Xs)$. (This makes use of the property **Fun** of $\mathcal{U}$.)
- If σ is a compound type $(\sigma_1, \ldots, \sigma_m)\nu$, then $[\![\alpha s.\sigma]\!]_M$ sends Xs to $M(\nu)(S_1, \ldots, S_m)$ where each S_j is $[\![\alpha s.\sigma_j]\!]_M(Xs)$.

One can now define the meaning of a type σ in a model M to be the function

$$[\![\sigma]\!]_M : \mathcal{U}^n \longrightarrow \mathcal{U}$$

[3] It is possible to work with unordered contexts, specified by finite sets rather than lists, but we choose not to do that since it mildly complicates the definition of the semantics to be given below.

given by $[\![\alpha s.\sigma]\!]_M$, where αs is the canonical context of σ. If σ is monomorphic, then $n = 0$ and $[\![\sigma]\!]_M$ can be identified with the element $[\![\sigma]\!]_M()$ of $\mathcal{U}$. When the particular model M is clear from the context, $[\![\_]\!]_M$ will be written $[\![\_]\!]$.

To summarize, given a model in $\mathcal{U}$ of a type structure Ω, the semantics interprets monomorphic types over Ω as sets in $\mathcal{U}$ and more generally, interprets polymorphic types involving n type variables as n-ary functions $\mathcal{U}^n \longrightarrow \mathcal{U}$ on the universe. Function types are interpreted by full function sets.

Examples Suppose that Ω contains a type constant $(b, 0)$ and that the model M assigns the set 2 to b. Then:

1. $[\![b{\to}b{\to}b]\!] = 2{\to}2{\to}2 \in \mathcal{U}$.
2. $[\![(\alpha{\to}b){\to}\alpha]\!] : \mathcal{U} \longrightarrow \mathcal{U}$ is the function sending $X \in \mathcal{U}$ to $(X{\to}2){\to}X \in \mathcal{U}$.
3. $[\![\alpha, \beta.(\alpha{\to}b){\to}\alpha]\!] : \mathcal{U}^2 \longrightarrow \mathcal{U}$ is the function sending $(X, Y) \in \mathcal{U}^2$ to $(X{\to}2){\to}X \in \mathcal{U}$.

Remark A more traditional approach to the semantics would involve giving meanings to types in the presence of 'environments' assigning sets in $\mathcal{U}$ to all type variables. The use of types-in-contexts is almost the same as using partial environments with finite domains—it is just that the context ties down the admissible domain to a particular finite (ordered) set of type variables. At the level of types there is not much to choose between the two approaches. However for the syntax and semantics of terms to be given below, where there is a dependency both on type variables and on individual variables, the approach used here seems best.

9.2.3 Instances and substitution

If σ and $\tau_1, \ldots, \tau_n$ are types over a type structure Ω,

$$\sigma[\tau_1, \ldots, \tau_p/\beta_1, \ldots, \beta_p]$$

will denote the type resulting from the simultaneous substitution for each $i = 1, \ldots, p$ of τ_i for the type variable β_i in σ. The resulting type is called an *instance* of σ. The following lemma about instances will be useful later; it is proved by induction on the structure of σ.

Lemma 1 *Suppose that σ is a type containing distinct type variables $\beta_1, \ldots, \beta_p$ and that $\sigma' = \sigma[\tau_1, \ldots, \tau_n/\beta_1, \ldots, \beta_p]$ is an instance of σ. Then the types $\tau_1, \ldots, \tau_p$ are uniquely determined by σ and σ'.*

We also need to know how the semantics of types behaves with respect to substitution:

Lemma 2 *Given types-in-context $\beta s.\sigma$ and $\alpha s.\tau_i$ ($i = 1, \ldots, p$, where p is the length of βs), let σ' be the instance $\sigma[\tau s/\beta s]$. Then $\alpha s.\sigma'$ is also a type-in-context and its meaning*

in any model M is related to that of $\beta s.\sigma$ as follows. For all $Xs \in \mathcal{U}^n$ (where n is the length of αs)

$$[\![\alpha s.\sigma']\!](Xs) = [\![\beta s.\sigma]\!]([\![\alpha s.\tau_1]\!](Xs), \ldots, [\![\alpha s.\tau_p]\!](Xs))$$

Once again, the lemma can be proved by induction on the structure of σ.

9.3 Terms

The terms of the HOL logic are expressions that denote elements of the sets denoted by types. The meta-variable t is used to range over arbitrary terms, possibly decorated with subscripts or primes.

There are four kinds of terms in the HOL logic. These can be described approximately by the following BNF grammar, in which x ranges over variables and c ranges over constants.

$$t \quad ::= \quad \underset{\text{variables}}{x} \quad | \quad \underset{\text{constants}}{\mathsf{c}} \quad | \quad \underbrace{t\ t'}_{\text{function applications}} \quad | \quad \underbrace{\lambda x.\ t}_{\lambda\text{-abstractions}}$$

(function t, argument t')

Informally, a λ-term $\lambda x.\ t$ denotes a function $v \mapsto t[v/x]$, where $t[v/x]$ denotes the result of substituting v for x in t. An application $t\ t'$ denotes the result of applying the function denoted by t to the value denoted by t'. This will be made more precise below.

The BNF grammar just given omits mention of types. In fact, each term in the HOL logic is associated with a unique type. The notation t_σ is traditionally used to range over terms of type σ. A more accurate grammar of terms is:

$$t_\sigma \quad ::= \quad x_\sigma \quad | \quad \mathsf{c}_\sigma \quad | \quad (t_{\sigma' \to \sigma}\ t'_{\sigma'})_\sigma \quad | \quad (\lambda x_{\sigma_1}.\ t_{\sigma_2})_{\sigma_1 \to \sigma_2}$$

In fact, just as the definition of types was relative to a particular type structure Ω, the formal definition of terms is relative to a given collection of typed constants over Ω. Assume that an infinite set Names of names is given. A *constant* over Ω is a pair (c, σ), where $\mathsf{c} \in \mathsf{Names}$ and $\sigma \in \mathsf{Types}_\Omega$. A *signature* over Ω is just a set Σ_Ω of such constants.

The set $\mathsf{Terms}_{\Sigma_\Omega}$ of terms over Σ_Ω is defined to be the smallest set closed under the following rules of formation:

1. **Constants:** If $(\mathsf{c}, \sigma) \in \Sigma_\Omega$ and $\sigma' \in \mathsf{Types}_\Omega$ is an instance of σ, then $(\mathsf{c}, \sigma') \in \mathsf{Terms}_{\Sigma_\Omega}$. Terms formed in this way are called *constants* and are written $\mathsf{c}_{\sigma'}$.

2. **Variables:** If $x \in \mathsf{Names}$ and $\sigma \in \mathsf{Types}_\Omega$, then $\mathtt{var}\ x_\sigma \in \mathsf{Terms}_{\Sigma_\Omega}$. Terms formed in this way are called *variables*. The marker `var` is purely a device to distinguish

variables from constants with the same name. A variable $\texttt{var}\ x_\sigma$ will usually be written as x_σ, if it is clear from the context that x is a variable rather than a constant.

3. **Function applications:** If $t_{\sigma'\to\sigma} \in \mathsf{Terms}_{\Sigma_\Omega}$ and $t'_{\sigma'} \in \mathsf{Terms}_{\Sigma_\Omega}$, then $(t_{\sigma'\to\sigma}\ t'_{\sigma'})_\sigma \in \mathsf{Terms}_{\Sigma_\Omega}$. (Terms formed in this way are sometimes called *combinations*.)

4. **λ-Abstractions:** If $\texttt{var}\ x_{\sigma_1} \in \mathsf{Terms}_{\Sigma_\Omega}$ and $t_{\sigma_2} \in \mathsf{Terms}_{\Sigma_\Omega}$, then $(\lambda x_{\sigma_1}.\ t_{\sigma_2})_{\sigma_1\to\sigma_2} \in \mathsf{Terms}_{\Sigma_\Omega}$.

Note that it is possible for constants and variables to have the same name. It is also possible for different variables to have the same name, if they have different types.

The type subscript on a term may be omitted if it is clear from the structure of the term or the context in which it occurs what its type must be.

Function application is assumed to associate to the left, so that $t\ t_1\ t_2\ \ldots\ t_n$ abbreviates $(\ \ldots\ ((t\ t_1)\ t_2)\ \ldots\ t_n)$.

The notation $\lambda x_1\ x_2\ \cdots\ x_n.\ t$ abbreviates $\lambda x_1.\ (\lambda x_2.\ \cdots\ (\lambda x_n.\ t)\ \cdots)$.

A term is called polymorphic if it contains a type variable. Otherwise it is called monomorphic. Note that a term t_σ may be polymorphic even though σ is monomorphic—for example, $(f_{\alpha\to b}\ x_\alpha)_b$, where b is an atomic type. The expression $tyvars(t_\sigma)$ denotes the set of type variables occurring in t_σ.

An occurrence of a variable x_σ is called *bound* if it occurs within the scope of a textually enclosing λx_σ, otherwise the occurrence is called *free*. Note that λx_σ does not bind $x_{\sigma'}$ if $\sigma \neq \sigma'$. A term in which all occurrences of variables are bound is called *closed*.

9.3.1 Terms-in-context

A *context* αs,xs consists of a type context αs together with a list $xs = x_1, \ldots, x_m$ of distinct variables whose types only contain type variables from the list αs.

The condition that xs contains *distinct* variables needs some comment. Since a variable is specified by both a name and a type, it is permitted for xs to contain repeated names, so long as different types are attached to the names. This aspect of the syntax means that one has to proceed with caution when defining the meaning of type variable instantiation, since instantiation may cause variables to become equal 'accidentally': see Section 9.3.3.

A *term-in-context* αs,$xs.t$ consists of a context together with a term t satisfying the following conditions.

- αs contains any type variable that occurs in xs and t.
- xs contains any variable that occurs freely in t.
- xs does not contain any variable that occurs bound in t.

The context αs,xs may contain (type) variables which do not appear in t. Note that the combination of the second and third conditions implies that a variable cannot have both free and bound occurrences in t. For an arbitrary term, there is always an α-equivalent term which satisfies this condition, obtained by renaming the bound variables as necessary.[4] In the semantics of terms to be given below we will restrict attention to such terms. Then the meaning of an arbitrary term is taken to be the meaning of some α-variant of it having no variable both free and bound. (The semantics will equate α-variants, so it does not matter which is chosen.) Evidently for such a term there is a minimal context αs,xs, unique up to the order in which variables are listed, for which αs,$xs.t$ is a term-in-context. As for type variables, we will assume given a fixed total order on variables. Then the unique minimal context with variables listed in order will be called the *canonical* context of the term t.

9.3.2 Semantics of terms

Let Σ_Ω be a signature over a type structure Ω (see Section 9.3). A *model* M of Σ_Ω is specified by a model of the type structure plus for each constant $(\mathsf{c}, \sigma) \in \Sigma_\Omega$ an element

$$M(\mathsf{c}, \sigma) \in \prod_{Xs \in \mathcal{U}^n} [\![\sigma]\!]_M(Xs)$$

of the indicated cartesian product, where n is the number of type variables occurring in σ. In other words $M(\mathsf{c}, \sigma)$ is a (dependently typed) function assigning to each $Xs \in \mathcal{U}^n$ an element of $[\![\sigma]\!]_M(Xs)$. In the case that $n = 0$ (so that σ is monomorphic), $[\![\sigma]\!]_M$ was identified with a set in $\mathcal{U}$ and then $M(c, \sigma)$ can be identified with an element of that set.

The meaning of HOL terms in such a model will now be described. The semantics interprets closed terms involving no type variables as elements of sets in $\mathcal{U}$ (the particular set involved being derived from the type of the term as in Section 9.2.2). More generally, if the closed term involves n type variables then it is interpreted as an element of a product $\prod_{Xs \in \mathcal{U}^n} Y(Xs)$, where the function $Y : \mathcal{U}^n \longrightarrow \mathcal{U}$ is derived from the type of the term (in a type context derived from the term). Thus the meaning of the term is a (dependently typed) function which, when applied to any meanings chosen for the type variables in the term, yields a meaning for the term as an element of a set in $\mathcal{U}$. On the other hand, if the term involves m free variables but no type variables, then it is interpreted as a function $Y_1 \times \cdots \times Y_m \to Y$ where the sets $Y_1, \ldots, Y_m$ in $\mathcal{U}$ are the interpretations of the types of the free variables in the term and the set $Y \in \mathcal{U}$ is the interpretation of the type of the term; thus the meaning of the term is a function which, when applied to any meanings chosen for the free variables in the term, yields a meaning for the term. Finally, the most

[4] Recall that two terms are said to be α-equivalent if they differ only in the names of their bound variables.

general case is of a term involving n type variables and m free variables: it is interpreted as an element of a product

$$\prod_{Xs \in \mathcal{U}^n} Y_1(Xs) \times \cdots \times Y_m(Xs) \to Y(Xs)$$

where the functions $Y_1, \ldots, Y_m, Y : \mathcal{U}^n \longrightarrow \mathcal{U}$ are determined by the types of the free variables and the type of the term (in a type context derived from the term).

More precisely, given a term-in-context $\alpha s,xs.t$ over Σ_Ω suppose

- t has type τ
- $xs = x_1, \ldots, x_m$ and each x_j has type σ_j
- $\alpha s = \alpha_1, \ldots, \alpha_n$.

Then since $\alpha s,xs.t$ is a term-in-context, $\alpha s.\tau$ and $\alpha s.\sigma_j$ are types-in-context, and hence give rise to functions $[\![\alpha s.\tau]\!]_M$ and $[\![\alpha s.\sigma_j]\!]_M$ from $\mathcal{U}^n$ to $\mathcal{U}$ as in section 9.2.2. The meaning of $\alpha s,xs.t$ in the model M will be given by an element

$$[\![\alpha s,xs.t]\!]_M \in \prod_{Xs \in \mathcal{U}^n} \left(\prod_{j=1}^{m} [\![\alpha s.\sigma_j]\!]_M(Xs) \right) \to [\![\alpha s.\tau]\!]_M(Xs).$$

In other words, given

$$\begin{aligned} Xs &= (X_1, \ldots, X_n) \in \mathcal{U}^n \\ ys &= (y_1, \ldots, y_m) \in [\![\alpha s.\sigma_1]\!]_M(Xs) \times \cdots \times [\![\alpha s.\sigma_m]\!]_M(Xs) \end{aligned}$$

one gets an element $[\![\alpha s,xs.t]\!]_M(Xs)(ys)$ of $[\![\alpha s.\tau]\!]_M(Xs)$. The definition of $[\![\alpha s,xs.t]\!]_M$ proceeds by induction on the structure of the term t, as follows. (As before, the subscript M will be dropped from the semantic brackets $[\![\_]\!]$ when the particular model involved is clear from the context.)

- If t is a variable, it must be x_j for some unique $j = 1, \ldots, m$, so $\tau = \sigma_j$ and then $[\![\alpha s,xs.t]\!](Xs)(ys)$ is defined to be y_j.
- Suppose t is a constant $\mathsf{c}_{\sigma'}$, where $(\mathsf{c}, \sigma) \in \Sigma_\Omega$ and σ' is an instance of σ. Then by Lemma 1 of 9.2.3, $\sigma' = \sigma[\tau_1, \ldots, \tau_p/\beta_1, \ldots, \beta_p]$ for uniquely determined types $\tau_1, \ldots, \tau_p$ (where $\beta_1, \ldots, \beta_p$ are the type variables occurring in σ). Then define $[\![\alpha s,xs.t]\!](Xs)(ys)$ to be $M(\mathsf{c}, \sigma)([\![\alpha s.\tau_1]\!](Xs), \ldots, [\![\alpha s.\tau_p]\!](Xs))$, which is an element of $[\![\alpha s.\tau]\!](Xs)$ by Lemma 2 of 9.2.3 (since τ is σ').
- Suppose t is a function application term $(t_1\ t_2)$ where t_1 is of type $\tau' \to \tau$ and t_2 is of type τ'. Then $f = [\![\alpha s,xs.t_1]\!](Xs)(ys)$, being an element of $[\![\alpha s.\tau' \to \tau]\!](Xs)$, is a function from the set $[\![\alpha s.\tau']\!](Xs)$ to the set $[\![\alpha s.\tau]\!](Xs)$ which one can apply to the element $y = [\![\alpha s,xs.t_2]\!](Xs)(ys)$. Define $[\![\alpha s,xs.t]\!](Xs)(ys)$ to be $f(y)$.

- Suppose t is the abstraction term $\lambda x.t_2$where x is of type τ_1 and t_2 of type τ_2. Thus $\tau = \tau_1 \to \tau_2$ and $[\![\alpha s.\tau]\!](Xs)$ is the function set $[\![\alpha s.\tau_1]\!](Xs) \to [\![\alpha s.\tau_2]\!](Xs)$. Define $[\![\alpha s,xs.t]\!](Xs)(ys)$ to be the element of this set which is the function sending $y \in [\![\alpha s.\tau_1]\!](Xs)$ to $[\![\alpha s,xs,x.t_2]\!](Xs)(ys,y)$. (Note that since $\alpha s,xs.t$ is a term-in-context, by convention the bound variable x does not occur in xs and thus $\alpha s,xs,x.t_2$ is also a term-in-context.)

Now define the meaning of a term t_τ in a model M to be the dependently typed function

$$[\![t_\tau]\!] \in \prod_{Xs \in \mathcal{U}^n} \left(\prod_{j=1}^{m} [\![\alpha s.\sigma_j]\!](Xs) \right) \to [\![\alpha s.\tau]\!](Xs)$$

given by $[\![\alpha s,xs.t_\tau]\!]$, where $\alpha s,xs$ is the canonical context of t_τ. So n is the number of type variables in t_τ, αs is a list of those type variables, m is the number of ordinary variables occurring freely in t_τ (assumed to be distinct from the bound variables of t_τ) and the σ_j are the types of those variables. (It is important to note that the list αs, which is part of the canonical context of t, may be strictly bigger than the canonical type contexts of σ_j or τ. So it would not make sense to write just $[\![\sigma_j]\!]$ or $[\![\tau]\!]$ in the above definition.)

If t_τ is a closed term, then $m = 0$ and for each $Xs \in \mathcal{U}^n$ one can identify $[\![t_\tau]\!]$ with the element $[\![t_\tau]\!](Xs)() \in [\![\alpha s.\tau]\!](Xs)$. So for closed terms one gets

$$[\![t_\tau]\!] \in \prod_{Xs \in \mathcal{U}^n} [\![\alpha s.\tau]\!](Xs)$$

where αs is the list of type variables occurring in t_τ and n is the length of that list. If moreover, no type variables occur in t_τ, then $n = 0$ and $[\![t_\tau]\!]$ can be identified with the element $[\![t_\tau]\!]()$ of the set $[\![\tau]\!] \in \mathcal{U}$.

The semantics of terms appears somewhat complicated because of the possible dependency of a term upon both type variables and ordinary variables. Examples of how the definition of the semantics works in practice can be found in Section 10.4.2, where the meaning of several terms denoting logical constants is given.

9.3.3 Substitution

Since terms may involve both type variables and ordinary variables, there are two different operations of substitution on terms which have to be considered—substitution of types for type variables and substitution of terms for variables.

Substituting types for type variables in terms

Suppose t is a term, with canonical context $\alpha s,xs$ say, where $\alpha s = \alpha_1, \ldots, \alpha_n$, $xs = x_1, \ldots, x_m$ and where for $j = 1, \ldots, m$ the type of the variable x_j is σ_j. If $\alpha s'.\tau_i$ $(i = 1, \ldots, n)$ are types-in-context, then substituting the types τ_i for the type variables α_i in the list xs, one obtains a new list of variables xs'. Thus the jth entry of xs' has type $\sigma'_j = \sigma_j[\tau s/\alpha s]$. Only substitutions with the following property will be considered.

> In instantiating the type variables αs with the types τs, no two distinct variables in the list xs become equal in the list xs'.[5]

This condition ensures that $\alpha s', xs'$ really is a context. Then one obtains a new term-in-context $\alpha s'$,$xs'.t'$ by substituting the types $\tau s = \tau_1, \ldots, \tau_n$ for the type variables αs in t (with suitable renaming of bound occurrences of variables to make them distinct from the variables in xs'). The notation

$$t[\tau s/\alpha s]$$

is used for the term t'.

Lemma 3 *The meaning of $\alpha s'$,$xs'.t'$ in a model is related to that of t as follows. For all $Xs' \in \mathcal{U}^{n'}$ (where n' is the length of $\alpha s'$)*

$$[\![\alpha s', xs'.t']\!](Xs') = [\![t]\!]([\![\alpha s'.\tau_1]\!](Xs'), \ldots, [\![\alpha s'.\tau_n]\!](Xs')).$$

Lemma 2 in 9.2.3 is needed to see that both sides of the above equation are elements of the same set of functions. The validity of the equation is proved by induction on the structure of the term t.

Substituting terms for variables in terms

Suppose t is a term, with canonical context αs,xs say, where $\alpha s = \alpha_1, \ldots, \alpha_n$, $xs = x_1, \ldots, x_m$ and where for $j = 1, \ldots, m$ the type of the variable x_j is σ_j. If one has terms-in-context αs,$xs'.t_j$ for $j = 1, \ldots, m$ with t_j of the same type as x_j, say σ_j, then one obtains a new term-in-context αs,$xs'.t''$ by substituting the terms $ts = t_1, \ldots, t_m$ for the variables xs in t (with suitable renaming of bound occurrences of variables to prevent the free variables of the t_j becoming bound after substitution). The notation

$$t[ts/xs]$$

is used for the term t''.

Lemma 4 *The meaning of αs,$xs'.t''$ in a model is related to that of t as follows. For all $Xs \in \mathcal{U}^n$ and all $ys' \in [\![\alpha s.\sigma'_1]\!] \times \cdots \times [\![\alpha s.\sigma'_{m'}]\!]$ (where σ'_j is the type of x'_j)*

$$[\![\alpha s, xs'.t'']\!](Xs)(ys') = [\![t]\!](Xs)([\![\alpha s, xs'.t_1]\!](Xs)(ys'), \ldots, [\![\alpha s, xs'.t_m]\!](Xs)(ys'))$$

Once again, this result is proved by induction on the structure of the term t.

[5] Such an identification of variables could occur if the variables had the same name component and their types became equal on instantiation.

9.4 Standard notions

Up to now the syntax of types and terms has been very general. To represent the standard formulas of logic it is necessary to impose some specific structure. In particular, every type structure must contain an atomic type *bool* which is intended to denote the distinguished two-element set $2 \in \mathcal{U}$, regarded as a set of truth-values. Logical formulas are then identified with terms of type *bool*. In addition, various logical constants are assumed to be in all signatures. These requirements are formalized by defining the notion of a standard signature.

9.4.1 Standard type structures

A type structure Ω is *standard* if it contains the atomic types *bool* (of booleans or truth-values) and *ind* (of individuals). (In the literature, the symbol o is often used instead of *bool* and ι instead of *ind*.)

A model M of Ω is *standard* if $M(bool)$ and $M(ind)$ are respectively the distinguished sets 2 and I in the universe $\mathcal{U}$.

It will be assumed from now on that type structures and their models are standard.

9.4.2 Standard signatures

A signature Σ_Ω is *standard* if it contains the following three primitive constants:

$\Rightarrow_{bool \to bool \to bool}$

$=_{\alpha \to \alpha \to bool}$

$\varepsilon_{(\alpha \to bool) \to \alpha}$

The intended interpretation of these constants is that $\Rightarrow$ denotes implication, $=_{\sigma \to \sigma \to bool}$ denotes equality on the set denoted by σ, and $\varepsilon_{(\sigma \to bool) \to \sigma}$ denotes a choice function on the set denoted by σ. More precisely, a model M of Σ_Ω will be called *standard* if

- $M(\Rightarrow, bool \to bool \to bool) \in (2 \to 2 \to 2)$ is the standard implication function, sending $b, b' \in 2$ to

$$(b \Rightarrow b') = \begin{cases} 0 & \text{if } b = 1 \text{ and } b' = 0 \\ 1 & \text{otherwise} \end{cases}$$

- $M(=, \alpha \to \alpha \to bool) \in \prod_{X \in \mathcal{U}} . X \to X \to 2$ is the function assigning to each $X \in \mathcal{U}$ the equality test function, sending $x, x' \in X$ to

$$(x =_X x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases}$$

- $M(\varepsilon, (\alpha \to bool) \to \alpha) \in \prod_{X \in \mathcal{U}} .(X \to 2) \to X$ is the function assigning to each $X \in \mathcal{U}$ the choice function sending $f \in (X \to 2)$ to

$$\mathrm{ch}_X(f) = \begin{cases} \mathrm{ch}(f^{-1}\{1\}) & \text{if } f^{-1}\{1\} \neq \emptyset \\ \mathrm{ch}(X) & \text{otherwise} \end{cases}$$

  where $f^{-1}\{1\} = \{x \in X : f(x) = 1\}$. (Note that $f^{-1}\{1\}$ is in $\mathcal{U}$ when it is non-empty, by the property **Sub** of the universe $\mathcal{U}$ given in Section 9.1. The function ch is given by property **Choice**.)

It will be assumed from now on that signatures and their models are standard.

Remark This particular choice of primitive constants is arbitrary. The standard collection of logical constants includes T ('true'), F ('false'), $\Rightarrow$ ('implies'), $\wedge$ ('and'), $\vee$ ('or'), $\neg$ ('not'), $\forall$ ('for all'), $\exists$ ('there exists'), $=$ ('equals'), ι ('the'), and ε ('a'). This set is redundant, since it can be defined (in a sense explained in Section 10.5.1) from various subsets. In practice, it is necessary to work with the full set of logical constants, and the particular subset taken as primitive is not important. The interested reader can explore this topic further by reading Andrews' book [1] and the references it contains.

Terms of type *bool* are called *formulas*.

The following notational abbreviations are used:

Notation	Meaning
$t_\sigma = t'_\sigma$	$=_{\sigma \to \sigma \to bool}\ t_\sigma\ t'_\sigma$
$t \Rightarrow t'$	$\Rightarrow_{bool \to bool \to bool}\ t_{bool}\ t'_{bool}$
$\varepsilon x_\sigma .\ t$	$\varepsilon_{(\sigma \to bool) \to \sigma}(\lambda x_\sigma .\ t)$

These notations are special cases of general abbreviatory conventions supported by the HOL system. The first two are infixes and the third is a binder (see Section 11.4.3).