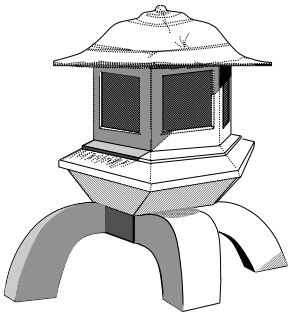
[For HOL Kananaskis-14] February 3, 2021

The HOL System
LOGIC



Preface

This volume contains the description of the HOL system's logic. It is one of four volumes making up the documentation for HOL:

(i) *LOGIC*: a formal description of the higher order logic implemented by the HOL system;

(ii) *TUTORIAL*: a tutorial introduction to HOL, with case studies;

(iii) *DESCRIPTION*: a detailed user's guide for the HOL system;

(iv) *REFERENCE*: the reference manual for HOL.

These four documents will be referred to by the short names (in small slanted capitals) given above.

This document, *LOGIC*, serves as a formal definition of higher order logic in terms of a set-theoretic semantics. This material was written by Andrew Pitts in 1991, and was originally part of *DESCRIPTION*. Because this logic is shared with other theorem-proving systems (HOL Light, ProofPower), and is similar to that implemented in Isabelle, where it is called Isabelle/HOL, it is now presented in its own manual.

The HOL system is designed to support interactive theorem proving in higher order logic (hence the acronym 'HOL'). To this end, the formal logic is interfaced to a general purpose programming language (ML, for meta-language) in which terms and theorems of the logic can be denoted, proof strategies expressed and applied, and logical theories developed. The version of higher order logic used in HOL is predicate calculus with terms from the typed lambda calculus (*i.e.* simple type theory). This was originally developed as a foundation for mathematics [2]. The primary application area of HOL was initially intended to be the specification and verification of hardware designs. (The use of higher order logic for this purpose was first advocated by Keith Hanna [4].) However, the logic does not restrict applications to hardware; HOL has been applied to many other areas.

Thus, this document describes the theoretical underpinnings of the HOL system, and presents it abstractly.

The approach to mechanizing formal proof used in HOL is due to Robin Milner [3], who also headed the team that designed and implemented the language ML. That work centred on a system called LCF (logic for computable functions), which was intended for

interactive automated reasoning about higher order recursively defined functions. The interface of the logic to the meta-language was made explicit, using the type structure of ML, with the intention that other logics eventually be tried in place of the original logic. The HOL system is a direct descendant of LCF; this is reflected in everything from its structure and outlook to its incorporation of ML, and even to parts of its implementation. Thus HOL satisfies the early plan to apply the LCF methodology to other logics.

The original LCF was implemented at Edinburgh in the early 1970's, and is now referred to as 'Edinburgh LCF'. Its code was ported from Stanford Lisp to Franz Lisp by Gérard Huet at INRIA, and was used in a French research project called 'Formel'. Huet's Franz Lisp version of LCF was further developed at Cambridge by Larry Paulson, and became known as 'Cambridge LCF'. The HOL system is implemented on top of an early version of Cambridge LCF and consequently many features of both Edinburgh and Cambridge LCF were inherited by HOL. For example, the axiomatization of higher order logic used is not the classical one due to Church, but an equivalent formulation influenced by LCF.

An enhanced and rationalized version of HOL, called HOL88, was released (in 1988), after the original HOL system had been in use for several years. HOL90 (released in 1990) was a port of HOL88 to SML [5] by Konrad Slind at the University of Calgary. It has been further developed through the 1990's. HOL 4 is the latest version of HOL, and is also implemented in SML; it features a number of novelties compared to its predecessors. HOL 4 is also the supported version of the system for the international HOL community.

We have retroactively decided to number HOL implementations in the following way

1. HOL88 and earlier: implementations based on a Lisp substrate, with Classic ML.
2. HOL90: implementations in Standard ML, principally using the SML/NJ implementation.
3. HOL98 (Athabasca and Taupo releases): implementations using Moscow ML, and with a new library and theory mechanism.
4. HOL (Kananaskis releases)

Therefore, with HOL 4, we do away with the habit of associating implementations with year numbers. Individual releases within HOL 4 will retain the *lake-number* naming scheme.

In this document, the acronym 'HOL' refers to both the interactive theorem proving system and to the version of higher order logic that the system supports; where there is serious ambiguity, the former is called 'the HOL system' and the latter 'the HOL logic'.

Acknowledgements

The bulk of HOL is based on code written by—in alphabetical order—Hasan Amjad, Richard Boulton, Anthony Fox, Mike Gordon, Elsa Gunter, John Harrison, Peter Homeier, Gérard Huet (and others at INRIA), Joe Hurd, Ramana Kumar, Ken Friis Larsen, Tom Melham, Robin Milner, Lockwood Morris, Magnus Myreen, Malcolm Newey, Michael Norrish, Larry Paulson, Konrad Slind, Don Syme, Chun Tian, Thomas Türk, Chris Wadsworth, and Tjark Weber. Many others have supplied parts of the system, bug fixes, etc.

Current edition

The current edition of all four volumes (*LOGIC*, *TUTORIAL*, *DESCRIPTION* and *REFERENCE*) has been prepared by Michael Norrish and Konrad Slind. Further contributions to these volumes came from: Hasan Amjad, who developed a model checking library and wrote sections describing its use; Jens Brandt, who developed and documented a library for the rational numbers; Anthony Fox, who formalized and documented new word theories and the associated libraries; Mike Gordon, who documented the libraries for BDDs and SAT; Peter Homeier, who implemented and documented the quotient library; Joe Hurd, who added material on first order proof search; Chun Tian, who documented the HOL theories for probability and measure theory; and Tjark Weber, who wrote libraries for Satisfiability Modulo Theories (SMT) and Quantified Boolean Formulae (QBF).

The material in the third edition constitutes a thorough re-working and extension of previous editions. The only essentially unaltered piece is the semantics by Andy Pitts (in *LOGIC*), reflecting the fact that, although the HOL system has undergone continual development and improvement, the HOL logic is unchanged since the first edition (1988).

Second edition

The second edition of *REFERENCE* was a joint effort by the Cambridge HOL group.

First edition

The three volumes *TUTORIAL*, *DESCRIPTION* and *REFERENCE* were produced at the Cambridge Research Center of SRI International with the support of DSTO Australia.

The HOL documentation project was managed by Mike Gordon, who also wrote parts of *DESCRIPTION* and *TUTORIAL* using material based on an early paper describing the HOL system[1] and *The ML Handbook*[2]. Other contributers to *DESCRIPTION* incude Avra Cohn, who contributed material on theorems, rules, conversions and tactics, and also composed the index (which was typeset by Juanito Camilleri); Tom Melham, who wrote the sections describing type definitions, the concrete type package and the 'resolution' tactics; and Andy Pitts, who devised the set-theoretic semantics of the HOL logic and wrote the material describing it.

The original document design used LaTeX macros supplied by Elsa Gunter, Tom Melham and Larry Paulson. The typesetting of all three volumes was managed by Tom Melham. The cover design is by Arnold Smith, who used a photograph of a 'snow watching lantern' taken by Avra Cohn (in whose garden the original object resides). John Van Tassel composed the LaTeX picture of the lantern.

Many people other than those listed above have contributed to the HOL documentation effort, either by providing material, or by sending lists of errors in the first edition. Thanks to everyone who helped, and thanks to DSTO and SRI for their generous support.

[1] M.J.C. Gordon, 'HOL: a Proof Generating System for Higher Order Logic', in: *VLSI Specification, Verification and Synthesis*, edited by G. Birtwistle and P.A. Subrahmanyam, (Kluwer Academic Publishers, 1988), pp. 73–128.

[2] *The ML Handbook*, unpublished report from Inria by Guy Cousineau, Mike Gordon, Gérard Huet, Robin Milner, Larry Paulson and Chris Wadsworth.

In Memory of Mike Gordon

As documented in the academic literature, in material available from his archived webpages at the University of Cambridge Computer Laboratory, and in these manuals, Mike Gordon was HOL's primary creator and developer. Mike not only created a significant piece of software, inspiring this and many other projects since, but also built a world-leading research group in the Computer Laboratory. This research environment was wonderfully productive for many of the system's authors, and we all owe Mike an enormous debt for both the original work on HOL, and for the way he and his group supported our own development as researchers and HOL hackers.

Mike Gordon, 1948–2017

Contents

Chapter 1

Syntax and Semantics

1.1 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 [2] 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. It is hoped that because of context, no confusion results from these two uses of the word 'meta-language'. When ML is the object of study (as in [5]), 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 alongside 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 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.

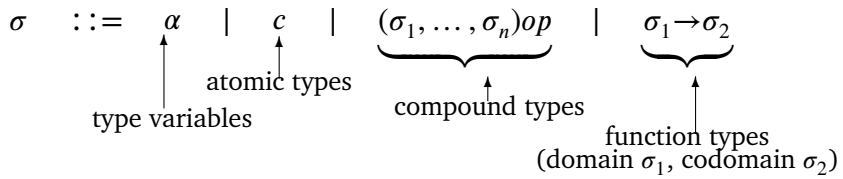
Bool $\mathcal{U}$ contains a distinguished two-element set $2 = \{0, 1\}$.

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.

1.2 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.



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 $\sigma_1 \to \sigma_2$ is the function type with *domain* σ_1 and *codomain* σ_2. It denotes the set of all (total) functions from the set denoted by its domain to the set denoted by its codomain. (In the literature $\sigma_1 \to \sigma_2$ is written without the arrow and backwards—*i.e.* as $\sigma_2\sigma_1$.) Note that syntactically $\to$ 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 1.2.2.)

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.

1.2.1 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 $\nu \in \mathsf{TyNames}$ is the name of the constant and n is its arity. Thus $\Omega \subseteq \mathsf{TyNames} \times \mathbf{N}$ (where $\mathbf{N}$ 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:

- $\mathsf{TyVars} \subseteq \mathsf{Types}_\Omega$.
- 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.

1.2.2 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.

1.2.3 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.

1.3 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.

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

(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 $\mathtt{var}\ x_\sigma$ 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 $\mathtt{var}\ x_{\sigma_1} \in \mathsf{Terms}_{\Sigma_\Omega}$ 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*.

1.3.1 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 1.3.3.

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.

1.3.2 Semantics of terms

Let Σ_Ω be a signature over a type structure Ω (see Section 1.3). 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 1.2.2). 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,

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

when applied to any meanings chosen for the free variables in the term, yields a meaning for the term. Finally, the most 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 1.2.2. 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 1.2.3, $\sigma' = \sigma[\tau_1, \ldots, \tau_p/\beta_1, \ldots, \beta_p]$ 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 1.2.3 (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 2.4.2, where the meaning of several terms denoting logical constants is given.

1.3.3 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 1.2.3 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 $\alpha s, xs$ 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 one has terms-in-context $\alpha s, xs'.t_j$ 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 $\alpha s, xs'.t''$ 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 $\alpha s, xs'.t''$ 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.

1.4 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.

1.4.1 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.

1.4.2 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 1.1. 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 2.5.1) 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 *DESCRIPTION*'s sections on parsing and pretty-printing).

Chapter 2

Theories

2.1 Introduction

The result, if any, of a session with the HOL system is an object called a *theory*. This object is closely related to what a logician would call a theory, but there are some differences arising from the needs of mechanical proof. A HOL theory, like a logician's theory, contains sets of types, constants, definitions and axioms. In addition, however, a HOL theory, at any point in time, contains an explicit list of theorems that have already been proved from the axioms and definitions. Logicians have no need to distinguish theorems actually proved from those merely provable; hence they do not normally consider sets of proven theorems as part of a theory; rather, they take the theorems of a theory to be the (often infinite) set of all consequences of the axioms and definitions. A related difference between logicians' theories and HOL theories is that for logicians, theories are static objects, but in HOL they can be thought of as potentially extendable. For example, the HOL system provides tools for adding to theories and combining theories. A typical interaction with HOL consists in combining some existing theories, making some definitions, proving some theorems and then saving the new results.

The purpose of the HOL system is to provide tools to enable well-formed theories to be constructed. The HOL logic is typed: each theory specifies a signature of type and individual constants; these then determine the sets of types and terms as in the previous chapter. All the theorems of such theories are logical consequences of the definitions and axioms of the theory. The HOL system ensures that only well-formed theories can be constructed by allowing theorems to be created only by *formal proof*. Explicating this involves defining what it means to be a theorem, which leads to the description of the proof system of HOL, to be given below. It is shown to be *sound* for the set theoretic semantics of HOL described in the previous chapter. This means that a theorem is satisfied by a model if it has a formal proof from axioms which are themselves satisfied by the model. Since a logical contradiction is not satisfied by any model, this guarantees in particular that a theory possessing a model is necessarily consistent, *i.e.* a logical contradiction cannot be formally proved from its axioms.

This chapter also describes the various mechanisms by which HOL theories can be extended to new theories. Each mechanism is shown to preserve the property of possessing a model. Thus theories built up from the initial HOL theory (which does possess a

model) using these mechanisms are guaranteed to be consistent.

2.2 Sequents

The HOL logic is phrased in terms of hypothetical assertions called *sequents*. Fixing a (standard) signature Σ_Ω, a sequent is a pair (Γ, t) where Γ is a finite set of formulas over Σ_Ω and t is a single formula over Σ_Ω.[1] The set of formulas Γ forming the first component of a sequent is called its set of *assumptions* and the term t forming the second component is called its *conclusion*. When it is not ambiguous to do so, a sequent $(\{\}, t)$ is written as just t.

Intuitively, a model M of Σ_Ω *satisfies* a sequent (Γ, t) if any interpretation of relevant free variables as elements of M making the formulas in Γ true, also makes the formula t true. To make this more precise, suppose $\Gamma = \{t_1, \ldots, t_p\}$ and let αs,xs be a context containing all the type variables and all the free variables occurring in the formulas $t, t_1, \ldots, t_p$. Suppose that αs has length n, that $xs = x_1, \ldots, x_m$ and that the type of x_j is σ_j. Since formulas are terms of type *bool*, the semantics of terms defined in the previous chapter gives rise to elements $[\![\alpha s,xs.t]\!]_M$ and $[\![\alpha s,xs.t_k]\!]_M$ $(k = 1, \ldots, p)$ in

$$\prod_{Xs \in \mathcal{U}^n} \left(\prod_{j=1}^{m} [\![\alpha s.\sigma_j]\!]_M(Xs) \right) \to 2$$

Say that the model M *satisfies* the sequent (Γ, t) and write

$$\Gamma \models_M t$$

if for all $Xs \in \mathcal{U}^n$ and all $ys \in [\![\alpha s.\sigma_1]\!]_M(Xs) \times \cdots \times [\![\alpha s.\sigma_m]\!]_M(Xs)$ with

$$[\![\alpha s,xs.t_k]\!]_M(Xs)(ys) = 1$$

for all $k = 1, \ldots, p$, it is also the case that

$$[\![\alpha s,xs.t]\!]_M(Xs)(ys) = 1.$$

(Recall that 2 is the set $\{0, 1\}$.)

In the case $p = 0$, the satisfaction of $(\{\}, t)$ by M will be written $\models_M t$. Thus $\models_M t$ means that the dependently typed function

$$[\![t]\!]_M \in \prod_{Xs \in \mathcal{U}^n} \left(\prod_{j=1}^{m} [\![\alpha s.\sigma_j]\!]_M(Xs) \right) \to 2$$

is constant with value $1 \in 2$.

[1] Note that the type subscript is omitted from terms when it is clear from the context that they are formulas, *i.e.* have type *bool*.

2.3 Logic

A deductive system $\mathcal{D}$ is a set of pairs $(L, (\Gamma, t))$ where L is a (possibly empty) list of sequents and (Γ, t) is a sequent.

A sequent (Γ, t) follows from a set of sequents Δ by a deductive system $\mathcal{D}$ if and only if there exist sequents (Γ_1, t_1), … , (Γ_n, t_n) such that:

1. $(\Gamma, t) = (\Gamma_n, t_n)$, and
2. for all i such that $1 \leq i \leq n$
   (a) either $(\Gamma_i, t_i) \in \Delta$ or
   (b) $(L_i, (\Gamma_i, t_i)) \in \mathcal{D}$ for some list L_i of members of $\Delta \cup \{(\Gamma_1, t_1), \ldots, (\Gamma_{i-1}, t_{i-1})\}$.

The sequence $(\Gamma_1, t_1), \cdots, (\Gamma_n, t_n)$ is called a *proof* of (Γ, t) from Δ with respect to $\mathcal{D}$.

Note that if (Γ, t) follows from Δ, then (Γ, t) also follows from any Δ' such that $\Delta \subseteq \Delta'$. This property is called *monotonicity*.

The notation $t_1, \ldots, t_n \vdash_{\mathcal{D},\Delta} t$ means that the sequent $(\{t_1, \ldots, t_n\},\ t)$ follows from Δ by $\mathcal{D}$. If either $\mathcal{D}$ or Δ is clear from the context then it may be omitted. In the case that there are no hypotheses (*i.e.* $n = 0$), just $\vdash t$ is written.

In practice, a particular deductive system is usually specified by a number of (schematic) *rules of inference*, which take the form

$$\frac{\Gamma_1 \vdash t_1 \qquad \cdots \qquad \Gamma_n \vdash t_n}{\Gamma \vdash t}$$

The sequents above the line are called the *hypotheses* of the rule and the sequent below the line is called its *conclusion*. Such a rule is schematic because it may contain metavariables standing for arbitrary terms of the appropriate types. Instantiating these metavariables with actual terms, one gets a list of sequents above the line and a single sequent below the line which together constitute a particular element of the deductive system. The instantiations allowed for a particular rule may be restricted by imposing a *side condition* on the rule.

2.3.1 The HOL deductive system

The deductive system of the HOL logic is specified by eight rules of inference, given below. The first three rules have no hypotheses; their conclusions can always be deduced. The identifiers in square brackets are the names of the ML functions in the HOL system that implement the corresponding inference rules (see *DESCRIPTION*). Any side conditions restricting the scope of a rule are given immediately below it.

Assumption introduction [ASSUME]

$$\frac{}{t \vdash t}$$

Reflexivity [REFL]

$$\frac{}{\vdash t = t}$$

Beta-conversion [BETA_CONV]

$$\frac{}{\vdash (\lambda x.\, t_1) t_2 = t_1[t_2/x]}$$

- Where $t_1[t_2/x]$ is the result of substituting t_2 for x in t_1, with suitable renaming of variables to prevent free variables in t_2 becoming bound after substitution.

Substitution [SUBST]

$$\frac{\Gamma_1 \vdash t_1 = t_1' \qquad \cdots \qquad \Gamma_n \vdash t_n = t_n' \qquad\qquad \Gamma \vdash t[t_1, \ldots, t_n]}{\Gamma_1 \cup \cdots \cup \Gamma_n \cup \Gamma \vdash t[t_1', \ldots, t_n']}$$

- Where $t[t_1, \ldots, t_n]$ denotes a term t with some free occurrences of subterms t_1, … , t_n singled out and $t[t_1', \ldots, t_n']$ denotes the result of replacing each selected occurrence of t_i by t_i' (for $1 \leq i \leq n$), with suitable renaming of variables to prevent free variables in t_i' becoming bound after substitution.

Abstraction [ABS]

$$\frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash (\lambda x.\, t_1) = (\lambda x.\, t_2)}$$

- Provided x is not free in Γ.

Type instantiation [INST_TYPE]

$$\frac{\Gamma \vdash t}{\Gamma[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n] \vdash t[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n]}$$

- Where $t[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n]$ is the result of substituting, in parallel, the types σ_1, … , σ_n for type variables α_1, … , α_n in t, and where $\Gamma[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n]$ is the result of performing the same substitution across all of the theorem's hypotheses.

- After the instantiation, variables free in the input can not become bound, but distinct free variables in the input may become identified.

Discharging an assumption [DISCH]

$$\frac{\Gamma \vdash t_2}{\Gamma - \{t_1\} \vdash t_1 \Rightarrow t_2}$$

- Where $\Gamma - \{t_1\}$ is the set subtraction of $\{t_1\}$ from Γ.

Modus Ponens [MP]

$$\frac{\Gamma_1 \vdash t_1 \Rightarrow t_2 \qquad \Gamma_2 \vdash t_1}{\Gamma_1 \cup \Gamma_2 \vdash t_2}$$

In addition to these eight rules, there are also four *axioms* which could have been regarded as rules of inference without hypotheses. This is not done, however, since it is most natural to state the axioms using some defined logical constants and the principle of constant definition has not yet been described. The axioms are given in Section 2.4.3 and the definitions of the extra logical constants they involve are given in Section 2.4.2.

The particular set of rules and axioms chosen to axiomatize the HOL logic is rather arbitrary. It is partly based on the rules that were used in the LCF logic PPλ, since HOL was implemented by modifying the LCF system. In particular, the substitution rule SUBST is exactly the same as the corresponding rule in LCF; the code implementing this was written by Robin Milner and is highly optimized. Because substitution is such a pervasive activity in proof, it was felt to be important that the system primitive be as fast as possible. From a logical point of view it would be better to have a simpler substitution primitive, such as 'Rule R' of Andrews' logic Q_0, and then to derive more complex rules from it.

2.3.2 Soundness theorem

The rules of the HOL *deductive system are* sound *for the notion of satisfaction defined in Section 2.2: for any instance of the rules of inference, if a (standard) model satisfies the hypotheses of the rule it also satisfies the conclusion.*

Proof The verification of the soundness of the rules is straightforward. The properties of the semantics with respect to substitution given by Lemmas 3 and 4 in Section 1.3.3 are needed for rules BETA_CONV, SUBST and INST_TYPE.[2] The fact that = and $\Rightarrow$ are interpreted standardly (as in Section 1.4.2) is needed for rules REFL, BETA_CONV, SUBST, ABS, DISCH and MP.

[2]Note in particular that the second restriction on INST_TYPE enables the result on the semantics of substituting types for type variables in terms to be applied.

2.4 HOL Theories

A HOL *theory* $\mathcal{T}$ is a 4-tuple:

$$\mathcal{T} \;=\; \langle \mathsf{Struc}_{\mathcal{T}}, \mathsf{Sig}_{\mathcal{T}}, \mathsf{Axioms}_{\mathcal{T}}, \mathsf{Theorems}_{\mathcal{T}} \rangle$$

where

(i) $\mathsf{Struc}_{\mathcal{T}}$ is a type structure called the type structure of $\mathcal{T}$;

(ii) $\mathsf{Sig}_{\mathcal{T}}$ is a signature over $\mathsf{Struc}_{\mathcal{T}}$ called the signature of $\mathcal{T}$;

(iii) $\mathsf{Axioms}_{\mathcal{T}}$ is a set of sequents over $\mathsf{Sig}_{\mathcal{T}}$ called the axioms of $\mathcal{T}$;

(iv) $\mathsf{Theorems}_{\mathcal{T}}$ is a set of sequents over $\mathsf{Sig}_{\mathcal{T}}$ called the theorems of $\mathcal{T}$, with the property that every member follows from $\mathsf{Axioms}_{\mathcal{T}}$ by the HOL deductive system.

The sets $\mathsf{Types}_{\mathcal{T}}$ and $\mathsf{Terms}_{\mathcal{T}}$ of types and terms of a theory $\mathcal{T}$ are, respectively, the sets of types and terms constructable from the type structure and signature of $\mathcal{T}$, *i.e.*:

$$\begin{aligned} \mathsf{Types}_{\mathcal{T}} &= \mathsf{Types}_{\mathsf{Struc}_{\mathcal{T}}} \\ \mathsf{Terms}_{\mathcal{T}} &= \mathsf{Terms}_{\mathsf{Sig}_{\mathcal{T}}} \end{aligned}$$

A model of a theory $\mathcal{T}$ is specified by giving a (standard) model M of the underlying signature of the theory with the property that M satisfies all the sequents which are axioms of $\mathcal{T}$. Because of the Soundness Theorem 2.3.2, it follows that M also satisfies any sequents in the set of given theorems, $\mathsf{Theorems}_{\mathcal{T}}$.

2.4.1 The theory `MIN`

The *minimal theory* `MIN` is defined by:

$$\texttt{MIN} = \langle \{(bool, 0),\ (ind, 0)\},\ \{\Rightarrow_{bool\to bool\to bool}, =_{\alpha\to\alpha\to bool}, \varepsilon_{(\alpha\to bool)\to\alpha}\},\ \{\},\ \{\} \rangle$$

Since the theory `MIN` has a signature consisting only of standard items and has no axioms, it possesses a unique standard model, which will be denoted *Min*.

Although the theory `MIN` contains only the minimal standard syntax, by exploiting the higher order constructs of HOL one can construct a rather rich collection of terms over it. The following theory introduces names for some of these terms that denote useful logical operations in the model *Min*.

In the implementation, the theory `MIN` is given the name `min`, and also contains the distinguished binary type operator $\to$, for constructing function spaces.

2.4.2 The theory LOG

The theory LOG has the same type structure as MIN. Its signature contains the constants in MIN and the following constants:

T_{bool}

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

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

F_{bool}

$\neg_{bool\to bool}$

$\wedge_{bool\to bool\to bool}$

$\vee_{bool\to bool\to bool}$

$\mathsf{One\_One}_{(\alpha\to\beta)\to bool}$

$\mathsf{Onto}_{(\alpha\to\beta)\to bool}$

$\mathsf{Type\_Definition}_{(\alpha\to bool)\to(\beta\to\alpha)\to bool}$

The following special notation is used in connection with these constants:

Notation	Meaning
$\forall x_\sigma.\ t$	$\forall(\lambda x_\sigma.\ t)$
$\forall x_1\ x_2\ \cdots\ x_n.\ t$	$\forall x_1.\ (\forall x_2.\ \cdots\ (\forall x_n.\ t)\ \cdots\ )$
$\exists x_\sigma.\ t$	$\exists(\lambda x_\sigma.\ t)$
$\exists x_1\ x_2\ \cdots\ x_n.\ t$	$\exists x_1.\ (\exists x_2.\ \cdots\ (\exists x_n.\ t)\ \cdots\ )$
$t_1\ \wedge\ t_2$	$\wedge\ t_1\ t_2$
$t_1\ \vee\ t_2$	$\vee\ t_1\ t_2$

The axioms of the theory LOG consist of the following sequents:

$\vdash\ \mathsf{T} = ((\lambda x_{bool}.\ x) = (\lambda x_{bool}.\ x))$
$\vdash\ \forall = \lambda P_{\alpha\to bool}.\ \ P = (\lambda x.\ \mathsf{T})$
$\vdash\ \exists = \lambda P_{\alpha\to bool}.\ \ P(\varepsilon\ P)$
$\vdash\ \mathsf{F} = \forall b_{bool}.\ \ b$
$\vdash\ \neg = \lambda b.\ \ b \Rightarrow \mathsf{F}$
$\vdash\ \wedge = \lambda b_1\ b_2.\ \forall b.\ (b_1 \Rightarrow (b_2 \Rightarrow b)) \Rightarrow b$
$\vdash\ \vee = \lambda b_1\ b_2.\ \forall b.\ (b_1 \Rightarrow b) \Rightarrow ((b_2 \Rightarrow b) \Rightarrow b)$
$\vdash\ \mathsf{One\_One} = \lambda f_{\alpha\to\beta}.\ \forall x_1\ x_2.\ (f\ x_1 = f\ x_2) \Rightarrow (x_1 = x_2)$
$\vdash\ \mathsf{Onto} = \lambda f_{\alpha\to\beta}.\ \forall y.\ \exists x.\ y = f\ x$
$\vdash\ \mathsf{Type\_Definition} = \ \lambda P_{\alpha\to bool}\ rep_{\beta\to\alpha}.\mathsf{One\_One}\ rep\ \ \wedge$
$(\forall x.\ P\ x\ =\ (\exists y.\ x = rep\ y))$

Finally, as for the theory MIN, the set $\mathsf{Theorems}_{\mathtt{LOG}}$ is taken to be empty.

Note that the axioms of the theory LOG are essentially *definitions* of the new constants of LOG as terms in the original theory MIN. (The mechanism for making such extensions of theories by definitions of new constants will be set out in general in Section 2.5.1.) The first seven axioms define the logical constants for truth, universal quantification, existential quantification, falsity, negation, conjunction and disjunction. Although these definitions may be obscure to some readers, they are in fact standard definitions of these logical constants in terms of implication, equality and choice within higher order logic. The next two axioms define the properties of a function being one-one and onto; they will be used to express the axiom of infinity (see Section 2.4.3), amongst other things. The last axiom defines a constant used for type definitions (see Section 2.5.4).

The unique standard model *Min* of MIN gives rise to a unique standard model of LOG. This is because, given the semantics of terms set out in Section 1.3.2, to satisfy the above equations one is forced to interpret the new constants in the following way:

- $[\![\mathsf{T}_{bool}]\!] = 1 \in 2$
- $[\![\forall_{(\alpha\to bool)\to bool}]\!] \in \prod_{X\in\mathcal{U}}(X\to 2)\to 2$ sends $X \in \mathcal{U}$ and $f \in X\to 2$ to
$$[\![\forall]\!](X)(f) = \begin{cases} 1 & \text{if } f^{-1}\{1\} = X \\ 0 & \text{otherwise} \end{cases}$$
- $[\![\exists_{(\alpha\to bool)\to bool}]\!] \in \prod_{X\in\mathcal{U}}(X\to 2)\to 2$ sends $X \in \mathcal{U}$ and $f \in X\to 2$ to
$$[\![\exists]\!](X)(f) = \begin{cases} 1 & \text{if } f^{-1}\{1\} \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$
- $[\![\mathsf{F}_{bool}]\!] = 0 \in 2$
- $[\![\neg_{bool\to bool}]\!] \in 2\to 2$ sends $b \in 2$ to
$$[\![\neg]\!](b) = \begin{cases} 1 & \text{if } b = 0 \\ 0 & \text{otherwise} \end{cases}$$
- $[\![\wedge_{bool\to bool\to bool}]\!] \in 2\to 2\to 2$ sends $b, b' \in 2$ to
$$[\![\wedge]\!](b)(b') = \begin{cases} 1 & \text{if } b = 1 = b' \\ 0 & \text{otherwise} \end{cases}$$
- $[\![\vee_{bool\to bool\to bool}]\!] \in 2\to 2\to 2$ sends $b, b' \in 2$ to
$$[\![\vee]\!](b)(b') = \begin{cases} 0 & \text{if } b = 0 = b' \\ 1 & \text{otherwise} \end{cases}$$

- $[\![\mathsf{One\_One}_{(\alpha\to\beta)\to bool}]\!] \in \prod_{(X,Y)\in\mathcal{U}^2}(X\to Y)\to 2$ sends $(X,Y)\in\mathcal{U}^2$ and $f\in(X\to Y)$ to

$$[\![\mathsf{One\_One}]\!](X,Y)(f) = \begin{cases} 0 & \text{if } f(x)=f(x') \text{ for some } x\neq x' \text{ in } X \\ 1 & \text{otherwise} \end{cases}$$

- $[\![\mathsf{Onto}_{(\alpha\to\beta)\to bool}]\!] \in \prod_{(X,Y)\in\mathcal{U}^2}(X\to Y)\to 2$ sends $(X,Y)\in\mathcal{U}^2$ and $f\in(X\to Y)$ to

$$[\![\mathsf{Onto}]\!](X,Y)(f) = \begin{cases} 1 & \text{if } \{f(x) : x\in X\} = Y \\ 0 & \text{otherwise} \end{cases}$$

- $[\![\mathsf{Type\_Definition}_{(\alpha\to bool)\to(\beta\to\alpha)\to bool}]\!] \in \prod_{(X,Y)\in\mathcal{U}^2}(X\to 2)\to(Y\to X)\to 2$ sends $(X,Y)\in\mathcal{U}^2$, $f\in(X\to 2)$ and $g\in(Y\to X)$ to

$$[\![\mathsf{Type\_Definition}]\!](X,Y)(f)(g) = \begin{cases} 1 & \text{if } [\![\mathsf{One\_One}]\!](Y,X)(g)=1 \\ & \text{and } f^{-1}\{1\} = \{g(y) : y\in Y\} \\ 0 & \text{otherwise.} \end{cases}$$

Since these definitions were obtained by applying the semantics of terms to the left hand sides of the equations which form the axioms of LOG, these axioms are satisfied and one obtains a model of the theory LOG.

2.4.3 The theory INIT

The theory INIT is obtained by adding the following four axioms to the theory LOG.

BOOL_CASES_AX	$\vdash \forall b.\ (b=\mathsf{T})\vee(b=\mathsf{F})$
ETA_AX	$\vdash \forall f_{\alpha\to\beta}.\ (\lambda x.\ f\ x) = f$
SELECT_AX	$\vdash \forall P_{\alpha\to bool}\ x.\ P\ x \Rightarrow P(\varepsilon\ P)$
INFINITY_AX	$\vdash \exists f_{ind\to ind}.\ \mathsf{One\_One}\ f\ \wedge\ \neg(\mathsf{Onto}\ f)$

The unique standard model of LOG satisfies these four axioms and hence is the unique standard model of the theory INIT. (For axiom SELECT_AX one needs to use the definition of $[\![\varepsilon]\!]$ given in Section 1.4.2; for axiom INFINITY_AX one needs the fact that $[\![ind]\!] = \mathrm{I}$ is an infinite set.)

The theory INIT is the initial theory of the HOL logic. A theory which extends INIT will be called a *standard theory*.

2.4.4 Implementing theories LOG and INIT

The implementation combines the theories LOG and INIT into a theory bool. It includes all of the constants and axioms from those theories, and includes a number of derived results about those constants. For more on the implementation's bool theory, see *DESCRIPTION*.

2.4.5 Consistency

A (standard) theory is *consistent* if it is not the case that every sequent over its signature can be derived from the theory's axioms using the HOL logic, or equivalently, if the particular sequent $\vdash \mathsf{F}$ cannot be so derived.

The existence of a (standard) model of a theory is sufficient to establish its consistency. For by the Soundness Theorem 2.3.2, any sequent that can be derived from the theory's axioms will be satisfied by the model, whereas the sequent $\vdash \mathsf{F}$ is never satisfied in any standard model. So in particular, the initial theory INIT is consistent.

However, it is possible for a theory to be consistent but not to possess a standard model. This is because the notion of a *standard* model is quite restrictive—in particular there is no choice how to interpret the integers and their arithmetic in such a model. The famous incompleteness theorem of Gödel ensures that there are sequents which are satisfied in all standard models (*i.e.* which are 'true'), but which are not provable in the HOL logic.

2.5 Extensions of theories

A theory $\mathcal{T}'$ is said to be an *extension* of a theory $\mathcal{T}$ if:

(i) $\mathsf{Struc}_{\mathcal{T}} \subseteq \mathsf{Struc}_{\mathcal{T}'}$.

(ii) $\mathsf{Sig}_{\mathcal{T}} \subseteq \mathsf{Sig}_{\mathcal{T}'}$.

(iii) $\mathsf{Axioms}_{\mathcal{T}} \subseteq \mathsf{Axioms}_{\mathcal{T}'}$.

(iv) $\mathsf{Theorems}_{\mathcal{T}} \subseteq \mathsf{Theorems}_{\mathcal{T}'}$.

In this case, any model M' of the larger theory $\mathcal{T}'$ can be restricted to a model of the smaller theory $\mathcal{T}$ in the following way. First, M' gives rise to a model of the structure and signature of $\mathcal{T}$ simply by forgetting the values of M' at constants not in $\mathsf{Struc}_{\mathcal{T}}$ or $\mathsf{Sig}_{\mathcal{T}}$. Denoting this model by M, one has for all $\sigma \in \mathsf{Types}_{\mathcal{T}}$, $t \in \mathsf{Terms}_{\mathcal{T}}$ and for all suitable contexts that

$$\begin{aligned} [\![\alpha s.\sigma]\!]_M &= [\![\alpha s.\sigma]\!]_{M'} \\ [\![\alpha s, xs.t]\!]_M &= [\![\alpha s, xs.t]\!]_{M'}. \end{aligned}$$

Consequently if (Γ, t) is a sequent over $\mathsf{Sig}_{\mathcal{T}}$ (and hence also over $\mathsf{Sig}_{\mathcal{T}'}$), then $\Gamma \models_M t$ if and only if $\Gamma \models_{M'} t$. Since $\mathsf{Axioms}_{\mathcal{T}} \subseteq \mathsf{Axioms}_{\mathcal{T}'}$ and M' is a model of $\mathcal{T}'$, it follows that M is a model of $\mathcal{T}$. M will be called the *restriction* of the model M' of the theory $\mathcal{T}'$ to the subtheory $\mathcal{T}$.

There are two main mechanisms for making extensions of theories in HOL:

- Extension by a constant specification (see Section 2.5.2).
- Extension by a type specification (see Section 2.5.5).[3]

The first mechanism allows 'loose specification' of constants (as in the **Z** notation [6], for example); the latter allows new types and type-operators to be introduced. As special cases (when the thing being specified is uniquely determined) one also has:

- Extension by a constant definition (see Section 2.5.1).
- Extension by a type definition (see Section 2.5.4).

These mechanisms are described in the following sections. They all produce *definitional extensions* in the sense that they extend a theory by adding new constants and types which are defined in terms of properties of existing ones. Their key property is that the extended theory possesses a (standard) model if the original theory does. So a series of these extensions starting from the theory `INIT` is guaranteed to result in a theory with a standard model, and hence in a consistent theory. It is also possible to extend theories simply by adding new uninterpreted constants and types. This preserves consistency, but is unlikely to be useful without additional axioms. However, when adding arbitrary new axioms, there is no guarantee that consistency is preserved. The advantages of postulation over definition have been likened by Bertrand Russell to the advantages of theft over honest toil.[4] As it is all too easy to introduce inconsistent axiomatizations, users of the HOL system are strongly advised to resist the temptation to add axioms, but to toil through definitional theories honestly.

2.5.1 Extension by constant definition

A *constant definition* over a signature Σ_Ω is a formula of the form $\mathsf{c}_\sigma = t_\sigma$, such that:

(i) c is not the name of any constant in Σ_Ω;

(ii) t_σ a closed term in $\mathsf{Terms}_{\Sigma_\Omega}$;

(iii) all the type variables occurring in t_σ also occur in σ.

[3] This theory extension mechanism is not implemented in the HOL4 system.

[4] See page 71 of Russell's book *Introduction to Mathematical Philosophy*.

Given a theory $\mathcal{T}$ and such a constant definition over $\mathsf{Sig}_{\mathcal{T}}$, then the *definitional extension* of $\mathcal{T}$ by $\mathsf{c}_\sigma = t_\sigma$ is the theory $\mathcal{T} +_{def} \langle \mathsf{c}_\sigma = t_\sigma \rangle$ defined by:

$$\begin{aligned}\mathcal{T} +_{def} \langle \mathsf{c}_\sigma = t_\sigma \rangle \;=\; \langle\; & \mathsf{Struc}_{\mathcal{T}},\ \mathsf{Sig}_{\mathcal{T}} \cup \{(\mathsf{c}, \sigma)\},\\ & \mathsf{Axioms}_{\mathcal{T}} \cup \{\mathsf{c}_\sigma = t_\sigma\},\ \mathsf{Theorems}_{\mathcal{T}} \rangle\end{aligned}$$

Note that the mechanism of extension by constant definition has already been used implicitly in forming the theory LOG from the theory MIN in Section 2.4.2. Thus with the notation of this section one has

$$\begin{aligned}\mathtt{LOG} \;=\; \mathtt{MIN} & +_{def} \langle \mathsf{T} \;=\; ((\lambda x_{bool}.\ x) = (\lambda x_{bool}.\ x)) \rangle\\ & +_{def} \langle \forall \;=\; \lambda P_{\alpha \to bool}.\ \ P = (\lambda x.\ \mathsf{T}) \rangle\\ & +_{def} \langle \exists \;=\; \lambda P_{\alpha \to bool}.\ \ P(\varepsilon\ P) \rangle\\ & +_{def} \langle \mathsf{F} \;=\; \forall b_{bool}.\ \ b \rangle\\ & +_{def} \langle \neg \;=\; \lambda b.\ \ b \Rightarrow \mathsf{F} \rangle\\ & +_{def} \langle \wedge \;=\; \lambda b_1\ b_2.\ \forall b.\ (b_1 \Rightarrow (b_2 \Rightarrow b)) \Rightarrow b \rangle\\ & +_{def} \langle \vee \;=\; \lambda b_1\ b_2.\ \forall b.\ (b_1 \Rightarrow b) \Rightarrow ((b_2 \Rightarrow b) \Rightarrow b) \rangle\\ & +_{def} \langle \mathsf{One\_One} \;=\; \lambda f_{\alpha \to \beta}.\ \forall x_1\ x_2.\ (f\ x_1 = f\ x_2) \Rightarrow (x_1 = x_2) \rangle\\ & +_{def} \langle \mathsf{Onto} \;=\; \lambda f_{\alpha \to \beta}.\ \forall y.\ \exists x.\ y = f\ x \rangle\\ & +_{def} \langle \mathsf{Type\_Definition} \;=\; \lambda P_{\alpha \to bool}\ rep_{\beta \to \alpha}.\\ & \qquad\qquad \mathsf{One\_One}\ rep\ \wedge\\ & \qquad\qquad (\forall x.\ P\ x \;=\; (\exists y.\ x = rep\ y)) \rangle\end{aligned}$$

If $\mathcal{T}$ possesses a standard model then so does the extension $\mathcal{T} +_{def} \langle \mathsf{c}_\sigma = t_\sigma \rangle$. This will be proved as a corollary of the corresponding result in Section 2.5.2 by showing that extension by constant definition is in fact a special case of extension by constant specification. (This reduction requires that one is dealing with *standard* theories in the sense of section 2.4.3, since although existential quantification is not needed for constant definitions, it is needed to state the mechanism of constant specification.)

Remark Condition (iii) in the definition of what constitutes a correct constant definition is an important restriction without which consistency could not be guaranteed. To see this, consider the term $\exists f_{\alpha \to \alpha}.\ \mathsf{One\_One}\ f\ \wedge\ \neg(\mathsf{Onto}\ f)$, which expresses the proposition that (the set of elements denoted by the) type α is infinite. The term contains the type variable α, whereas the type of the term, *bool*, does not. Thus by (iii)

$$\mathsf{c}_{bool} = \exists f_{\alpha \to \alpha}.\ \mathsf{One\_One}\ f\ \wedge\ \neg(\mathsf{Onto}\ f)$$

is not allowed as a constant definition. The problem is that the meaning of the right hand side of the definition varies with α, whereas the meaning of the constant on the left hand side is fixed, since it does not contain α. Indeed, if we were allowed to extend the consistent theory INIT by this definition, the result would be an inconsistent theory. For instantiating α to *ind* in the right hand side results in a term that is provable from the axioms of INIT, and hence $\mathsf{c}_{bool} = \mathsf{T}$ is provable in the extended theory. But equally,

instantiating α to *bool* makes the negation of the right hand side provable from the axioms of INIT, and hence $\mathsf{c}_{bool} = \mathsf{F}$ is also provable in the extended theory. Combining these theorems, one has that $\mathsf{T} = \mathsf{F}$, *i.e.* F is provable in the extended theory.

2.5.2 Extension by constant specification

Constant specifications introduce constants (or sets of constants) that satisfy arbitrary given (consistent) properties. For example, a theory could be extended by a constant specification to have two new constants b_1 and b_2 of type *bool* such that $\neg(\mathsf{b}_1 = \mathsf{b}_2)$. This specification does not uniquely define b_1 and b_2, since it is satisfied by either $\mathsf{b}_1 = \mathsf{T}$ and $\mathsf{b}_2 = \mathsf{F}$, or $\mathsf{b}_1 = \mathsf{F}$ and $\mathsf{b}_2 = \mathsf{T}$. To ensure that such specifications are consistent, they can only be made if it has already been proved that the properties which the new constants are to have are consistent. This rules out, for example, introducing three boolean constants b_1, b_2 and b_3 such that $\mathsf{b}_1 \neq \mathsf{b}_2$, $\mathsf{b}_1 \neq \mathsf{b}_3$ and $\mathsf{b}_2 \neq \mathsf{b}_3$.

Suppose $\exists x_1 \cdots x_n.\ t$ is a formula, with $x_1, \ldots, x_n$ distinct variables. If $\vdash\ \exists x_1 \cdots x_n.\ t$, then a constant specification allows new constants $\mathsf{c}_1, \ldots, \mathsf{c}_n$ to be introduced satisfying:

$$\vdash\ t[\mathsf{c}_1, \cdots, \mathsf{c}_n / x_1, \cdots, x_n]$$

where $t[\mathsf{c}_1, \cdots, \mathsf{c}_n / x_1, \cdots, x_n]$ denotes the result of simultaneously substituting $\mathsf{c}_1, \ldots, \mathsf{c}_n$ for $x_1, \ldots, x_n$ respectively. Of course the type of each constant c_i must be the same as the type of the corresponding variable x_i. To ensure that this extension mechanism preserves the property of possessing a model, a further more technical requirement is imposed on these types: they must each contain all the type variables occurring in t. This condition is discussed further in Section 2.5.3 below.

Formally, a *constant specification* for a theory $\mathcal{T}$ is given by

Data

$$\langle(\mathsf{c}_1, \ldots, \mathsf{c}_n), \lambda x_{1\sigma_1}, \ldots, x_{n\sigma_n}.\ t_{bool}\rangle$$

Conditions

(i) $\mathsf{c}_1, \ldots, \mathsf{c}_n$ are distinct names that are not the names of any constants in $\mathsf{Sig}_{\mathcal{T}}$.

(ii) $\lambda x_{1\sigma_1} \cdots x_{n\sigma_n}.\ t_{bool}\ \in\ \mathsf{Terms}_{\mathcal{T}}$.

(iii) $tyvars(t_{bool})\ =\ tyvars(\sigma_i)$ for $1 \leq i \leq n$.

(iv) $\exists x_{1\sigma_1}\ \cdots\ x_{n\sigma_n}.\ t\ \in\ \mathsf{Theorems}_{\mathcal{T}}$.

The extension of a standard theory $\mathcal{T}$ by such a constant specification is denoted by

$$\mathcal{T} +_{spec} \langle(\mathsf{c}_1, \ldots, \mathsf{c}_n), \lambda x_{1\sigma_1}, \ldots, x_{n\sigma_n}.\ t_{bool}\rangle$$

and is defined to be the theory:

$$\begin{array}{l}\langle \mathsf{Struc}_{\mathcal{T}},\\ \ \mathsf{Sig}_{\mathcal{T}} \cup \{\mathsf{c}_{1_{\sigma_1}}, \ldots, \mathsf{c}_{n_{\sigma_n}}\},\\ \ \mathsf{Axioms}_{\mathcal{T}} \cup \{t[\mathsf{c}_1, \ldots, \mathsf{c}_n / x_1, \ldots, x_n]\},\\ \ \mathsf{Theorems}_{\mathcal{T}}\rangle\end{array}$$

Proposition *The theory* $\mathcal{T} +_{spec} \langle (\mathsf{c}_1, \ldots, \mathsf{c}_n), \lambda x_{1_{\sigma_1}}, \ldots, x_{n_{\sigma_n}}.\ t_{bool}\rangle$ *has a standard model if the theory* $\mathcal{T}$ *does.*

Proof Suppose M is a standard model of $\mathcal{T}$. Let $\alpha s = \alpha_1, \ldots, \alpha_m$ be the list of distinct type variables occurring in the formula t. Then $\alpha s, xs.t$ is a term-in-context, where $xs = x_1, \ldots, x_n$. (Change any bound variables in t to make them distinct from xs if necessary.) Interpreting this term-in-context in the model M yields

$$[\![\alpha s, xs.t]\!]_M \in \prod_{Xs \in \mathcal{U}^m} \left(\prod_{i=1}^{n} [\![\alpha s.\sigma_i]\!]_M(Xs) \right) \to 2$$

Now $\exists xs.\ t$ is in $\mathsf{Theorems}_{\mathcal{T}}$ and hence by the Soundness Theorem 2.3.2 this sequent is satisfied by M. Using the semantics of $\exists$ given in Section 2.4.2, this means that for all $Xs \in \mathcal{U}^m$ the set

$$S(Xs) = \{ys \in [\![\alpha s.\sigma_1]\!]_M(Xs) \times \cdots \times [\![\alpha s.\sigma_n]\!]_M(Xs) \ :\ [\![\alpha s, xs.t]\!]_M(Xs)(ys) = 1\}$$

is non-empty. Since it is also a subset of a finite product of sets in $\mathcal{U}$, it follows that it is an element of $\mathcal{U}$ (using properties **Sub** and **Prod** of the universe). So one can apply the global choice function $\mathrm{ch} \in \prod_{X \in \mathcal{U}} X$ to select a specific element

$$(s_1(Xs), \ldots, s_n(Xs)) = \mathrm{ch}(S(Xs)) \in \prod_{i=1}^{n} [\![\alpha s.\sigma_i]\!]_M(Xs)$$

at which $[\![\alpha s, xs.t]\!]_M(Xs)$ takes the value 1. Extend M to a model M' of the signature of $\mathcal{T} +_{spec} \langle (\mathsf{c}_1, \ldots, \mathsf{c}_n), \lambda x_{1_{\sigma_1}}, \ldots, x_{n_{\sigma_n}}.\ t_{bool}\rangle$ by defining its value at each new constant (c_i, σ_i) to be

$$M'(\mathsf{c}_i, \sigma_i) = s_i \in \prod_{Xs \in \mathcal{U}^m} [\![\sigma_i]\!]_M(Xs).$$

Note that the Condition (iii) in the definition of a constant specification ensures that αs is the canonical context of each type σ_i, so that $[\![\sigma_i]\!] = [\![\alpha s.\sigma_i]\!]$ and thus s_i is indeed an element of the above product.

Since t is a term of the subtheory $\mathcal{T}$ of $\mathcal{T} +_{spec} \langle (\mathsf{c}_1, \ldots, \mathsf{c}_n), \lambda x_{1_{\sigma_1}}, \ldots, x_{n_{\sigma_n}}.\ t_{bool}\rangle$, as remarked at the beginning of Section 2.5, one has that $[\![\alpha s, xs.t]\!]_{M'} = [\![\alpha s, xs.t]\!]_M$. Hence by definition of the s_i, for all $Xs \in \mathcal{U}^m$

$$[\![\alpha s, xs.t]\!]_{M'}(Xs)(s_1(Xs), \ldots, s_n(Xs)) = 1$$

Then using Lemma 4 in Section 1.3.3 on the semantics of substitution together with the definition of $[\![\mathsf{c}_i]\!]_{M'}$, one finally obtains that for all $Xs \in \mathcal{U}^m$

$$[\![t[\mathsf{c}_1, \ldots, \mathsf{c}_n/x_1, \ldots, x_n]]\!]_{M'}(Xs) = 1$$

or in other words that M' satisfies $t[\mathsf{c}_1, \ldots, \mathsf{c}_n/x_1, \ldots, x_n]$. Hence M' is a model of $\mathcal{T} +_{spec} \langle (\mathsf{c}_1, \ldots, \mathsf{c}_n), \lambda x_{1_{\sigma_1}}, \ldots, x_{n_{\sigma_n}}.\ t_{bool} \rangle$, as required.

The constants which are asserted to exist in a constant specification are not necessarily uniquely determined. Correspondingly, there may be many different models of $\mathcal{T} +_{spec} \langle (\mathsf{c}_1, \ldots, \mathsf{c}_n), \lambda x_{1_{\sigma_1}}, \ldots, x_{n_{\sigma_n}}.\ t_{bool} \rangle$ whose restriction to $\mathcal{T}$ is M; the above construction produces such a model in a uniform manner by making use of the global choice function on the universe.

Extension by a constant definition, $\mathsf{c}_\sigma = t_\sigma$, is a special case of extension by constant specification. For let t' be the formula $x_\sigma = t_\sigma$, where x_σ is a variable not occurring in t_σ. Then clearly $\vdash \exists x_\sigma.\ t'$ and one can apply the method of constant specification to obtain the theory

$$\mathcal{T} +_{spec} \langle \mathsf{c}, \lambda x_\sigma.\ t' \rangle$$

But since $t'[\mathsf{c}_\sigma/x_\sigma]$ is just $\mathsf{c}_\sigma = t_\sigma$, this extension yields exactly $\mathcal{T} +_{def} \langle \mathsf{c}_\sigma = t_\sigma \rangle$. So as a corollary of the Proposition, one has that for each standard model M of $\mathcal{T}$, there is a standard model M' of $\mathcal{T} +_{def} \langle \mathsf{c}_\sigma = t_\sigma \rangle$ whose restriction to $\mathcal{T}$ is M. In contrast with the case of constant specifications, M' is uniquely determined by M and the constant definition.

2.5.3 Remarks about constants in HOL

Note how Condition (iii) in the definition of a constant specification was needed in the proof that the extension mechanism preserves the property of possessing a standard model. Its role is to ensure that the introduced constants have, via their types, the same dependency on type variables as does the formula loosely specifying them. The situation is the same as that discussed in the Remark in Section 2.5.1. In a sense, what is causing the problem in the example given in that Remark is not so much the method of extension by introducing constants, but rather the syntax of HOL which does not allow constants to depend explicitly on type variables (in the way that type operators can). Thus in the example one would like to introduce a 'polymorphic' constant $\mathsf{c}_{bool}(\alpha)$ explicitly depending upon α, and define it to be $\exists f_{\alpha\to\alpha}.\ \mathsf{One\_One}\ f\ \wedge\ \neg(\mathsf{Onto}\ f)$. Then in the extended theory one could derive $\mathsf{c}_{bool}(ind) = \mathsf{T}$ and $\mathsf{c}_{bool}(bool) = \mathsf{F}$, but now no contradiction results since $\mathsf{c}_{bool}(ind)$ and $\mathsf{c}_{bool}(bool)$ are different.

In the current version of HOL, constants are (name,type)-pairs. One can envision a slight extension of the HOL syntax with 'polymorphic' constants, specified by pairs

$(\mathsf{c}, \alpha s.\sigma)$ where now $\alpha s.\sigma$ is a type-in-context and the list αs may well contain extra type variables not occurring in σ. Such a pair would give rise to the particular constant term $\mathsf{c}_\sigma(\alpha s)$, and more generally to constant terms $\mathsf{c}_{\sigma'}(\tau s)$ obtained from this one by instantiating the type variables α_i with types τ_i (so σ' is the instance of σ obtained by substituting τs for αs). This new syntax of polymorphic constants is comparable to the existing syntax of compound types (see section 1.2): an n-ary type operator *op* gives rise to a compound type $(\alpha_1, \ldots, \alpha_n)op$ depending upon n type variables. Similarly, the above syntax of polymorphic constants records how they depend upon type variables (as well as which generic type the constant has).

However, explicitly recording dependency of constants on type variables makes for a rather cumbersome syntax which in practice one would like to avoid where possible. It is possible to avoid it if the type context αs in $(\mathsf{c}, \alpha s.\sigma)$ is actually the *canonical* context of σ, *i.e.* contains exactly the type variables of σ. For then one can apply Lemma 1 of Section 1.2.3 to deduce that the polymorphic constant $\mathsf{c}_{\sigma'}(\tau s)$ can be abbreviated to the ordinary constant $\mathsf{c}_{\sigma'}$ without ambiguity—the missing information τs can be reconstructed from σ' and the information about the constant c given in the signature. From this perspective, the rather technical side Conditions (iii) in Sections 2.5.1 and 2.5.2 become rather less mysterious: they precisely ensure that in introducing new constants one is always dealing just with canonical contexts, and so can use ordinary constants rather than polymorphic ones without ambiguity. In this way one avoids complicating the existing syntax at the expense of restricting somewhat the applicability of these theory extension mechanisms.

2.5.4 Extension by type definition

Every (monomorphic) type σ in the initial theory `INIT` determines a set $[\![\sigma]\!]$ in the universe $\mathcal{U}$. However, there are many more sets in $\mathcal{U}$ than there are types in `INIT`. In particular, whilst $\mathcal{U}$ is closed under the operation of taking a non-empty subset of $[\![\sigma]\!]$, there is no corresponding mechanism for forming a 'subtype' of σ. Instead, subsets are denoted indirectly via characteristic functions, whereby a closed term p of type $\sigma \to bool$ determines the subset $\{x \in [\![\sigma]\!] : [\![p]\!](x) = 1\}$ (which is a set in the universe provided it is non-empty). However, it is useful to have a mechanism for introducing new types which are subtypes of existing ones. Such types are defined in HOL by introducing a new type constant and asserting an axiom that characterizes it as denoting a set in bijection (*i.e.* one-to-one correspondence) with a non-empty subset of an existing type (called the *representing type*). For example, the type `num` is defined to be equal to a countable subset of the type `ind`, which is guaranteed to exist by the axiom `INFINITY_AX` (see Section 2.4.3).

As well as defining types, it is also convenient to be able to define type operators. An example would be a type operator *inj* which mapped a set to the set of one-to-one (*i.e.*

injective) functions on it. The subset of $\sigma \rightarrow \sigma$ representing $(\sigma)inj$ would be defined by the predicate One_One. Another example would be a binary cartesian product type operator *prod*. This is defined by choosing a representing type containing two type variables, say $\sigma[\alpha_1; \alpha_2]$, such that for any types σ_1 and σ_2, a subset of $\sigma[\sigma_1; \sigma_2]$ represents the cartesian product of σ_1 and σ_2. The details of such a definition are given in *DESCRIPTION*'s section on the theory of cartesian products.

Types in HOL must denote non-empty sets. Thus it is only consistent to define a new type isomorphic to a subset specified by a predicate p, if there is at least one thing for which p holds, *i.e.* $\vdash \exists x.\ p\ x$. For example, it would be inconsistent to define a binary type operator *iso* such that $(\sigma_1, \sigma_2)iso$ denoted the set of one-to-one functions from σ_1 *onto* σ_2 because for some values of σ_1 and σ_2 the set would be empty; for example $(ind, bool)iso$ would denote the empty set. To avoid this, a precondition of defining a new type is that the representing subset is non-empty.

To summarize, a new type is defined by:

1. Specifying an existing type.
2. Specifying a subset of this type.
3. Proving that this subset is non-empty.
4. Specifying that the new type is isomorphic to this subset.

In more detail, defining a new type $(\alpha_1, \ldots, \alpha_n)op$ consists in:

1. Specifying a type-in-context, $\alpha_1, \ldots, \alpha_n.\sigma$ say. The type σ is called the *representing type*, and the type $(\alpha_1, \ldots, \alpha_n)op$ is intended to be isomorphic to a subset of σ.
2. Specifying a closed term-in-context, $\alpha_1, \ldots, \alpha_n, .p$ say, of type $\sigma \rightarrow bool$. The term p is called the *characteristic function*. This defines the subset of σ to which $(\alpha_1, \ldots, \alpha_n)op$ is to be isomorphic.[5]
3. Proving $\vdash \exists x_\sigma.\ p\ x$.
4. Asserting an axiom saying that $(\alpha_1, \ldots, \alpha_n)op$ is isomorphic to the subset of σ selected by p.

To make this formal, the theory LOG provides the polymorphic constant Type_Definition defined in Section 2.4.2. The formula $\exists f_{(\alpha_1,\ldots,\alpha_n)op \rightarrow \sigma}.$ Type_Definition $p\ f$ asserts that there

[5]The reason for restricting p to be closed, *i.e.* to have no free variables, is that otherwise for consistency the defined type operator would have to *depend* upon (*i.e.* be a function of) those variables. Such dependent types are not (yet!) a part of the HOL system.

exists a one-to-one map f from $(\alpha_1, \ldots, \alpha_n)op$ onto the subset of elements of σ for which p is true. Hence, the axiom that characterizes $(\alpha_1, \ldots, \alpha_n)op$ is:

$$\vdash \exists f_{(\alpha_1,\ldots,\alpha_n)op\to\sigma}.\ \mathsf{Type\_Definition}\ p\ f$$

Defining a new type $(\alpha_1, \ldots, \alpha_n)op$ in a theory $\mathcal{T}$ thus consists of introducing op as a new n-ary type operator and the above axiom as a new axiom. Formally, a *type definition* for a theory $\mathcal{T}$ is given by

Data

$$\langle (\alpha_1, \ldots, \alpha_n)op,\ \sigma,\ p_{\sigma\to bool}\rangle$$

Conditions

(i) (op, n) is not the name of a type constant in $\mathsf{Struc}_{\mathcal{T}}$.

(ii) $\alpha_1, \ldots, \alpha_n.\sigma$ is a type-in-context with $\sigma \in \mathsf{Types}_{\mathcal{T}}$.

(iii) $p_{\sigma\to bool}$ is a closed term in $\mathsf{Terms}_{\mathcal{T}}$ whose type variables occur in $\alpha_1, \ldots, \alpha_n$.

(iv) $\exists x_\sigma.\ p\ x\ \in\ \mathsf{Theorems}_{\mathcal{T}}$.

The extension of a standard theory $\mathcal{T}$ by a such a type definition is denoted by

$$\mathcal{T} +_{tydef} \langle (\alpha_1, \ldots, \alpha_n)op, \sigma, p\rangle$$

and defined to be the theory

$$\begin{array}{l}\langle \mathsf{Struc}_{\mathcal{T}} \cup \{(op, n)\},\\ \ \mathsf{Sig}_{\mathcal{T}},\\ \ \mathsf{Axioms}_{\mathcal{T}} \cup \{\exists f_{(\alpha_1,\ldots,\alpha_n)op\to\sigma}.\ \mathsf{Type\_Definition}\ p\ f\},\\ \ \mathsf{Theorems}_{\mathcal{T}}\rangle\end{array}$$

Proposition *The theory $\mathcal{T} +_{tydef} \langle (\alpha_1, \ldots, \alpha_n)op, \sigma, p\rangle$ has a standard model if the theory $\mathcal{T}$ does.*

Instead of giving a direct proof of this result, it will be deduced as a corollary of the corresponding proposition in the next section.

2.5.5 Extension by type specification

(**Note:** This theory extension mechanism is *not* implemented in the HOL4 system. It was proposed by T. Melham and refines a suggestion from R. Jones and R. Arthan.)

The type definition mechanism allows one to introduce new types by giving a concrete representation of the type as a 'subtype' of an existing type. One might instead wish

to introduce a new type satisfying some property without having to give an explicit representation for the type. For example, one might want to extend INIT with an atomic type *one* satisfying $\vdash\ \forall f_{\alpha\to one}\ g_{\alpha\to one}.\ f = g$ without choosing a specific type in INIT and saying that *one* is in bijection with a one-element subset of it. (The idea being that the choice of representing type is irrelevant to the properties of *one* that can be expressed in HOL.) The mechanism described in this section provides one way of achieving this while at the same time preserving the all-important property of possessing a standard model and hence maintaining consistency.

Each closed formula q involving a single type variable α can be thought of as specifying a property $q[\tau/\alpha]$ of types τ. Its interpretation in a model is of the form

$$[\![\alpha, .q]\!] \in \prod_{X\in\mathcal{U}} [\![\alpha.bool]\!](X)\ =\ \prod_{X\in\mathcal{U}} 2\ =\ \mathcal{U}\to 2$$

which is a characteristic function on the universe, determining a subset $\{X \in \mathcal{U} : [\![\alpha, .q]\!](X) = 1\}$ consisting of those sets in the universe for which the property q holds. The most general way of ensuring the consistency of introducing a new atomic type ν satisfying $q[\nu/\alpha]$ would be to prove '$\exists\alpha.\ q$'. However, such a formula with quantification over types is not[6] a part of the HOL logic and one must proceed indirectly—replacing the formula by (a logically weaker) one that can be expressed formally with HOL syntax. The formula used is

$$(\exists f_{\alpha\to\sigma}.\ \mathsf{Type\_Definition}\ p\ f)\ \Rightarrow\ q$$

where σ is a type, $p_{\sigma\to bool}$ is a closed term and neither involve the type variable α. This formula says 'q holds of any type which is in bijection with the subtype of σ determined by p'. If this formula is provable and if the subtype is non-empty, *i.e.* if

$$\exists x_\sigma.\ p\ x$$

is provable, then it is consistent to introduce an extension with a new atomic type ν satisfying $q[\nu/\alpha]$.

In giving the formal definition of this extension mechanism, two refinements will be made. Firstly, σ is allowed to be polymorphic and hence a new type constant of appropriate arity is introduced, rather than just an atomic type. Secondly, the above existential formulas are permitted to be proved (in the theory to be extended) from some hypotheses.[7] Thus a *type specification* for a theory $\mathcal{T}$ is given by

Data

$$\langle(\alpha_1, \ldots, \alpha_n)op, \sigma, p, \alpha, \Gamma, q\rangle$$

Conditions

[6] yet!

[7] This refinement increases the applicability of the extension mechanism without increasing its expressive power. A similar refinement could have be made to the other theory extension mechanisms.

(i) (op, n) is a type constant that is not in $\mathsf{Struc}_{\mathcal{T}}$.

(ii) $\alpha_1, \ldots, \alpha_n.\sigma$ is a type-in-context with $\sigma \in \mathsf{Types}_{\mathcal{T}}$.

(iii) $p_{\sigma\to bool}$ is a closed term in $\mathsf{Terms}_{\mathcal{T}}$ whose type variables occur in $\alpha s = \alpha_1, \ldots, \alpha_n$.

(iv) α is a type variable distinct from those in αs.

(v) Γ is a list of closed formulas in $\mathsf{Terms}_{\mathcal{T}}$ not involving the type variable α.

(vi) q is a closed formula in $\mathsf{Terms}_{\mathcal{T}}$.

(vii) The sequents

$$(\Gamma \ , \ \exists x_{\sigma}.\ p\ x)$$
$$(\Gamma \ , \ (\exists f_{\alpha\to\sigma}.\ \mathsf{Type\_Definition}\ p\ f) \ \Rightarrow \ q)$$

are in $\mathsf{Theorems}_{\mathcal{T}}$.

The extension of a standard theory $\mathcal{T}$ by such a type specification is denoted

$$\mathcal{T} +_{tyspec} \langle (\alpha_1, \ldots, \alpha_n) op, \sigma, p, \alpha, \Gamma, q \rangle$$

and is defined to be the theory

$$\begin{array}{l}\langle \mathsf{Struc}_{\mathcal{T}} \cup \{(op, n)\},\\ \ \mathsf{Sig}_{\mathcal{T}},\\ \ \mathsf{Axioms}_{\mathcal{T}} \cup \{(\Gamma, q[(\alpha_1, \ldots, \alpha_n) op/\alpha])\},\\ \ \mathsf{Theorems}_{\mathcal{T}} \rangle\end{array}$$

Example To carry out the extension of INIT mentioned at the start of this section, one forms

$$\mathtt{INIT} +_{tyspec} \langle () one, bool, p, \alpha, \emptyset, q \rangle$$

where p is the term $\lambda b_{bool}.\ b$ and q is the formula $\forall f_{\beta\to\alpha}\ g_{\beta\to\alpha}.\ f = g$. Thus the result is a theory extending INIT with a new type constant *one* satisfying the axiom $\forall f_{\beta\to one}\ g_{\beta\to one}.\ f = g$.

To verify that this is a correct application of the extension mechanism, one has to check Conditions (i) to (vii) above. Only the last one is non-trivial: it imposes the obligation of proving two sequents from the axioms of INIT. The first sequent says that p defines an inhabited subset of *bool*, which is certainly the case since T witnesses this fact. The second sequent says in effect that any type α that is in bijection with the subset of *bool* defined by p has the property that there is at most one function to it from any given type β; the proof of this from the axioms of INIT is left as an exercise.

Proposition *The theory* $\mathcal{T}+_{tyspec}\langle(\alpha_1,\ldots,\alpha_n)op,\sigma,p,\alpha,\Gamma,q\rangle$ *has a standard model if the theory* $\mathcal{T}$ *does.*

Proof Write αs for $\alpha_1,\ldots,\alpha_n$, and suppose that $\alpha s' = \alpha'_1,\ldots,\alpha'_m$ is the list of type variables occurring in Γ and q, but not already in the list $\alpha s, \alpha$.

Suppose M is a standard model of $\mathcal{T}$. Since $\alpha s, .p$ is a term-in-context of type $\sigma{\to}bool$, interpreting it in M yields

$$[\![\alpha s, .p]\!]_M \in \prod_{Xs\in\mathcal{U}^n} [\![\alpha s.\sigma{\to}bool]\!]_M(Xs) = \prod_{Xs\in\mathcal{U}^n} [\![\alpha s.\sigma]\!]_M(Xs){\to}2.$$

There is no loss of generality in assuming that Γ consists of a single formula γ. (Just replace Γ by the conjunction of the formulas it contains, with the convention that this conjunction is $\top$ if Γ is empty.) By assumption on $\alpha s'$ and by Condition (iv), $\alpha s, \alpha s', .\gamma$ is a term-in-context.[8] Interpreting it in M yields

$$[\![\alpha s, \alpha s', .\gamma]\!]_M \in \prod_{(Xs,Xs')\in\mathcal{U}^{n+m}} [\![\alpha s, \alpha s'.bool]\!]_M(Xs,Xs') = \mathcal{U}^{n+m}{\to}2$$

Now $(\gamma, \exists x_\sigma.\ p\ x)$ is in $\mathsf{Theorems}_{\mathcal{T}}$ and hence by the Soundness Theorem 2.3.2 this sequent is satisfied by M. Using the semantics of $\exists$ given in Section 2.4.2 and the definition of satisfaction of a sequent from Section 2.2, this means that for all $(Xs, Xs') \in \mathcal{U}^{n+m}$ if $[\![\alpha s, \alpha s', .\gamma]\!]_M(Xs, Xs') = 1$, then the set

$$\{y \in [\![\alpha s.\sigma]\!]_M \ :\ [\![\alpha s, .p]\!](Xs)(y) = 1\}$$

is non-empty. (This uses the fact that p does not involve the type variables $\alpha s'$, so that by Lemma 4 in Section 1.3.3 $[\![\alpha s, \alpha s', .p]\!]_M(Xs, Xs') = [\![\alpha s, .p]\!]_M(Xs)$.) Since it is also a subset of a set in $\mathcal{U}$, it follows by property **Sub** of the universe that this set is an element of $\mathcal{U}$. So defining

$$S(Xs) = \begin{cases} \{y \in [\![\alpha s.\sigma]\!]_M \ :\ [\![\alpha s, .p]\!](Xs)(y) = 1\} & \text{if } [\![\alpha s, .\gamma]\!]_M(Xs, Xs') = 1, \text{ some } Xs' \\ 1 & \text{otherwise} \end{cases}$$

one has that S is a function $\mathcal{U}^n{\to}\mathcal{U}$. Extend M to a model of the signature of $\mathcal{T}'$ by defining its value at the new n-ary type constant op to be this function S. Note that the values of σ, p, γ and q in M' are the same as in M, since these expressions do not involve the new type constant op.

For each $Xs \in \mathcal{U}^n$ define i_{Xs} to be the inclusion function for the subset $S(Xs) \subseteq [\![\alpha s.\sigma]\!]_M$ if $[\![\alpha s, \alpha s', .\gamma]\!]_M(Xs, Xs') = 1$ for some Xs', and otherwise to be the function $1{\to}[\![\alpha s.\sigma]\!]_M$ sending $0 \in 1$ to $\mathrm{ch}([\![\alpha s.\sigma]\!]_M)$. Then $i_{Xs} \in (S(Xs){\to}[\![\alpha s.\sigma]\!]_{M'}(Xs))$ because $[\![\alpha s.\sigma]\!]_{M'} =$

[8] Note the two commas in $\alpha s, \alpha s', .\gamma$. The first separates the two lists of type variables; the second splits type variables from term variables.

$[\![\alpha s.\sigma]\!]_M$. Using the semantics of Type_Definition given in Section 2.4.2, one has that for any $(Xs, Xs') \in \mathcal{U}^{n+m}$, if $[\![\alpha s, \alpha s', .\gamma]\!]_{M'}(Xs, Xs') = 1$ then

$$[\![\mathsf{Type\_Definition}]\!]_{M'}([\![\alpha s.\sigma]\!]_{M'}, S(Xs))([\![\alpha s, .p]\!]_{M'})(i_{Xs}) = 1.$$

Thus M' satisfies the sequent

$$(\gamma\ ,\ \exists f_{(\alpha s)op\to\sigma}.\ \mathsf{Type\_Definition}\ p\ f).$$

But since the sequent $(\gamma, (\exists f_{\alpha\to\sigma}.\ \mathsf{Type\_Definition}\ p\ f)\ \Rightarrow\ q)$ is in $\mathsf{Theorems}_{\mathcal{T}}$, it is satisfied by the model M and hence also by the model M' (since the sequent does not involve the new type constant op). Instantiating α to $(\alpha s)op$ in this sequent (which is permissible since by Condition (iv) α does not occur in γ), one thus has that M' satisfies the sequent

$$(\gamma\ ,\ (\exists f_{(\alpha s)op\to\sigma}.\ \mathsf{Type\_Definition}\ p\ f) \Rightarrow q[(\alpha s)op/\alpha]).$$

Applying Modus Ponens, one concludes that M' satisfies $(\gamma\ ,\ q[(\alpha s)op/\alpha])$ and therefore M' is a model of $\mathcal{T}'$, as required.

An extension by type definition is in fact a special case of extension by type specification. To see this, suppose $\langle(\alpha_1, \ldots, \alpha_n)op,\ \sigma,\ p_{\sigma\to bool}\rangle$ is a type definition for a theory $\mathcal{T}$. Choosing a type variable α different from $\alpha_1, \ldots, \alpha_n$, let q denote the formula

$$\exists f_{\alpha\to\sigma}.\ \mathsf{Type\_Definition}\ p\ f$$

Then $\langle(\alpha_1, \ldots, \alpha_n)op, \sigma, p, \alpha, \emptyset, q\rangle$ satisfies all the conditions necessary to be a type specification for $\mathcal{T}$. Since $q[(\alpha_1, \ldots, \alpha_n)op/\alpha]$ is just $\exists f_{(\alpha_1,\ldots,\alpha_n)op\to\sigma}.\ \mathsf{Type\_Definition}\ p\ f$, one has that

$$\mathcal{T} +_{tydef} \langle(\alpha_1, \ldots, \alpha_n)op, \sigma, p\rangle = \mathcal{T} +_{tyspec} \langle(\alpha_1, \ldots, \alpha_n)op, \sigma, p, \alpha, \emptyset, q\rangle$$

Thus the Proposition in Section 2.5.4 is a special case of the above Proposition.

In an extension by type specification, the property q which is asserted of the newly introduced type constant need not determine the type constant uniquely (even up to bijection). Correspondingly there may be many different standard models of the extended theory whose restriction to $\mathcal{T}$ is a given model M. By contrast, a type definition determines the new type constant uniquely up to bijection, and any two models of the extended theory which restrict to the same model of the original theory will be isomorphic.

References

[1] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Computer Science and Applied Mathematics Series. Academic Press, 1986.

[2] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[3] M. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[4] F. K. Hanna and N. Daeche. Specification and verification using higher-order logic: A case study. In G. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design: Proceedings of the 1985 Edinburgh Workshop on VLSI*, pages 179–213. North-Holland, 1986.

[5] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, 1990.

[6] M. Spivey. *The Z Notation*. Prentice-Hall, 1989.