Set Theory, Higher Order Logic or Both?

Mike Gordon

University of Cambridge Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG
U.K.

Abstract. The majority of general purpose mechanised proof assistants support versions of typed higher order logic, even though set theory is the standard foundation for mathematics. For many applications higher order logic works well and provides, for specification, the benefits of type-checking that are well-known in programming. However, there are areas where types get in the way or seem unmotivated. Furthermore, most people with a scientific or engineering background already know set theory, but not higher order logic. This paper discusses some approaches to getting the best of both worlds: the expressiveness and standardness of set theory with the efficient treatment of functions provided by typed higher order logic.

1 Introduction

Higher order logic is a successful and popular formalism for computer assisted reasoning. Proof systems based on higher order logic include ALF [18], Automath [20], Coq [9], EHDM [19], HOL [13], IMPS [10], LAMBDA [11], LEGO [17], Nuprl [6], PVS [22] and Veritas [14].

Set theory is the standard foundation for mathematics and for formal notations like Z [24], VDM [15] and TLA+ [16]. Several proof assistants for set theory exist, such as Mizar [23] and Isabelle/ZF [21].

Anecdotal evidence suggests that, for equivalent kinds of theorems, proof in higher order logic is usually easier and shorter than in set theory. Isabelle users liken set theory to machine code and type theory to a high-level language.

Functions are a pervasive concept in computer science and so taking them as primitive, as is done by (most forms of) higher order logic, is natural. Higher order logic is typed. Types are an accepted and effective method of structuring data and type-checking is a powerful technique for finding errors. Types can be used to index terms and formulae for efficient retrieval. General laws become simpler when typed.

Unfortunately, certain common mathematical constructions do not fit into the type disciplines associated with higher order logic. For example, the set $\{\emptyset, \{\emptyset\}, \{\emptyset,\{\emptyset\}\}, \{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}, \ldots\}$ is traditionally used as the definition of the nateral numbers and lists are defined as the union of the infinite chain $\{\langle\rangle\} \cup (X \times \{\langle\rangle\}) \cup (X \times X \times \{\langle\rangle\}) \cup \ldots$. These sets are essentially untyped.

Furthermore, the traditional axiomatic method used in mathematics needs to be reformulated to fit into type theory [3].

There is no standard formulation of higher order logic. The various higher order logics/type theories differ widely both in the notation used and in their underlying philosophical conception of mathematical truth (e.g. intuitionistic or constructive versus classical). Automath is based on de Bruijn's own very general logic [20, A.2] (which anticipated many more recent developments). Coq and LEGO support different versions of the Calculus of Constructions. EHDM, PVS and Veritas each support different classical higher order logics with subtypes and/or dependent types. HOL and LAMBDA support similar polymorphic versions of the simple theory of types. IMPS supports monomorphic simple type theory with non-denoting terms and a theory interpretation mechanism. ALF and Nuprl support versions of Martin Löf type theory (a constructive logic with a very elaborate type system).

There is much less variation among set theories. The well known formulations are, for practical purposes, pretty much equivalent. They are all defined by axioms in predicate calculus. The only variations are whether proper classes are in the object or meta language and how many large cardinals are postulated to exist. The vast majority of mathematicians are happy with ZFC (Zermelo-Fraenkel set theory with the Axiom of Choice).

It would be wonderful if one could get the best of both worlds: the expressiveness and standardness of set theory with the efficient treatment of functions provided by typed higher order logic. In Section 2 an approach is outlined in which set theory is provided as a resource within higher order logic, and in Section 3 a reverse approach is sketched in which higher order logic is built on top of set theory. Both these approaches are explored in the context of the HOL system's version of higher order logic[1], but in the presentation I have tried to minimise the dependence on the details of the HOL logic. Some conclusions are discussed in Section 4.

2 Sets in Higher Order Logic

Set theory can be postulated inside higher order logic by declaring a type V and a constant $\in\ :\ V \times V \to \mathit{bool}$ (where *bool* is the type of the two truthvalues) and then asserting the normal axioms of set theory. The resulting theory has a consistency strength stronger than ZF, because one can define inside it a semantic function from a concrete type representing first order formulae to V

---

[1] The HOL logic is just higher order predicate calculus with a type system, due to Milner, consisting of Church's simple theory of types [5] with type variables moved from the meta-language into the object language. In Church's system, a term with type variables is actually a meta-notation – a term-schema – denoting a family of terms, whereas in HOL it is a single polymorphic term. Other versions of mechanised simple type theory (e.g. IMPS, PVS) use uninterpreted type constants instead of type variables, and then permit these to be instantiated via a theory interpretation mechanism.

such that all the theorems of ZF can be proved.[2] However, a model for higher order logic plus V can be constructed in ZF with one inaccessible cardinal. Thus the strength of higher order logic augmented with ZF-like axioms for V is somewhere between ZF and ZF plus one inaccessible cardinal.[3]

An alternative approach to using some of the linguistic facilities of higher order logic, whilst remaining essentially first order, has been investigated by Francisco Corella. His PhD thesis [8] contains a very interesting discussion of the different roles type theory can have in the formalisation of set theory.

Defining set theory inside higher order logic is very smooth. For example the Axiom of Replacement is simply:

$$\forall f\ s.\ \exists t.\ \forall y.\ y \in t \ = \ \exists x.\ x \in s \ \wedge \ y = f(x) \tag{1}$$

In traditional first-order formulations of ZF, the second-order quantification of f is not permitted, so a messy axiom scheme is needed. Another example of a useful second order quantification is the Axiom of Global Choice:[4]

$$\exists f.\ \forall s.\ \neg(s = \emptyset) \Rightarrow f(s) \in s \tag{2}$$

Standard definitional methods allow all the usual set-theoretic notions to be defined and their properties established. Such notions include, for example, the empty set, numbers, Booleans, union, intersection, finite sets, powersets, ordered pairs, products, relations, functions etc.

When set theory is axiomatised in higher order logic, the Axiom of Separation interacts nicely with λ-notation to allow $\{x \in X \mid P(x)\}$ to be represented by $\mathsf{Spec}\ X\ (\lambda x.\ P(x))$, for a suitably defined constant Spec.

More generally, $\{f(x_1, \ldots, x_n) \in X \mid P(x_1, \ldots, x_n)\}$ can be represented by $\mathsf{Spec}\ X\ (\lambda x.\ \exists x_1 \ldots x_n.\ x = f(x_1, \ldots, x_n) \ \wedge \ P(x_1, \ldots, x_n))$.

In HOL, new types are defined by giving names to non-empty subsets of existing types. Each element s of type V determines a subtype of V whose characteristic predicate is $\lambda x.\ x \in s$ (i.e. the set of all members of set s). A type σ of HOL is *represented* by $s : V$ iff there is a one-to-one function of type $\sigma \to V$ onto the subtype of V determined by s. It is straightforward to find members of V corresponding to the built-in types of HOL, for example $\{\emptyset, \{\emptyset\}\}$ represents the type of Booleans, and $\{\emptyset,\ \{\emptyset\},\ \{\emptyset,\{\emptyset\}\},\ \{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}, \ldots\ \}$ represents the natural numbers.

Standard set-theoretic constructions can be used to mimic type operators. If s_1, s_2 represent types σ_1 and σ_2, respectively, then the Cartesian product of s_1 and s_2, which will be denoted here by $s_1{\times}s_2$, represents the type $\sigma_1 \times \sigma_2$. The set of all total, single-valued relations between s_1 and s_2, denoted here by $s_1 \twoheadrightarrow s_2$,

---

[2] In HOL jargon, this is a deep embedding [4] of ZF in higher order logic plus V.

[3] These facts about consistency strength were provided by Ken Kunen (private communication).

[4] When type V is postulated in the HOL logic this is actually a theorem (because of Hilbert's ε-operator).

represents the type $\sigma_1 \to \sigma_2$.[5] Since there are lots of non-empty subsets of the class of sets, this provides a rich source of new types.

There are two ways this richness can be exploited: (i) to define types that could be defined without V in a slicker and more natural manner, and (ii) to define types that could not be defined at all.

An example of a construction that can be done in HOL without V, but is neater with it, is the definition of lists. In the current HOL system, lists of elements of type σ are represented as a subtype of the type $(\mathit{num} \to \sigma) \times \mathit{num}$, the idea being that a pair (f, n) represents the list $[f(0), f(1), \ldots, f(n{-}1)]$.[6] A more direct and natural approach uses $\langle x_1, \langle x_2, \ldots, \langle x_n, \mathsf{True}\rangle \ldots\rangle\rangle$ to represent the list $[x_1, \ldots, x_n]$ (the empty list $\langle\rangle$ can be represented by an arbitary set). However, this is not 'well-typed' since tuples with different lengths have different types. Thus this approach cannot be used to define lists in higher order logic. However, the construction can easily be performed inside V, by defining (using primitive recursion):

$$\mathsf{List}(X) = \{\langle\rangle\} \cup (X \times \{\langle\rangle\}) \cup (X \times X \times \{\langle\rangle\}) \cup \ldots \tag{3}$$

The required properties of lists are easily derived, such as the fixed-point property:

$$\forall X.\ \mathsf{List}(X) = \{\langle\rangle\} \cup (X \times \mathsf{List}(X)) \tag{4}$$

and structural induction:

$$\begin{array}{l} \forall P\ X. \\ \quad P(\langle\rangle)\ \wedge\ (\forall l \in \mathsf{List}(X).\ P(l) \Rightarrow \forall x \in X.\ P(\langle x, l\rangle)) \\ \quad \Rightarrow \\ \quad \forall l \in \mathsf{List}(X).\ P(l) \end{array} \tag{5}$$

If $s{:}V$ represents a type σ, then $\mathsf{List}(s)$ represents the type of finite lists of elements of type σ. Thus a type of lists of elements of type σ could be defined in HOL as the subtype of V determined by the predicate $\lambda x.\ x \in \mathsf{List}(s)$. This illustrates how having a type of ZF-like sets can benefit developments in higher order logic.

An example of a construction using V that would be difficult or impossible without it, is Scott's classical model D_∞ of the λ-calculus. This is a subset of an infinite product $D_0 \times D_1 \times D_2 \ldots$ where D_0 is given and D_{i+1} is a subset of the set of functions from D_i to D_i. If the D_is are represented as sets inside V (they are actually sets equipped with a complete partial order structure) then Scott's classical inverse limit can be performed directly, as has been elegantly shown by Sten Agerholm [1]. However, there seems to be no way to do it within

---

[5] Details are in the technical report 'Merging HOL with Set Theory' [12] available at `http://www.cl.cam.ac.uk/users/mjcg/papers/holst/index.html`.

[6] To ensure that the pairs (f_1, x_1) and (f_2, x_2) are equal if and only if the corresponding lists are equal, it is required that pairs (f, n) representing lists have the property that $f\ m$ equals some canonical value when m is greater than or equal to the length n of the list. The subtype consisting of such pairs (f, n) is used to define lists.

higher order logics based on simple type theory (HOL, PVS, IMPS), though it could be done in type theories with dependent products.[7]

The set-theoretic construction of lists outlined above only works if the type of the list's elements has already been represented as a set. In type systems with polymorphism, like the HOL logic, a type-operator can be defined that uniformely constructs a type $(\alpha)\mathit{list}$ of lists over an arbitrary type α (α is a type variable). Ching-Tsun Chou[8] has proposed a method of defining such polymorphic operations via set theory. He suggests that instead of having just a type V, one could have a type operator that constructed a universe of sets, $(\alpha)V$ say, over an arbitrary type α of atoms ('urelements'). The type $(\alpha)\mathit{list}$ could then be defined as a suitable subtype of $(\alpha)V$, for an arbitrary α, using a similar construction to the one given above.

To make this work, a set theory with atoms needs to be axiomatised – that is, members of $(\alpha)V$ are either atoms – i.e. non-sets – or sets (but not both). The subtype of atoms is specified to be in bijection with α. Such atoms or 'urelements' have a long history in set-theory. For example, before the technique of forcing was developed, they were used for finding models in which the Axiom of Choice doesn't hold. The axioms of set theory need to be tweaked to cope with atoms; in particular, the Axiom of Extensionality needs to be restricted to apply only to sets.[9]

I have not tried working out the details of Chou's proposal, but it strikes me as very promising. Not only does it appear to enable generic constructions to be done over an arbitrary type, by it is also philsophically satisfying in that it seems to correspond to a certain common mathematical practice in which general (e.g. categorical) developments are done directly in a kind of informal type theory, and set-theoretic constructions are restricted to where they are needed.

Adding a type V of sets (or a type operator $(\alpha)V$) to higher order logic can be compared to the already successful mechanisations of first-order set theory provided by Isabelle/ZF and Mizar. It is not clear whether axiomatising ZF in higher order logic is really better than using first order logic.

Sten Agerholm has compared Isabelle/ZF with HOL + V for the construction of D_∞ [2]. The results of his study were somewhat inconclusive. He found that using higher order logic can simplify formulation and proof, but one is faced with a "difficult question" of "which parts of the formalisation should be done in set theory and which parts in higher order logic". Chains (sequences) of sets illustrate the issue: with higher order logic chains can be represented as logical functions from the type of numbers to V, i.e. as elements of type $\mathit{num} \to V$, but with first order logic the user is not burdened with this decision as chains have to be represented inside set theory as set-functions.[10] However, if chains are

---

[7] Define types D_i (for $i = 0, 1, 2, \ldots$), then D_∞ is a subtype of the product $\prod_{n=0}^{\infty} D_i$.

[8] Private communication.

[9] Ken Kunen provided me with information (private communication) on the use of urelements in set theory.

[10] Isabelle does have a polymorphic higher order metalogic but, as Agerholm puts it:

represented as set-functions, then certain things that correspond to type checking in higher order logic need to be done using theorem proving in Isabelle/ZF. Agerholm found that Isabelle's superior theorem proving infrastructure ensured that this was not much of a burden.

3 Higher Order Logic on top of Set Theory

In the previous section set theory – in the guise of the type V – was provided as a resource within typed higher order logic. An alternative approach is to turn this upside down and to take set theory as primary and then to 'build' higher order logic on top of it. The idea is to provide the notations of higher order logic as derived forms on top of set theory and then to use set-theoretic principles to derive the axioms and rules of higher order logic. A key component of this scheme would be the development of special purpose decision procedures that correspond to typechecking.

The set-theoretic interpretation of the HOL logic is straightforward [13, Chapter 15] and can be used to provide a shallow embedding [4] of it in set theory. Each type constant corresponds to a (non-empty) set and each n-ary type operator *op* corresponds to an operation, $|\mathit{op}|$ say, for combining n sets into a set. In HOL + V such an operation on sets can be represented as a function with type $V \rightarrow V \rightarrow \cdots \rightarrow V$. For example, $|\times|$ is the Cartesian product operation $\divideontimes$ and $|\rightarrow|$ takes sets X and Y to the set $X \twoheadrightarrow Y$.

The set, $[\![\sigma]\!]$ say, corresponding to an arbitary type σ is defined inductively on the structure of σ. If σ is a a type constant c, then $[\![\sigma]\!]$ is just $|c|$. Type variables can be simply interpreted as ordinary variables ranging over V. A compound type $(\sigma_1, \ldots, \sigma_n)op_n$ is interpreted as the application of the logical function $|op_n|$ to the types corresponding to σ_1, …, σ_n – i.e. $[\![(\sigma_1, \ldots, \sigma_n)op_n]\!] = |op_n|\ [\![\sigma_1]\!] \ldots [\![\sigma_n]\!]$.

The interpretation of HOL constants in set theory is complicated by polymorphism, because the interpretation of a polymorphic constant depends on the sets corresponding to the type variables it contains. For example, the identity function in HOL is a polymorphic constant $\mathsf{I} : \alpha \rightarrow \alpha$. For any type α, I is the identity on α. The interpretation of I in set theory, $|\mathsf{I}|$ say, is the identity set-function on some set A – where the set-valued variable A corresponds to the type variable α. Thus $|\mathsf{I}|$ takes a set A and returns the identity set-function on A (so is a logical function of type $V \rightarrow V$). If c is monomorphic (has a type containing no type variables), then $|\mathsf{c}|$ will have type V. If the type of c contains n distinct type variables, then $|\mathsf{c}|$ will be a (curried) function taking n arguments of type V and returning a result of type V.

The fact that the type parameterisation of functions like I is hidden makes the HOL logic clean and uncluttered compared with set theory. One of the challenges in supporting higher order logic on top of set theory is to gracefully manage the correspondence between implicit type variables and explicit set-valued variables.

"The metalogic is meant for expressing and reasonining in logic instantiations ... not for formalising concepts in object logics".

The embedding of terms (i.e. the simply-typed λ-calculus) in set theory requires set-theoretic counterparts of function application and λ-abstraction. The application of a set-function f to an argument x is the unique y such that the pair $\langle x, y\rangle$ is a member of f (necessarily unique if f is a set-function – i.e. a total and single-valued relation). Let us write this set-theoretic application as $f{\diamond}x$, which is neatly defined using Hilbert's ε-operator by:

$$f{\diamond}x \ = \ \varepsilon y.\ \langle x, y\rangle \in f \tag{6}$$

The set-theoretic counterpart to λ-abstraction is $\mathtt{Fn}\ x \in X.\ t[x]$, where $t[x]$ is a set-valued term containing a set variable x. The definition of this notation is:

$$\mathtt{Fn}\ x \in X.\ t[x] \ = \ \{\langle x, y\rangle \in X \boldsymbol{\times} \mathsf{Image}(\lambda x.t[x])X \ \mid \ y = t[x]\} \tag{7}$$

where $\mathsf{Image}\ \mathcal{F}\ X$ is the image of set X under a logical function $\mathcal{F}$ (which exists by the Axiom of Replacement).

Each HOL term t is translated to a term $[\![t]\!]$ of type V as follows:

$$\begin{array}{lcll}
[\![x : \sigma]\!] & = & x : V & \text{(variables)} \\
[\![\,\mathsf{c} : \sigma[\sigma_1, \ldots, \sigma_n]\,]\!] & = & |\mathsf{c}|\ [\![\sigma_1]\!]\ \ldots\ [\![\sigma_n]\!] & \text{(constants)} \\
[\![\lambda x : \sigma.\ t]\!] & = & \mathtt{Fn}\ x \in [\![\sigma]\!].\ [\![t]\!] & \text{(abstractions)} \\
[\![t_1\ t_2]\!] & = & [\![t_1]\!] \diamond [\![t_2]\!] & \text{(applications)}
\end{array} \tag{8}$$

Notice that $[\![t]\!]$ lies in (monomorphic) simple type theory using just the type V. Applying this translation to the term $\forall m\ n.\ m + n \ = \ n + m$ results is:

$$\begin{array}{l}
(|\forall|\ |num|) \diamond \\
\ (\mathtt{Fn}\ m \in |num|. \\
\ \ (|\forall|\ |num|) \diamond \\
\ \ \ (\mathtt{Fn}\ n \in |num|. \\
\ \ \ \ ((|{=}|\ |num|) \diamond ((|{+}| \diamond m) \diamond n)) \diamond ((|{+}| \diamond n) \diamond m)))
\end{array} \tag{9}$$

This is a set-denoting term – i.e. a term of type V – the logical constants $\forall$ and $=$ have been 'internalised' into set-functions $|\forall|$ and $|{=}|$, respectively.

In HOL, Boolean terms can play the role of *formulae* which denote 'true' or 'false' – i.e. are judgements. Terms play this role when they are postulated as axioms or definitions or occur in theorems. When embedding higher order logic in set theory, formulae of the former should be translated to formulae of the latter. The translation of a Boolean term via (8) can be made into a logical formula of set theory by equating it to the set representing 'true' inside V, $|\mathsf{T}|$ say. Thus the formula corresponding to $\forall m\ n.\ m + n \ = \ n + m$ is obtained by equating (9) to $|\mathsf{T}|$. Using suitable definitions of the internalised constants, the resulting formula will be equivalent to:

$$\forall m\ n \in |num|.\ (|{+}| \diamond m) \diamond n = (|{+}| \diamond n) \diamond m \tag{10}$$

Free variables in formulae are interpreted as implicitly universally quantified. Thus the set-theoretic formula corresponding to $m + n \ = \ n + m$ should be equivalent to (10). If we want higher order logic formulae with free variables to translate to set-theoretic formulae with the same free variables, then the universal quantifier in (10) can be stripped off – but the restrictions that m and n be in $|bool|$ must be retained, i.e.:

$$m \in |num| \;\wedge\; n \in |num| \;\Rightarrow\; ((|{+}| \diamond m) \diamond n = (|{+}| \diamond n) \diamond m) \tag{11}$$

Thus when translating formulae, the typing of variables in higher order logic has to be converted into explicit set membership conditions in set theory.[11]

Using the scheme just described, a term tm of higher order logic is interpreted, depending on context, as the set-denoting term $[\![tm]\!]$ or the formula $x_1 \in [\![\sigma_1]\!] \wedge \ldots \wedge x_n \in [\![\sigma_n]\!] \;\Rightarrow\; [\![tm]\!] = |\mathsf{T}|$ (where the free variables in tm are $x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$). These interpretations could be handled by a parser and pretty printer (i.e. implemented as a shallow embedding).

In the HOL logic there are two primitive types $bool$ and ind and three primitive constants $\Rightarrow$, $=$ and ε. The internalised primitive types $|bool|$ and $|ind|$ are the set of two truthvalues and some arbitrary infinite set. The internalised primitive constants $|{\Rightarrow}|$, $|{=}|$ and $|\varepsilon|$ are easily defined – $|{\Rightarrow}|$ by explicitly writing down the set representing the appropriate set-function (details omitted), and the other two by:

$$\begin{aligned} |{=}|\ X \;&=\; \{\langle x,y\rangle \in X \times X \;\mid\; x = y\} \\ |\varepsilon|\ X \;&=\; \texttt{Fn}\, f.\ \mathsf{Choose}\{x \in X \;\mid\; f \diamond x = |\mathsf{T}|\} \end{aligned} \tag{12}$$

where Choose is a suitable choice operator (of logical type $V \to V$) legitimated by the Axiom of Choice.

If the shallow embedding described here is to provide the user with higher order logic, then the axioms and rules must be derived. An example of an axiom of the HOL logic is the Law of Excluded Middle: $\forall t.\ t{=}\mathsf{T} \vee t{=}\mathsf{F}$. Using (8) this is interpreted as the formula:

$$\begin{aligned} &(|\forall|\ |bool|) \diamond \\ &\quad (\texttt{Fn}\ t \in |bool|. \\ &\qquad (|\vee| \diamond (((|{=}|\ |bool|) \diamond t) \diamond |\mathsf{T}|)) \diamond (((|{=}|\ |bool|) \diamond t) \diamond |\mathsf{F}|)) \\ &=\ |\mathsf{T}| \end{aligned} \tag{13}$$

which, with suitable definitions of $|\vee|$ and $|{=}|$ will be equivalent to:

$$\forall t \in |bool|.\ t = |\mathsf{T}| \;\vee\; t = |\mathsf{F}| \tag{14}$$

which can be proved in set theory if $|bool| = \{|\mathsf{T}|, |\mathsf{F}|\}$.

An example of a rule of inference in higher order logic is β-conversion. A β-redex $(\lambda x : \sigma.\ t_1[x])t_2$ translates to: $(\texttt{Fn}\ x \in [\![\sigma]\!].\ [\![t_1[x]]\!])[\![t_2]\!]$. Now it is a theorem of set theory that:

$$y \in X \;\Rightarrow\; (\texttt{Fn}\ x \in X.\ t(x)) \;\diamond\; y \;=\; t(y) \tag{15}$$

---

[11] In HOL, different variables can have the same name as long as they have different types, so if a HOL formula contains two distinct variables with the same name, then these variables will need to be separated on translation (e.g. by priming one of them).

and hence (using higher-order matching etc.) β-conversion can be derived. Notice, however, that to apply (15) an instance of the the explicit set membership condition $x \in X$ has to be proved. In higher order logic this happens automatically via typechecking. In set theory, a special 'typechecking' theorem prover can be implemented [12, Section 6.6] using theorems such as:

$$\begin{array}{l} f \in (X \twoheadrightarrow Y) \ \wedge \ x \in X \ \Rightarrow \ f \diamond x \in Y \\[1em] (\forall x.\ x \in X \ \Rightarrow \ t[x] \in Y) \ \Rightarrow \ (\mathtt{Fn}\ x.\ t[x]) \in (X \twoheadrightarrow Y) \end{array} \qquad (16)$$

The set-theoretic versions of the non-primitive types and constants could be defined by interpreting the HOL definitions in set theory. However, this leads to a potential confusion between certain standard set-theoretic constructions, and the versions obtained by translating HOL definitions. In particular, the translation of the HOL definition of ordered pairs (product types) via (8) does not result in the familiar model of pairing used in set theory. This is an area needing further thought. Probably the best strategy is to support on top of set theory a higher order logic with more primitives than the HOL logic (e.g. with pairing built-in) – and then to augment the translation (8) to interpret the additional constructs as their natural set-theoretic counterparts.

A potentially useful feature of having set theory as the underlying logical platform is that theories in set theory can be encoded as single (large) theorems in a way that can't be done for theories in some versions of higher order logic (e.g. HOL). A theory in set theory can be regarded as an implication with the antecedents being the axioms. Constants declared in the theory will be monomorphic and can just be treated as free variables. This doesn't work in HOL because polymorphic constants can occur at different type instances of their declared (i.e. generic) type in different axioms and theorems of a theory, but a polymorphic variable must have the same type at all its occurrences in an individual theorem. Relaxing this restriction is known to make the HOL logic inconsistent [7].

Being able to code up theories as theorems could enable 'abstract theories' to be naturally supported, since theory interpretation then becomes just ordinary instantiation.

4 Discussion and Conclusions

In Section 2 it was shown how by postulating V (or, better, $(\alpha)V$) it was possible to increase the power of higher order logic, whilst still retaining its attractive features. This idea has been explored in some detail by Sten Agerholm and seems a success.

In Section 3 a more radical idea is outlined in which higher order logic is 'mounted' on top of set theory as a derived language (via a shallow embedding). If this can be made to work – and it is not yet clear whether it can – then it would seem to offer the best of the two worlds. Users could choose to work entirely within higher order logic, but they could also choose to stray into the rich pastures of set theory. Furthermore, users could add additional constructs themselves without having to modify the core system. Thus, for example, the record and dependent subtypes of PVS could be added (type correctness conditions just being handled by normal theorem proving). However, this is currently all fantasy: it still remains to see whether it is possible to get an efficient and well-engineered type theory via a shallow embedding into set theory.

In conclusion, my answer to the question posed as the title of this paper is that both set theory and higher order logic are needed. In the short term useful things can be done by adding a type of sets to higher order logic, but building higher order logic on top of set theory is an exciting research challenge that promises a bigger payoff.

Acknowledgements

Sten Agerholm, Ching-Tsun Chou, Francisco Corella, John Harrison, Tom Melham, Larry Paulson and Andy Pitts provided various kinds of help in the development of the ideas described here.

References

1. S. Agerholm. Formalising a model of the λ-calculus in HOL-ST. Technical Report 354, University of Cambridge Computer Laboratory, 1994.
2. S. Agerholm and M.J.C. Gordon. Experiments with ZF Set Theory in HOL and Isabelle. In E. T. Schubert, P. J. Windley, and J. Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications: 8th International Workshop*, volume 971 of *Lecture Notes in Computer Science*, pages 32–45. Springer-Verlag, September 1995.
3. Jackson Paul B. Exploring abstract algebra in constructive type theory. In A. Bundy, editor, *12th Conference on Automated Deduction*, Lecture Notes in Artifical Intelligence. Springer, June 1994.
4. R. J. Boulton, A. D. Gordon, M. J. C. Gordon, J. R. Harrison, J. M. J. Herbert, and J. Van Tassel. Experience with embedding hardware description languages in HOL. In V. Stavridou, T. F. Melham, and R. T. Boute, editors, *Theorem Provers in Circuit Design: Theory, Practice and Experience: Proceedings of the IFIP TC10/WG 10.2 International Conference*, IFIP Transactions A-10, pages 129–156. North-Holland, June 1992.
5. A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.
6. R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, 1986.
7. Thierry Coquand. An analysis of Girard's paradox. In *Proceedings, Symposium on Logic in Computer Science*, pages 227–236, Cambridge, Massachusetts, 16–18 June 1986. IEEE Computer Society.
8. Francisco Corella. Mechanizing set theory. Technical Report 232, University of Cambridge Computer Laboratory, August 1991.

9. G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. The Coq proof assistant user's guide - version 5.8. Technical Report 154, INRIA-Rocquencourt, 1993.
10. W. M. Farmer, J. D. Guttman, and F. Javier Thayer. IMPS: An interactive mathematical proof system. *Journal of Automated Reasoning*, 11(2):213–248, 1993.
11. S. Finn and M. P. Fourman. *L2 – The LAMBDA Logic.* Abstract Hardware Limited, September 1993. In LAMBDA 4.3 Reference Manuals.
12. M. J. C. Gordon. Merging HOL with set theory. Technical Report 353, University of Cambridge Computer Laboratory, November 1994.
13. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem-proving Environment for Higher-Order Logic.* Cambridge University Press, 1993.
14. F. K. Hanna, N. Daeche, and M. Longley. Veritas+: a specification language based on type theory. In M. Leeser and G. Brown, editors, *Hardware specification, verification and synthesis: mathematical aspects*, volume 408 of *Lecture Notes in Computer Science*, pages 358–379. Springer-Verlag, 1989.
15. C. B. Jones. *Systematic Software Development using VDM.* Prentice Hall International, 1990.
16. L. Lamport and S. Merz. Specifying and verifying fault-tolerant systems. In *Proceedings of FTRTFT'94*, Lecture Notes in Computer Science. Springer-Verlag, 1994. See also: `http://www.research.digital.com/SRC/tla/papers.html#TLA+`.
17. Z. Luo and R. Pollack. LEGO proof development system: User's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh, LFCS, Computer Science Department, University of Edinburgh, The King's Buildings, Edinburgh, EH9 3JZ, May 1992.
18. L. Magnusson and B. Nordström. The ALF proof editor and its proof engine. In *Types for Proofs and Programs: International Workshop TYPES '93*, pages 213–237. Springer, published 1994. LNCS 806.
19. P. M. Melliar-Smith and John Rushby. The enhanced HDM system for specification and verification. In *Proc. Verkshop III*, volume 10 of *ACM Software Engineering Notes*, pages 41–43. Springer-Verlag, 1985.
20. R. P. Nederpelt, J. H. Geuvers, and R. C. De Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and The Foundations of Mathematics.* North Holland, 1994.
21. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994.
22. PVS Web page. `http://www.csl.sri.com/pvs/overview.html`.
23. Piotr Rudnicki. *An Overview of the MIZAR Project.* Unpublished manuscript; but available by anonymous FTP from `menaik.cs.ualberta.ca` in the directory `pub/Mizar/Mizar_Over.tar.Z`, 1992.
24. J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall International Series in Computer Science, 2nd edition, 1992.

This article was processed using the LaTeX macro package with LLNCS style