Higher Order Logic versus Set Theory

- Higher order logic is based on functions
  - primitive notions are
    - application $f\ x$
    - abstraction $\lambda x.\ t$
- Traditional 'text book' mathematics is founded on set theory
  - primitive notions are
    - membership $x \in S$
    - set construction principles – e.g. $\{x \mid t\}$

Type Theory is Popular

- Automath uses de Bruijn's own logic
  - anticipated much recent work
- HOL, Isabelle/HOL, TPS and Lambda
  - support classical higher order logics with simple types
- IMPS
  - supports simple types with non-denoting terms
- PVS and Veritas
  - classical higher order logics with dependent types
- Coq and LEGO
  - versions of the Calculus of Constructions
- ALF and Nuprl
  - versions of Martin Löf type theory

Why is Type Theory Popular?

- Functions are a natural primitive
  - tedious to derive laws like β-conversion
  - functional programming idiom popular
- Types improve specification
  - document overall structure
  - catch errors early
- Laws are simpler with types
  - $x + 0 = x$ is an equation if x has type *num*
  - without types: $x \in \mathbb{N} \Rightarrow x + 0 = x$
    - such a conditional is harder to use
- Simple set theory can be represented in type theory

Set Theory in HOL

- Represent a set by its characteristic function
  - set of elements of type σ is a predicate on σ
  - $\{x : \sigma \mid P(x)\}$ represented by $P : \sigma \to bool$
- All elements of a set have the same type
  - in practice often only need simple set operations on a type
- Can define usual set theoretic operations

Set Theory	Higher Order Logic
$\emptyset$	$\lambda x.\ \mathsf{F}$
$\{a\}$	$\lambda x.\ x = a$
$\{x \mid \mathcal{P}(x)\}$	$\lambda x.\ \mathcal{P}(x)$
$x \in P$	$P(x)$
$P \cup Q$	$\lambda x.\ P(x) \vee Q(x)$
$P \cap Q$	$\lambda x.\ P(x) \wedge Q(x)$

Set Theoretic Toolkit in HOL

- HOL has lots of set theoretic infrastructure in `setLib`
  - standard properties relating of $\in$, $\subset$, $\supset$, $=$, $\cup$, $\cap$ etc
  - properties of finite and infinite sets
    * $\mathsf{Finite}\ s\ =\ \forall P.\ P\emptyset \wedge (\forall s.\ P\ s \Rightarrow \forall e.P(\{e\} \cup s)) \Rightarrow P\ s$
    * $\mathsf{Infinite}\ s\ =\ \forall t.\ \mathsf{Finite}\ t \Rightarrow t \subseteq s \Rightarrow t \subset s$
  - properties of the size of finite sets
    * $$\begin{array}{l}(\mathsf{Size}\ \emptyset = 0)\\ \wedge\\ \forall s.\ \mathsf{Finite}\ s \Rightarrow\\ \quad\quad \forall x.\ \mathsf{Size}\ (\{x\} \cup s) = (\mathtt{if}\ x \in s\ \mathtt{then}\ \mathsf{Size}\ s\ \mathtt{else}\ \mathsf{Size}\ s + 1)\end{array}$$
- Sufficient for ordinary set theoretic reasoning
- Not the traditional textbook set theory though

Traditional Set Theories

- There are several classical formulations of set theory
  - ZF: Zermelo-Fraenkel set theory
    - most popular: only has sets, needs axiom schemes
  - NBG: Neumann-Bernays-Gödel
    - finite axiomatisation using sets and classes
  - MKM: Mostowski-Kelley-Morse
    - more powerful version of NBG
  - NF: Quine's New Foundations
    - weird system not much used but theoretically interesting
- Recommended book

  The Logical Foundations of Mathematics

  William S. Hatcher, Pergamon Press, 1982

  ISBN 0-08-025800-X (out of print)

Set Theory Axioms

- Axioms assert existence of a universe V of sets
  - start with the empty set $\emptyset$
  - new sets using union, powerset etc
  - comprehension: S a set implies $\{x \in S \mid P(x)\}$ a set
  - everything is a set – no separation into types
  - Von Neuman numerals: $0 = \{\}$, $1 = \{\{\}\}$, $2 = \{\{\{\}\}\}$, . . .
- Logicians worry about consistency of axioms
  - $\{x \mid x \notin x\} \in \{x \mid x \notin x\} \Leftrightarrow \{x \mid x \notin x\} \notin \{x \mid x \notin x\}$
  - Russell's Paradox

Attractions of Untyped ZF-type Set Theory

- More standard
  - taught in school and university
- Underlies popular specification methods
  - Z, VDM, TLA+ ...
- Well understood axiomatisations (e.g. ZF)
  - stable compared with type theory
  - lots of metatheory
- More expressive than typed set theory
  - Von Neuman numerals: $0 = \{\}$, $1 = \{\{\}\}$, $2 = \{\{\{\}\}\}$, $\ldots$
  - construction of D_∞ by Sten Agerholm
- Can be effectively mechanised
  - Isabelle/ZF, Mizar, EVES ...

First-order Versus Higher-order Axiomatisation

- Usual axioms of set theory (ZF, NBG etc) are first order (FOL)
- Can formulate axioms in higher order logic (HOL)
  - examples given later
- ZF axioms in HOL makes them stronger than first order ZF
  - can define a deep embedding of first order ZF language
    - then define a semantics of first order ZF formulae in V
    - then prove ZF axioms as theorems
- Inaccessible cardinal + ZF is stronger than HOL + V
  - can model HOL + V inside an inaccessible cardinal
- FOL + ZF $\subset$ HOL + V $\subset$ FOL + ZF + large cardinal
  - I am not a set theory expert!
  - details thanks to email from noted set theorist Ken Kunen

Why Consider Higher Order Axiomatization of Set Theory?

- Formulating axioms in HOL logic makes them more readable
  - first order schemas replaced by single higher order terms
  - examples given later
- HOL provides useful infrastructure
  - Axiom of Choice: $\varepsilon x.\ x \in s$
  - definitional mechanisms for defining constants
- Must distinguish higher order syntax from higher order axioms
  - Isabelle/ZF higher order syntax equivalent to FOL ZF
  - HOL + V higher order axioms not equivalent to FOL ZF
  - see Corella's 1991 Cambridge PhD *Mechanizing Set Theory*

Two Ways of Using HOL + V

1 Utilise V as a resource for HOL

- Define datatypes via set classical set theoretic methods

2 Build a copy of HOL inside V

- Makes HOL type system 'soft' and extensible
  - add more powerful types (e.g. Σ and Π types)
- Platform for experiments
  - exploring spectrum: HOL $\longleftrightarrow$ PVS $\longleftrightarrow$ Nuprl/Coq

Definition of V: Primitive and Derived Notions

- Only the binary operator $\in$ is primitive
  - postulate type V and constant $\in$: $V \times V \to \mathit{bool}$
- Predicates can be defined
- Subset $\subseteq$ defined by:

  $\mathtt{s} \subseteq \mathtt{t} = \forall\mathtt{x}.\ \mathtt{x} \in \mathtt{s} \Rightarrow \mathtt{x} \in \mathtt{t}$
- Proper subset $\subset$ defined by:

  $\mathtt{s} \subset \mathtt{t} = \mathtt{s} \subseteq \mathtt{t} \wedge \neg(\mathtt{s}=\mathtt{t})$
- Set operators can be justified by set theory axioms
- Empty set axiom

  $\exists\mathtt{s}.\ \forall\mathtt{x}.\ \neg(\mathtt{x} \in \mathtt{s})$

  legitimates $\varnothing$ defined to have the property $\forall x.\ \neg(x \in \varnothing)$

Axioms and Definitions

- **Extensionality**

  $\forall$s t. (s = t) $\equiv$ ($\forall$x. x $\in$ s = x $\in$ t)

- **Empty set**

  $\exists$s. $\forall$x. $\neg$(x $\in$ s)

  justifies definition of $\emptyset$

- **Union**

  $\forall$s. $\exists$t. $\forall$x. x $\in$ t $\equiv$ ($\exists$u. x $\in$ u $\wedge$ u $\in$ s)

  justifies definition of $\bigcup$

  $\forall$s x. x $\in$ $\bigcup$s = ($\exists$u. x $\in$ u $\wedge$ u $\in$ s)

Axioms and Definitions – 2

- **Power sets**

  $\forall \mathtt{s}.\ \exists \mathtt{t}.\ \forall \mathtt{x}.\ \mathtt{x} \in \mathtt{t} \equiv \mathtt{x} \subseteq \mathtt{s}$

  justifies definition of $\mathbb{P}$

  $\forall \mathtt{s}\ \mathtt{x}.\ \mathtt{x} \in \mathbb{P}\ \mathtt{s} = \mathtt{x} \subseteq \mathtt{s}$

- **Separation**

  $\forall \mathtt{p}\ \mathtt{s}.\ \exists \mathtt{t}.\ \forall \mathtt{x}.\ \mathtt{x} \in \mathtt{t} \equiv \mathtt{x} \in \mathtt{s} \wedge \mathtt{p}\ \mathtt{x}$

  Note: not first order!

  justifies the notation $\{\mathtt{x} \in \mathtt{s} \mid \mathcal{P}(\mathtt{x})\}$

  which can be used to define $\cap$ where:

  $\mathtt{s} \cap \mathtt{t} = \{\mathtt{x} \in \mathtt{s} \mid \mathtt{x} \in \mathtt{t}\}$

- **Foundation** (sometimes omitted)

  $\forall \mathtt{s}.\ \neg(\mathtt{s} = \emptyset) \Rightarrow \exists \mathtt{x}.\ \mathtt{x} \in \mathtt{s} \wedge (\mathtt{x} \cap \mathtt{s} = \emptyset)$

First Order Versus Higher Order Formulation of Set-theoretic Axioms

- Notation
  - $\equiv$ is "if and only if" – i.e. $=$ restricted to booleans
  - $\forall\, x \in s.\ \mathcal{P}(x)$ means $\forall\, x.\ x \in s \Rightarrow \mathcal{P}(x)$
  - $\exists\, x \in s.\ \mathcal{P}(x)$ means $\exists\, x.\ x \in s \wedge \mathcal{P}(x)$
- First Order Axiom of Replacement

  $\forall$s. ($\forall$x $\in$ s. $\forall$y z. ϕ(x,y) $\wedge$ ϕ(x,z) $\Rightarrow$ y = z)
  $\quad\Rightarrow$
  $\quad\exists$t. $\forall$y. y $\in$ t $\equiv$ $\exists$x $\in$ s. ϕ(x,y)

  - a first order axiom schema: $\phi(x, y)$ ranges over formulae
- Higher Order Axiom of Replacement

  $\forall$f s. $\exists$t. $\forall$y. y $\in$ t = $\exists$x $\in$ s. y = f x

  - a single term expressing same concept as first order schema
    * type V
    * constant $\in\ : V \times V \rightarrow \mathit{bool}$

Axioms and Definitions – 3

- **Replacement**

  $\forall f\ s.\ \exists t.\ \forall y.\ y \in t \equiv \exists x.\ x \in s \wedge (y = f\ x)$

  legitimates Image where:

  $\forall f\ s\ y.\ y \in \mathsf{Image}\ f\ s = \exists x.\ x \in s \wedge (y = f\ x)$

  and the notation $\{s\}$ where:

  $\forall s.\ \{s\} = \mathsf{Image}\ (\lambda x.s)\ (\mathbb{P}\ \varnothing)$

  which satisfies:

  $\forall s\ x.\ x \in \{s\} = (x = s)$

- **Infinity**

  $\exists s.\ \varnothing \in s \wedge \forall x.\ x \in s \Rightarrow (x \cup \{x\}) \in s$

  justifies Inf: $\varnothing \in \mathsf{Inf} \wedge \forall x.\ x \in \mathsf{Inf} \Rightarrow (x \cup \{x\}) \in \mathsf{Inf}$

Summary of ZF Axioms in HOL

Extensionality	$\forall$s t. (s = t) $\equiv$ ($\forall$x. x $\in$ s = x $\in$ t)
Empty set	$\exists$s. $\forall$x. $\neg$(x $\in$ s)
Union	$\forall$s. $\exists$t. $\forall$x. x $\in$ t $\equiv$ ($\exists$u. x $\in$ u $\wedge$ u $\in$ s)
Power sets	$\forall$s. $\exists$t. $\forall$x. x $\in$ t $\equiv$ x $\subseteq$ s
Separation	$\forall$p s. $\exists$t. $\forall$x. x $\in$ t $\equiv$ x $\in$ s $\wedge$ p x
Foundation	$\forall$s. $\neg$(s = $\emptyset$) $\Rightarrow$ $\exists$x. x $\in$ s $\wedge$ (x $\cap$ s = $\emptyset$)
Replacement	$\forall$f s. $\exists$t. $\forall$y. y $\in$ t $\equiv$ $\exists$x. x $\in$ s $\wedge$ (y = f x)
Infinity	$\exists$s. $\emptyset$ $\in$ s $\wedge$ $\forall$x. x $\in$ s $\Rightarrow$ (x $\cup$ $\{$x$\}$) $\in$ s

Review and some Related Work on ZF in HOL

- Postulate a type V that satisfies the ZF axioms
  - this guaranties lots of sets exist
- Result is ordinary set theory within higher order logic
  - more sets than ordinary first order formulation
  - HOL provides powerful definitional mechanisms
- Larry Paulson's work on Isabelle/ZF
  - demonstrates that set theory is practical
  - many *tour de forces* of proof (e.g. `Vrec`)
  - Agerholm comparison of first & higher order axiomatisations
- Corella's 1991 Cambridge PhD *Mechanizing Set Theory*
  - discusses uses of type theory
    - for higher order syntax (Isabelle/ZF)
    - as the underlying logic (HOL-ST)

Recall: Two Ways of Using HOL + V

1 Utilise V as a resource for HOL

- Define datatypes via set classical set theoretic methods

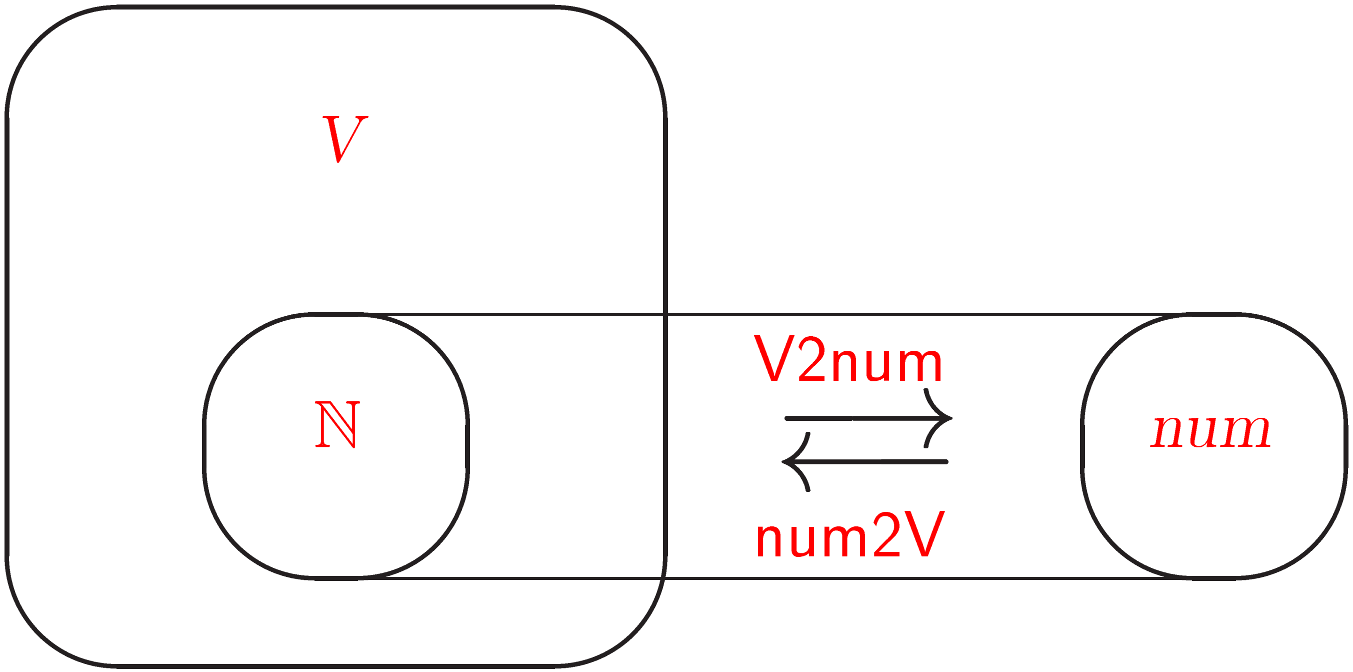
2 Build a copy of HOL inside V

- Makes HOL type system 'soft' and extensible
  - add more powerful types (e.g. Σ and Π types)
- Platform for experiments
  - exploring spectrum: HOL $\longleftrightarrow$ PVS $\longleftrightarrow$ Nuprl/Coq

1 V as a Resource for HOL

- Example: construction of type of lists of numbers
- List are already defined in HOL98
  - definition from scratch quite tricky and non-obvious
  - example here illustrates idea – not a killer ap for V
- First construct numbers in V
- Then define lists of numbers
- Constructing polymorphic lists raises interesting issues
  - α *list* rather than *num list*

Representing Numbers in V



Take $\mathbb{N}$ to be Von Neuman Numbers

- Define by recursion (in HOL in logic)

$$\begin{array}{rcl} \mathsf{num2V}\ 0 & = & \varnothing \\ \mathsf{num2V}(n{+}1) & = & (\mathsf{num2V}\ n)\ \cup\ \{\mathsf{num2V}\ n\} \end{array}$$

- Recursion done 'outside' set theory
- Function $\mathsf{num2V} : \mathit{num} \to V$ is injective
- Set-theoretic numbers $\mathbb{N}$ are range of num2V

$$\mathbb{N} = \{x \in \mathsf{Inf} \mid \exists\, n.\ x\ =\ \mathsf{num2V}\ n\}$$

- Function $\mathsf{V2num} : V \to \mathit{num}$ is inverse of num2V on $\mathbb{N}$

$$\forall\, n.\ \mathsf{V2num}(\mathsf{num2V}\ n)\ =\ n$$

- Can 'copy' operations from HOL logic to V

$$x \oplus y = \mathsf{num2V}((\mathsf{V2num}\ x) + (\mathsf{V2num}\ y))$$

Lists in V

- Traditionally in HOL $(\sigma)\mathit{list}$ a subtype of $(\mathit{num} \rightarrow \sigma) \times \mathit{num}$
  - $[x_1;\ x_2;\ \ldots;\ x_m]$ represented as pair (f, m)
  - where $f(i) = x_{i+1}$ $(0 \leq i < m)$
- Simpler representation of $[x_1;\ x_2;\ \ldots;\ x_m]$ is $(x_1, (x_2, (\ \cdots\ )))$
  - but this has a different type for each different length m
  - so can't be used in HOL
- However, inside untyped V the simpler definition is possible

Pairs in V

- Define $\langle x, y \rangle = \{\{x\}, \{x, y\}\}$
  - normal properties of pairing easily follow
- Define

$$\mathtt{X} \divideontimes \mathtt{Y} \;=\; \{\langle \mathtt{x}, \mathtt{y} \rangle \in \mathbb{P}(\mathbb{P}(\mathtt{X} \cup \mathtt{Y})) \mid \mathtt{x} \in \mathtt{X} \wedge \mathtt{y} \in \mathtt{Y}\}$$

- Define

$$\begin{aligned} \mathsf{False} &= \varnothing \\ \mathsf{True} &= \{\varnothing\} \\ \mathsf{Bool} &= \{\mathsf{True}, \mathsf{False}\} \end{aligned}$$

Constructing Lists via Countable Unions in V

- A countable sequence of sets is a function $s : num \rightarrow V$
- The union of the sequence is $s(0) \cup s(1) \cup \cdots \cup s(n) \cup \cdots$
- This is the 'big union' ($\bigcup$) of the image of $\mathbb{N}$ under $s \circ \mathsf{V2num}$

  $\mathsf{UnionSeq\ s} = \bigcup(\mathsf{Image(s\ o\ V2num)}\mathbb{N})$

- The notation $\bigcup_n t[n]$ abbreviates $\mathsf{UnionSeq}(\lambda\, n.\ t[n])$
- Define

  $(\mathsf{FiniteList\ X\ 0} = \{\mathsf{True}\})$
  $(\mathsf{FiniteList\ X\ (n+1)} = \mathsf{FiniteList\ X\ n} \cup (\mathsf{X} \divideontimes \mathsf{FiniteList\ X\ n}))$

  $\mathsf{List\ X} = \bigcup_{\mathsf{n}} \mathsf{FiniteList\ X\ n}$

Properties of Lists

- Follows that

$$\forall \mathtt{X}.\ \mathsf{List}\ \mathtt{X} = \{\mathsf{True}\} \cup (\mathtt{X} \divideontimes \mathsf{List}\ \mathtt{X})$$

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

- Can define HOL list of numbers
  - as a subtype of V
  - by predicate $\lambda s.\ s \in \mathsf{List}\ \mathbb{N}$
- What about (α)*list*?

ZFU instead of ZF

- List X is the set of finite lists of members of X
- To define a set to represent $(\sigma)list$ need set representing σ
- Ching Tsun Chou suggests set theory polymorphic over atoms
  - i.e. a type operator $(\alpha)V$
  - represents ZFU with the atoms isomorphic to type α
- Polymorphic list type could be defined set theoretically
- Seems like an interesting idea to explore
  - not done any work on this
  - ZFU well understood, but more messy than ZF
    - need a predicate to distinguish sets from atoms
    - extensionality restricted to sets (atoms have no elements)

      $\forall$s t. IsSet s $\wedge$ IsSet t $\Rightarrow$ ((s = t) $\equiv$ ($\forall$x. x $\in$ s = x $\in$ t))

Sten Agerholm's Experiments with V

- Lists can be constructed without V
- Other constructions are hard or impossible without V
- Sten Agerholm constructed Scott's λ-calculus model D_∞ in V
  - could not be done in pure HOL (I think)
- Comparison with Isabelle/ZF done
  - had to think about what to do inside versus outside V
  - e.g. chains could be HOL functions or pure sets
  - can benefit from HOL metalanguage
  - but also more decisions to make

V as a Resource for HOL – Conclusions

- Having a ZF set theory inside HOL is powerful
  - possibility of using textbook constructions
  - then exploiting in higher order logic
  - seem to be benefits over first order logic
- Type V not definitional
  - ZF seems pretty trustworthy though!
  - ZFU maybe a bit more dodgy?
- Conclusion: case for V not proven
  - more experiments (e.g. with $(\alpha)V$) needed

Recall the Two Possible Ways of Using HOL + V

1 Utilise V as a resource for HOL

- this has just been discussed

2 Build a copy of HOL inside V

- Makes HOL type system 'soft' and extensible
  - add more powerful types (e.g. Σ and Π types)
- Platform for experiments
  - exploring spectrum: HOL $\longleftrightarrow$ PVS $\longleftrightarrow$ Nuprl/Coq

A Soft HOL Inside V

- The HOL kernel is ‘hard coded’ in ML
  - difficult and logically hazardous to make changes
- Higher oder logic has a set theoretic semantics
  - due to Andrew Pitts (DSTO contract)
  - could do a semantic embedding of HOL inside V
- Dream: a single system combining
  - power and simplicity of ZF-style set theory
  - types and functions as in higher order logic
  - strong typechecking, but extensible soft types

An Experiment to Combine Higher Order Logic and Set Theory

- Start with higher order logic
  - simple type theory as in HOL
- Add set theory
  - axiomatise a type V using ZF axioms
- Embed higher order logic into set theory
  - typechecking derived – not ‘hardwired’
  - ‘soft’ types are flexible – Σ and Π can be added

First Order versus Higher Order Set Theory

- Could use first order set theory (e.g. Isabelle/ZF)
- First order: everything inside set theory
  - well-founded recursion
    - Isabelle's `wfrec` used to define numbers
  - recursion on rank of set
    - Isabelle's `Vrec` used to define lists
  - these methods powerful, but 'advanced'
- Higher order: constructions possible in logic
  - use normal HOL methods
  - then map into type V
  - more 'high level' and light weight?

First Order Metalogic versus Higher Order Metalogic

- Can 'talk about' sets using HOL
  - $(s_1, s_2) : V \times V$ is a pair of sets
    - $\langle s_1, s_2 \rangle : V$ is a set representing a pair
  - $f : num \to V$ is a sequence of sets
    - $f\ \mathsf{T}$ rejected by typechecking
  - $\bigcup f$ is an infinite union
    - $\bigcup : (num \to V) \to V$
- Sten Agerholm has interesting data from D_∞
- Like informal mathematics
  - constructions done in a higher order logic
  - use of set theory localised to where needed

Function Application and Abstraction inside V

- Functions represented by sets of ordered pairs
  - i.e. functions in set theory are sets
- Set-theoretic function application:

$$f \diamond x \;=\; \varepsilon y.\ \langle x, y\rangle \in f$$

  - ε is Hilbert's choice operator
- Set-theoretic function abstraction:

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

  - $\times$ is set-theoretic Cartesian Product
  - $\mathsf{Image}\ \mathcal{F}\ X$ is image of set X under $\mathcal{F}$
    - exists via Axiom of Replacement
- Set-theoretic version of β-reduction:

$$y \in X \;\Rightarrow\; (\boldsymbol{\lambda} x \in X.\ t[x]) \diamond y \;=\; t[y]$$

Sets of Relations and Functions

- Relations:

  $$\texttt{X} \leftrightarrow \texttt{Y} = \mathbb{P}(\texttt{X} \times \texttt{Y})$$

- Functions (partial and total):

  $$\begin{array}{l} \texttt{X} \nrightarrow \texttt{Y} = \\ \quad \{\texttt{f} \in \texttt{X} \leftrightarrow \texttt{Y} \mid \\ \qquad \forall \texttt{x y1 y2}. \\ \qquad\quad \langle \texttt{x}, \texttt{y1} \rangle \in \texttt{f} \wedge \langle \texttt{x}, \texttt{y2} \rangle \in \texttt{f} \\ \qquad\quad \Rightarrow \\ \qquad\quad (\texttt{y1} = \texttt{y2})\} \end{array}$$

- Total functions:

  $$\begin{array}{l} \texttt{X} \twoheadrightarrow \texttt{Y} = \\ \quad \{\texttt{f} \in \texttt{X} \nrightarrow \texttt{Y} \mid \\ \qquad \forall \texttt{x}.\ \texttt{x} \in \texttt{X} \\ \qquad\qquad \Rightarrow \\ \qquad\qquad \exists \texttt{y}.\ \texttt{y} \in \texttt{Y} \wedge \langle \texttt{x}, \texttt{y} \rangle \in \texttt{f}\} \end{array}$$

Types as Sets

- Set-theoretic type operators:
  - $X \times Y$ – Cartesian product of X and Y
  - $X \twoheadrightarrow Y$ – set of functions from X to Y
  - $\mathsf{List}\ X$ – set of lists over X
- Particular types:

  $\mathsf{true} \in \mathsf{Bool}$, $\mathsf{false} \in \mathsf{Bool}$, $|\wedge| \in \mathsf{Bool} \times \mathsf{Bool} \twoheadrightarrow \mathsf{Bool}$, $|+| \in \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$

- General typechecking theorems:

$$x \in X \ \wedge\ x \in Y \ \Rightarrow\ \langle x, y\rangle \in X \times Y$$

$$f \in (X \twoheadrightarrow Y) \ \wedge\ x \in X \ \Rightarrow\ f \diamond x \in Y$$

$$(\forall x.\ x \in X \ \Rightarrow\ t[x] \in Y) \ \Rightarrow\ (\boldsymbol{\lambda} x.\ t[x]) \in (X \twoheadrightarrow Y)$$

$$x_1 \in X \wedge \cdots \wedge x_n \in X \ \Rightarrow\ \langle x_1, \ldots, \langle x_n, \varnothing\rangle \ldots\rangle \in \mathsf{List}\ X$$

Translation of HOL Types to Sets

- Types variable α translates to ordinary an variable $\alpha : V$
- Type constant c translates to term $|c|$
  - e.g. $|bool| = \{\mathsf{true}, \mathsf{false}\}$
- Type operator op translates to function $|op|$
  - if op_n is an n-ary operator then
  $$|op_n| : \underbrace{V \to V \to \cdots \to V}_{n \text{ parameters}} \to V$$
  - $|\times| \ = \ \divideontimes$ where $\divideontimes : V \to V \to V$
  - $|\to| \ = \ \twoheadrightarrow$ where $\twoheadrightarrow : V \to V \to V$
  - $|list| = \mathsf{List}$ where $\mathsf{List} : V \to V$
- Type σ recursively translated to term $[\![\sigma]\!]$

$$[\![(\sigma_1, \ldots, \sigma_n)op_n]\!] = |op_n| \ [\![\sigma_1]\!] \ldots [\![\sigma_n]\!]$$

Embedding Constants in V

- Interpretation of constant c is $|\mathsf{c}|$
- If c is monomorphic then $|\mathsf{c}|$ will have type V
  - e.g. $|\mathsf{F}| = |0| = \varnothing$
- If type of c contains n distinct type variables
  - $|\mathsf{c}|$ will be a (curried) function:
    * taking n arguments of type V
    * returning a result of type V
- Example: $\mathsf{I} : \alpha \to \alpha$
  - for any type α, I is the identity on α
  - $|\mathsf{I}|$ is the identity set-function on some set A
  - set-valued variable A corresponds to the type variable α
  - $|\mathsf{I}| : V \to V$ maps set A to identity set-function on A

Polymorphism

- Consider identity function $\mathsf{I} : \alpha \to \alpha$
  - type variable α ranges over sets A
  - identity set-function on A:

$$(|\mathsf{I}|\ A) \ = \ \{\langle x, y\rangle \in A \times A \ \mid \ x = y\}$$

  - type variables represented by set variables
- Compare with the identity operator $\hat{\mathsf{I}}$ on sets
  - $\hat{\mathsf{I}} \ = \ \lambda x : V.\ x$
  - $\hat{\mathsf{I}} : V \to V$
  - $x \in X \ \Rightarrow \ \hat{\mathsf{I}}\ x = (|\mathsf{I}|\ X) \diamond x$
- $\hat{\mathsf{I}}$ doesn't need explicit parameter
  - 'polymorphic' operators like $\hat{\mathsf{I}}$ convenient
  - use function application rather than $\diamond$

HOL Polymorphism versus Set Parameters

- Type parameterisation of functions like I is hidden
  - HOL logic clean and uncluttered compared with set theory
- Challenge
  - gracefully manage
    - correspondence between implicit type variables
    - and explicit set-valued variables
  - standard problem in type theories like Nuprl and Coq
    - various type variable omitting conventions used
  - many examples from Isabelle/ZF

Embedding HOL Terms in V

- HOL term t is translated to a term $[\![t]\!]$ of type V

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

- Example: applying this translation to $\forall\, m\ \ n.\ m+n\ = \ n+m$

$$
\begin{array}{l}
(|\forall|\ |\mathbb{N}|) \diamond \\
\ (\boldsymbol{\lambda}\ m \in |\mathbb{N}|. \\
\ \ (|\forall|\ |\mathbb{N}|) \diamond \\
\ \ \ (\boldsymbol{\lambda}\ n \in |\mathbb{N}|. \\
\ \ \ \ ((|{=}|\ |\mathbb{N}|) \diamond ((|{+}| \diamond m) \diamond n)) \diamond ((|{+}| \diamond n) \diamond m)))
\end{array}
$$

Typechecking is Syntactic

- Suppose

$$\begin{array}{l} \mathsf{false} = \varnothing \\ \mathbb{N} = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}, \ \ldots \ \} \end{array}$$

  then $\mathsf{false} \in \mathbb{N}$ because $\mathsf{false} = |0|$

- Want typechecking to reject $\mathsf{false} \ \oplus \ |3|$
  - $x \oplus y = |+| \diamond \langle x, y\rangle$
  - $|+| \in \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$
  - theorem proving reduces $\mathsf{false} \ \oplus \ |3| \in \mathbb{N}$ to
    - $\mathsf{false} \in \mathbb{N}$
    - $|3| \in \mathbb{N}$
  - typechecker should reject $\mathsf{false} \in \mathbb{N}$
    - even though it is true!
    - in fact $\mathsf{false} \ \oplus \ |3| \ = \ |3|$

Problems with Translation

- Another example:

$$
\begin{aligned}
[\![(\lambda\, x.\ x)\ (1,\mathsf{T})]\!] \quad = \quad & (\boldsymbol{\lambda} x \in |num| \times |bool|.\ x) \\
& \diamond\, (((|,|\ \ |num|\ \ |bool|) \diamond |1|) \\
& \quad \diamond\, |\mathsf{T}|)
\end{aligned}
$$

- Would prefer:

$$[\![(\lambda\, x.\ x)\ (1,\mathsf{T})]\!] \quad = \quad \hat{\mathsf{I}}\ \langle |1|, \mathsf{true} \rangle$$

- Achievable by logical simplification if:

$$|\mathsf{T}| \quad = \quad \mathsf{true}$$

$$x \in X \wedge y \in Y \Rightarrow (((|,|\ \ X\ \ Y) \diamond x) \diamond y) \quad = \quad \langle x, y \rangle$$

$$y \in X \Rightarrow (\boldsymbol{\lambda} x \in X.\ x) \diamond y \quad = \quad \hat{\mathsf{I}}\ y$$

- Must override HOL definitions of T, pairing (,) etc.

Theories versus Theorems

- HOL theories can't be encoded as theorems

  Definition: $\forall x:\alpha.\ \mathsf{f}\ x \ = \ x$

  Theorems: $\vdash \ \mathsf{f}\ 0 \ = \ 0$
  $\vdash \ \forall x:\alpha.\tilde{}\ \mathsf{f}(\mathsf{f}\tilde{}x) \ = \ x$

  is not equivalent to:

  $$\forall f.\ (\forall x:\alpha.\ f\ x = x) \ \Rightarrow \ (f\ 0 = 0) \ \wedge \ (\forall x:\alpha.\ f(f\ x) = x)$$

  because variable f is used at different types

- With set theory:
  - theories can be encoded as theorems
  - 'theory interpretation' = specialisation
  - theories abbreviated with definitions

Theories as Theorems

- From the previous transparency:

  `Definition:` $\forall x:\alpha.\ \mathsf{f}\ x = x$

  `Theorems:` $\vdash\ \mathsf{f}\ 0 = 0$
  $\vdash\ \forall x:\alpha.\tilde{}\ \mathsf{f}(\mathsf{f}\tilde{}\, x) = x$

- Translated to set theory:

  `Definition:` $\forall\alpha\ x.\ x{\in}\alpha\ \Rightarrow\ (|\mathsf{f}|\ \alpha)\diamond x = x$

  `Theorems:` $\vdash\ |0|{\in}|num|\ \Rightarrow\ (|\mathsf{f}|\ |num|)\diamond|0| = 0$
  $\vdash\ \forall\alpha\ x.\ x{\in}\alpha\ \Rightarrow\ (|\mathsf{f}|\ \alpha)\diamond((|\mathsf{f}|\ \alpha)\diamond x) = x$

- As a single theorem:

$$
\begin{aligned}
\vdash\ \forall f:V\to V.\ & (\forall\alpha\ x.\ x{\in}\alpha\ \Rightarrow\ (f\ \alpha)\diamond x = x) \\
& \Rightarrow \\
& (|0|{\in}|num|\ \Rightarrow\ (f\ |num|)\diamond|0| = |0|\,) \\
& \wedge \\
& \forall\alpha\ x.\ x{\in}\alpha\ \Rightarrow\ (f\ \alpha)\diamond((f\ \alpha)\diamond x) = x
\end{aligned}
$$

Set Theory Or Higher Order Logic?

- Answer: BOTH
- Set theory is a more flexible foundation
- Types improve specification
  - type system should be customisable
- Proposed solution:
  - start with higher-order set theory
  - support type theoretic notations on top
- Research questions:
  - is this general scheme good
  - can types-as-sets be made practical
    - i.e. as efficient as native type theories
  - are 'soft types' really useful