Set Theory, Higher Order Logic Or Both?

- Why proof assistants use type theory
- Arguments for set theory
- Is the best of both worlds possible?
- A proposal
- Discussion of design issues
- Conclusions and the future

Type Theory is Popular

- Automath uses de Bruijn's own logic
  - anticipated much recent work
- Coq and LEGO support versions of the Calculus of Constructions
- HOL, Isabelle/HOL, TPS and Lambda support classical higher order logics with simple types
- IMPS supports simple types with
  - non-denoting terms
  - a theory interpretation mechanism
- PVS and Veritas support classical higher order logics with dependent types
- ALF and Nuprl support 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
- Advanced type theories are more general
  - e.g. Calculus of Constructions and Martin Löf type theory
  - support better 'structuring in the large'

Attractions of Set Theory

- More standard
  - taught in school and university
- Underlies popular specification methods
  - Z, VDM, TLA+ ...
- Well understood axiomatisations (ZFC etc)
  - stable compared with type theory
  - lots of metatheory
- More expressive than simple type theory
  - construction of D_∞ by Sten Agerholm
- Can be effectively mechanised
  - Isabelle/ZF, Mizar, EVES ...

The Best of Both Worlds?

- Can one have a single system combining
  - power and simplicity of set theory
  - types and functions as in higher order logic
- Common idea:
  - type system 'layered' over set theory
    - typechecking is theorem proving
    - guided by typechecking algorithms
  - λ-notation interpreted set-theoretically
    - shallow embedding
    - derived laws (e.g. β-conversion)
- Types as sets means types are 'first class'
  - quantified, passed as parameters etc.

A Concrete Proposal

- Start with higher order logic
  - simple type theory
  - e.g. HOL logic, Isabelle/HOL or PVS logic
- Add set theory
  - axiomatise a type V using ZFC axioms
- Embed 'specification logics' into set theory
  - typechecking derived – not 'hardwired'
- Case studies
  - Simply typed higher order logic
    - proof of concept
  - add dependent subtypes and record types
    - to show 'soft' types are flexible
  - add Σ and Π types
    - to show can get beyond simple types

Background to Proposal

- Experiments with HOL and set theory:
  - various lash-ups by me (Mike Gordon)
    - set theory defined inside HOL logic
    - transfer principles link types with sets
  - construction of D_∞ by Sten Agerholm
- Larry Paulson's work on Isabelle/ZF
  - demonstrates that set theory is practical
  - many *tour de forces* of proof (e.g. `Vrec`)
  - basis for comparison of first and higher order axiomatisations (Sten Agerholm)
- Francisco 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)

Design Issues

- Version of set theory
  - first order versus higher order
- Logical type of polymorphic constants
  - set variables instead of type variables
    - uniform, but verbose
  - 'true polymorphism' via operators on sets
    - ad hoc, but optimised
- Theories as theorems
  - conceptually clean
  - maybe impractical to implement

First Order versus Higher Order

formulation of axioms

- First Order Axiom of Replacement

$$\forall \mathtt{s}.\ (\forall \mathtt{x}\in\mathtt{s}.\ \forall \mathtt{y}\ \mathtt{z}.\ \phi(\mathtt{x},\mathtt{y}) \wedge \phi(\mathtt{x},\mathtt{z}) \Rightarrow \mathtt{y} = \mathtt{z}) \Rightarrow \exists \mathtt{t}.\ \forall \mathtt{y}.\ \mathtt{y} \in \mathtt{t} \equiv \exists \mathtt{x}\in\mathtt{s}.\ \phi(\mathtt{x},\mathtt{y})$$

  - an axiom schema
  - $\phi(x, y)$ ranges over formulae

- Higher Order Axiom of Replacement

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

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

Higher Order ZF Axioms

Extensionality

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

Empty set

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

Union

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

Power sets

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

Separation

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

Foundation

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

Replacement

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

Infinity

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

First Order versus Higher Order

construction of datatypes

- 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 versus Higher Order

construction of natural numbers

- Define by recursion in logic

$$\mathsf{numV}\ 0 \ = \ \varnothing$$
$$\mathsf{numV}(\mathsf{SUC}\ n) \ = \ (\mathsf{numV}\ n) \ \cup \ \{\mathsf{numV}\ n\}$$

- $\mathsf{numV} : \mathit{num} \to V$ is injective
- Set-theoretic numbers $\mathbb{N}$ are range of numV
- $\mathsf{Vnum} : V \to \mathit{num}$ is inverse of numV
- Define operations on sets via logic:

$$x \oplus y = \mathsf{numV}((\mathsf{Vnum}\ x) + (\mathsf{Vnum}\ y))$$
$$|+| = \{\langle\langle x, y\rangle, z\rangle \in (\mathbb{N} \times \mathbb{N}) \times \mathbb{N} \ \mid \ z = x \oplus y\}$$

- No need for `wfrec`
  - recursion done 'outside' set theory

First Order versus Higher Order

construction of lists

- Define $\langle x, y\rangle = \{\{x\}, \{x, y\}\}$
- Show

$$[x_1;\ x_2;\ \ldots;\ x_n] \mapsto \langle x_1, \langle x_2, \ldots, \langle x_n, \emptyset\rangle \ldots\rangle\rangle$$

  encodes HOL lists of sets as sets

- No need for `Vrec`

First Order versus Higher Order

powerful '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
  - $f : num \rightarrow V$ is a sequence of sets
    - $f\ \mathsf{T}$ rejected by typechecking
  - $\bigcup f$ is an infinite union
    - $\bigcup : (num \rightarrow V) \rightarrow V$
- 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
- Maybe set theory in Martin Löf Type Theory or the Calculus of Constructions even better!

Functions as Sets

- 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:

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

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

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

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:

$$\mathsf{true} \in \mathsf{Bool}, \quad \mathsf{false} \in \mathsf{Bool}$$

$$|\wedge| \in \mathsf{Bool} \times \mathsf{Bool} \twoheadrightarrow \mathsf{Bool}$$

$$|+| \in \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$$

- General typechecking theorems:

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

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

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

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

Typechecking is Syntactic

- Suppose

$$\mathsf{false} = \varnothing$$

$$\mathbb{N} = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\},\ \ldots\ \}$$

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 $\mathsf{false}\ \oplus\ |3|\ =\ |3|$

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 $\diamond$
  - many examples from Isabelle/ZF
  - can parameters be *uniformly* eliminated?

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 $[\![\sigma]\!]$

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

Tranlation of HOL Terms to Sets

- Term constant c translates to term $|\mathsf{c}|$
  - e.g. $|\mathsf{F}| \;=\; |0| \;=\; \varnothing$
- Term t recursively translates to term $[\![t]\!]$

$$
\begin{array}{lcl}
[\![x:\sigma]\!] & = & x : V \\
[\![\mathsf{c}:\sigma[\sigma_1,\ldots,\sigma_n]\,]\!] & = & |\mathsf{c}|\ [\![\sigma_1]\!]\ \ldots\ [\![\sigma_n]\!] \\
[\![\lambda\, x:\sigma.\ t]\!] & = & \boldsymbol{\lambda} x \in [\![\sigma]\!].\ [\![t]\!] \\
[\![t_1\ \ t_2]\!] & = & [\![t_1]\!] \diamond [\![t_2]\!]
\end{array}
$$

- Examples:

$$
\begin{array}{lcl}
[\![\lambda\, x:\alpha.\ x]\!] & = & \boldsymbol{\lambda} x \in \alpha.\ x \\
[\![(\lambda\, x.\ x)\ 1]\!] & = & (\boldsymbol{\lambda} x \in |num|.\ x) \diamond |1|
\end{array}
$$

Problems with Translation

- Another example:

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

- Would prefer:

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

- Achievable by logical simplification if:

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

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

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

- Must override HOL definitions of T, pairing (,) etc. to achieve this
  - needs more thought

Theories versus Theorems

- HOL theories can't be encoded as theorems

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

  $\texttt{Theorems:}\ \vdash\ \mathsf{f}\ 0\ =\ 0$

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

  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: $\vdash\ \mathsf{f}\ 0 = 0$

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

- 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:

$$
\begin{array}{l}
\vdash\ \forall f : V \rightarrow V.\\
\qquad (\forall \alpha\ x.\ x{\in}\alpha\ \Rightarrow\ (f\ \alpha){\diamond}x = x)\\
\qquad \Rightarrow\\
\qquad (|0|{\in}|num|\ \Rightarrow\ (f\ |num|){\diamond}|0| = |0|)\\
\qquad \wedge\\
\qquad \forall \alpha\ x.\ x{\in}\alpha\ \Rightarrow\ (f\ \alpha){\diamond}((f\ \alpha){\diamond}x) = x
\end{array}
$$

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

THE END