Levels of abstraction when defining type theory in type theory

Ambrus Kaposi
Eötvös Loránd University, Budapest

Workshop on type theory in type theory,
Gödel 90 conference
Nürtingen near Tübingen, 6 July 2021
What is the syntax of type theory?
What is the syntax of type theory?

(1) A term is a string:

"(λx.x+x) 3"
"(λx.x+x) 3a"
"((λx.x+x) 3)"
"(λx.x+x 3)"
"(λx.x+y) 3"
"(λx.x+y) 3"
"(λx.x+x) true"
What is the syntax of type theory?

(2) A term is a list of lexical elements:

"(\lambda x.x+x) 3"

"((\lambda x.x+x) 3)"

"(\lambda x.x+y) 3"
What is the syntax of type theory?

(3) A term is a tree (AST):

"(\lambda x. x+x) 3"

"(\lambda x. x+x) 3"

"6"

"(\lambda x. x + x) 3"

"((\lambda x. x+x) 3)"

"(\lambda y. y+y) 3"

"(\lambda x. x+y) 3"

"(\lambda x. x+x) true"
What is the syntax of type theory?

(4) A term is a well-scoped syntax tree (ABT):

"(\lambda x. x+x) 3"
"((\lambda x. x+x) 3)"
"(\lambda x. x + x) 3"
"6"
"(\lambda y. y+y) 3"
"(\lambda x. x+y) 3"
"(\lambda x. x+x) true"
What is the syntax of type theory?

(5) A term is a well-typed syntax tree (intrinsic):

"(\lambda x. x+x) 3"

"((\lambda x. x+x) 3)"

"(\lambda x. x + x) 3"

"6"

"(\lambda y. y+y) 3"

"(\lambda x. x+x) true"
What is the syntax of type theory?

(6) A term is a well-typed syntax tree quotiented by conversion (algebraic, equational theory, model-theoretic):

```
"(λx.x+x) 3"
"((λx.x+x) 3)"
"(λx.x + x) 3"
"6"
"(λy.y+y) 3"
```
Going abstract

(1) string
    \downarrow\text{lexical analysis}

(2) list of lexical elements
    \downarrow\text{parsing}

(3) syntax tree
    \downarrow\text{scope checking}

(4) well scoped syntax tree
    \downarrow\text{type checking}

(5) well typed syntax

(6) well typed syntax quotiented
Going abstract: errors

(1) string → invalid lexical element
      \lexical analysis

(2) list of lexical elements → bad num of params
      \parsing

(3) syntax tree → variable not in scope
      \scope checking

(4) well scoped syntax tree → non-matching types
      \type checking

(5) well typed syntax

(6) well typed syntax quotiented
Going abstract: quotienting

1. string  
   ↓ lexical analysis

2. list of lexical elements  
   ↓ parsing

3. syntax tree  
   ↓ scope checking

4. well scoped syntax tree  
   ↓ type checking

5. well typed syntax

6. well typed syntax quotiented
Going concrete (i)

(1) string
  add spaces \( \uparrow \) lexical analysis

(2) list of lexical elements
  add brackets \( \uparrow \) parsing

(3) syntax tree
  pick var names \( \uparrow \) scope checking

(4) well scoped syntax tree
  \( \uparrow \) type checking

(5) well typed syntax
  normalise \( \uparrow \)

(6) well typed syntax quotiented
Non-theorems (and another level)

(1) string
  ↗   ↘
(2) list of lexical elements
  ↗   ↘
(3) syntax tree
  ↗   ↘
(4) well scoped syntax tree
  ↗   ↘
(5) well typed syntax
  ↗   ↘
(6) well typed syntax quotiented
  ↗   ↘
(7) higher order abstract syntax

\(\alpha\)-renaming preserves
 MATCHING BRACKETS
\(\alpha\)-renaming preserves
 TYPING
CONVERSION PRESERVES
 TYPING
NORMALISATION IS SOUND
EVERYTHING IS STABLE UNDER SUBSTITUTION
What do we need to define the syntax?

1. string
2. list of lexical elements
3. syntax tree
4. well scoped syntax tree
5. well typed syntax
6. well typed syntax quotiented
7. higher order abstract syntax

strings
lists
inductive types (ITs)
indexed ITs
inductive-inductive types (IITs)
quotient IITs (QIITs)
QIITs with bindings
T.T. in T.T. at levels (1)–(4)

- **String**
- **List** \{(), $, \lambda, x, y, z, \ldots\}
- **tree**: inductive type given by the BNF grammar
  
  \[(\text{Abel–Öhman–Vezzosi POPL 2018})\]

  \[
v ::= x \mid y \mid z \mid \ldots \]

  \[
t ::= v \mid \lambda v.t \mid t \$ t\]

- **well-scoped tree**: indexed inductive type

  \[
  \text{Tm} : \mathbb{N} \rightarrow \text{Set} \\
  \text{var} : (i : \mathbb{N}) \rightarrow i < n \rightarrow \text{Tm} n \\
  \text{lam} : \text{Tm} (1 + n) \rightarrow \text{Tm} n \\
  \$ : \text{Tm} n \rightarrow \text{Tm} n \rightarrow \text{Tm} n
  \]

well-typed tree: inductive-inductive type\(^1\)
(Chapman: Type theory should eat itself 2009)

\[
\begin{align*}
\text{Con} & : \text{Set} \\
\text{Ty} & : \text{Con} \to \text{Set} \\
\cdot & : \text{Con} \\
\_ \land \_ & : (\Gamma : \text{Con}) \to \text{Ty} \Gamma \to \text{Con} \\
\ldots
\end{align*}
\]

\[
\begin{align*}
\text{Tm} & : (\Gamma : \text{Con}) \to \text{Ty} \Gamma \to \text{Set} \\
\_ \Rightarrow \_ & : \text{Ty} \Gamma \to \text{Ty} \Gamma \to \text{Ty} \Gamma \\
\text{lam} & : \text{Tm} (\Gamma, A) B \to \text{Tm} \Gamma (A \Rightarrow B) \\
\_/ \_ & : \text{Tm} \Gamma (A \Rightarrow B) \to \text{Tm} \Gamma A \to \text{Tm} \Gamma B
\end{align*}
\]

\(^1\)the first \(B\) in the type of lam needs to be weakened, also in the next slide
**T.T. in T.T. at level (6)**

- well-typed tree quotiented: QIIT
  
  (Dybjer CwF 1996, Altenkirch–Kaposi POPL 2016)

\[
\begin{align*}
\text{Con} & : \text{Set} \\
\text{Ty} & : \text{Con} \rightarrow \text{Set} \\
\cdot & : \text{Con} \\
-,- & : (\Gamma : \text{Con}) \rightarrow \text{Ty} \Gamma \rightarrow \text{Con} \\
\text{Sub} & : \text{Con} \rightarrow \text{Con} \rightarrow \text{Set} \\
\ldots
\end{align*}
\]

\[
\begin{align*}
\text{Tm} & : (\Gamma : \text{Con}) \rightarrow \text{Ty} \Gamma \rightarrow \text{Set} \\
-[-] & : \text{Ty} \Gamma \rightarrow \text{Sub} \Delta \Gamma \rightarrow \text{Ty} \Delta \\
-[-] & : \text{Tm} \Gamma A \rightarrow (\sigma : \text{Sub} \Delta \Gamma) \rightarrow \text{Tm} \Delta (A[\sigma]) \\
\text{lam} & : \text{Tm} (\Gamma, A) B \rightarrow \text{Tm} \Gamma (A \Rightarrow B) \\
- \$- & : \text{Tm} \Gamma (A \Rightarrow B) \rightarrow \text{Tm} \Gamma A \rightarrow \text{Tm} \Gamma B \\
\beta & : \text{lam} t \$ u = t[\text{id}, u]
\end{align*}
\]
T.T. in T.T. at level (7)

- higher order abstract syntax

\[
\begin{align*}
\text{Ty} & : \text{Set} \\
\text{Tm} & : \text{Ty} \rightarrow \overline{\text{Set}} \\
\Rightarrow & : \text{Ty} \rightarrow \text{Ty} \rightarrow \text{Ty} \\
\text{lam} & : (\text{Tm } A \Rightarrow \text{Tm } B) \rightarrow \text{Tm } (A \Rightarrow B) \\
\$ & : \text{Tm } (A \Rightarrow B) \rightarrow \text{Tm } A \rightarrow \text{Tm } B \\
\beta & : \text{lam } t \$ u = t^- u
\end{align*}
\]
Going concrete (ii)

(1) string strings

(2) list of lexical elements lists
    “Gödel numbering”

(3) syntax tree inductive types (ITs)
    indexed W types → W types

(4) well scoped syntax tree indexed ITs
    “typing” predicates

(5) well typed syntax IITs
    setoid model

(6) well typed syntax quotiented QIITs
    presheaf model

(7) higher order abstract syntax QIITs with bindings
What has been done at levels (6)/(7)?

- Normalisation (canonicity, decidability of equality).
  - statement: \( \text{Tm} \Gamma A \cong \text{Nf} \Gamma A \)
  - normalisation by evaluation, logical predicates (Altenkirch–Kaposi 2016, Coquand 2019)
  - big-step normalisation (Altenkirch–Geniet TYPES 2019)
- Bidirectional type checking: only need to check equality of level (6) terms.
- Elaboration. Metavariabes can be handled by a modality, they live at level (6). (e.g. Kovács ICFP 2020)
- Call by value, call by name (see Levy’s call by push value).
- Closure conversion (Kovács TYPES 2018).

Some of the above at level (7): (Bocquet–Kaposi–Sattler 2021)
What is hard at levels (6)/(7)?

- Compilation to lower level language: the low level language needs a matching equational theory.
- Level (7) cannot formalise calculi where some operations are not stable under substitution (e.g. Martin-Löf’s first presentation of t.t.)
- Level (6) formalisation is still hard because QIITs are not supported (except Cubical Agda).
- Level (7) needs modalities when moving between models, e.g. multi-modal type theory (Gratzer–Kavvos–Nuyts–Birkedal 2021).
Why not normal forms instead of quotienting?

1. not easier to formalise
2. We want to write non-normal proofs and programs
Questions

- Is there a presentation of normal forms of t.t. that does not refer to the equational theory?
- What features of programming languages cannot be described at the algebraic level? E.g. small step semantics.
- Can we reproduce (Abel–Öhman–Vezzosi POPL 2018) at level (6) without UIP?
- What is the best calculus for level (7)? Binding and names built-in, maybe multi-modal t.t.?