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+x 3"`

`"6"`

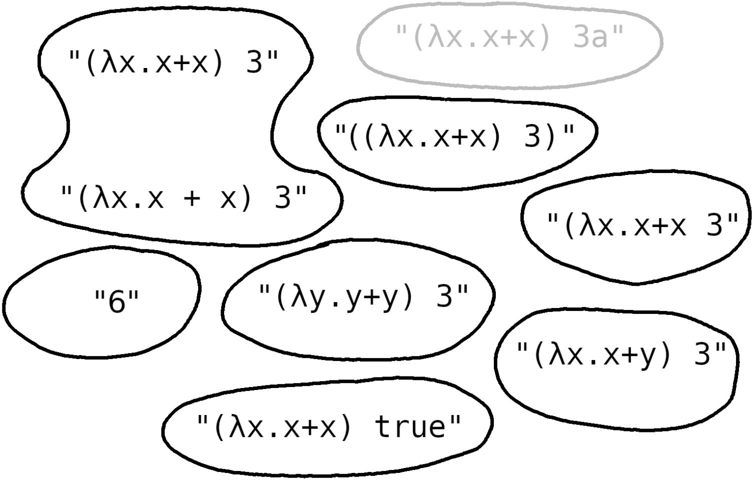
`"(λy.y+y) 3"`

`"(λx.x+y) 3"`

`"(λx.x+x) true"`

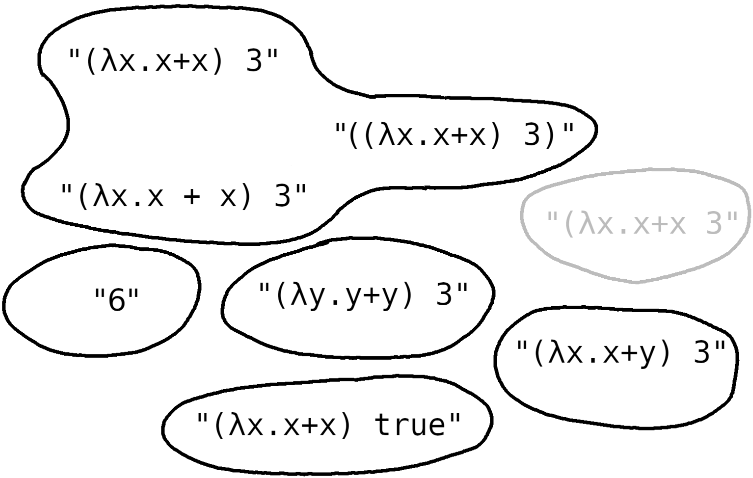
What is the syntax of type theory?

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



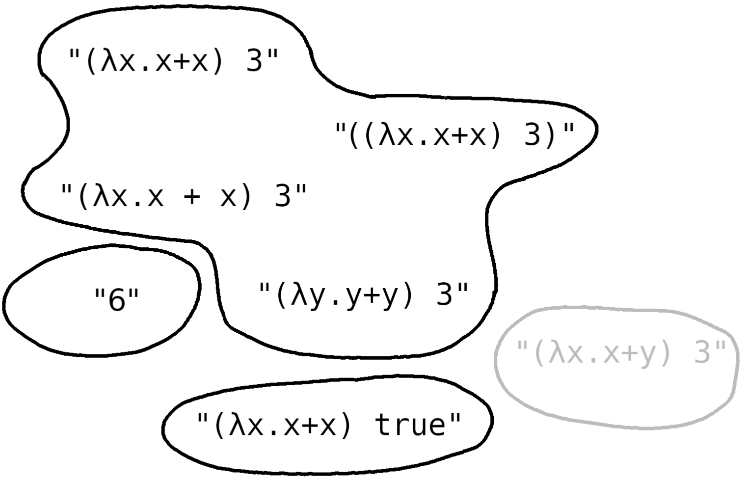
What is the syntax of type theory?

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



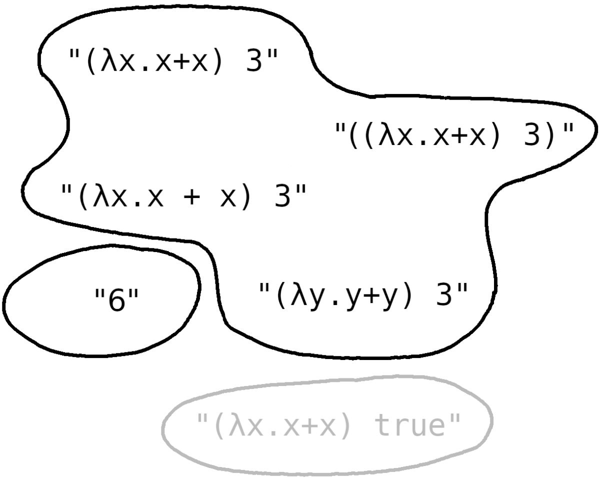
What is the syntax of type theory?

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



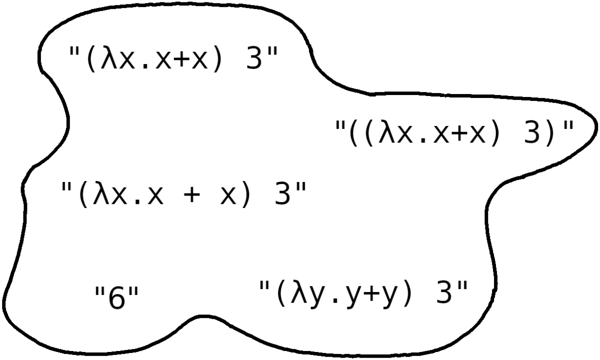
What is the syntax of type theory?

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



What is the syntax of type theory?

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



Going abstract

(1) string

$\rightsquigarrow$ lexical analysis

(2) list of lexical elements

$\rightsquigarrow$ parsing

(3) syntax tree

$\rightsquigarrow$ scope checking

(4) well scoped syntax tree

$\rightsquigarrow$ type checking

(5) well typed syntax

$\rightsquigarrow$

(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

$\rightsquigarrow$ lexical analysis

(2) list of lexical elements `"1+1" = "1 + 1"`

$\rightsquigarrow$ parsing

(3) syntax tree $[1,+,1] = [(,1,+,1,)]$

$\rightsquigarrow$ scope checking

(4) well scoped syntax tree $\lambda x.x = \lambda y.y$

$\rightsquigarrow$ type checking

(5) well typed syntax

$\rightsquigarrow$

(6) well typed syntax quotiented $(\lambda x.x + x)\ 3 = 3 + 3$

Going concrete (i)

(1) string

add spaces ↑ ↓ lexical analysis

(2) list of lexical elements

add brackets ↑ ↓ parsing

(3) syntax tree

pick var names ↑ ↓ scope checking

(4) well scoped syntax tree

↑ ↓ type checking

(5) well typed syntax

normalise ↑ ↓

(6) well typed syntax quotiented

Non-theorems (and another level)

(1) string

(2) list of lexical elements

(3) syntax tree — α-renaming preserves matching brackets

(4) well scoped syntax tree — α-renaming preserves typing

(5) well typed syntax — conversion preserves typing

(6) well typed syntax quotiented — normalisation is sound

(7) higher order abstract syntax — everything is stable under substitution

What do we need to define the syntax?

(1) string	strings
(2) list of lexical elements	lists
(3) syntax tree	inductive types (ITs)
(4) well scoped syntax tree	indexed ITs
(5) well typed syntax	inductive-inductive types (IITs)
(6) well typed syntax quotiented	quotient IITs (QIITs)
(7) higher order abstract syntax	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
  (Abel–Öhman–Vezzosi POPL 2018)

$$
\begin{aligned}
v &::= x \mid y \mid z \mid \ldots \\
t &::= \; v \mid \lambda v.t \mid t \,\$\, t
\end{aligned}
$$

- well-scoped tree: indexed inductive type

$$
\begin{aligned}
&\mathsf{Tm} && : \mathbb{N} \to \mathsf{Set} \\
&\mathsf{var} && : (i : \mathbb{N}) \to i < n \to \mathsf{Tm}\, n \\
&\mathsf{lam} && : \mathsf{Tm}\,(1 + n) \to \mathsf{Tm}\, n \\
&- \,\$\, - && : \mathsf{Tm}\, n \to \mathsf{Tm}\, n \to \mathsf{Tm}\, n
\end{aligned}
$$

T.T. in T.T. at level (5)

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

$$
\begin{aligned}
\mathsf{Con} &: \mathsf{Set} \\
\mathsf{Ty} &: \mathsf{Con} \to \mathsf{Set} \\
\cdot &: \mathsf{Con} \\
-,- &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con} \\
\ldots & \\
\mathsf{Tm} &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \\
- \Rightarrow - &: \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \\
\mathsf{lam} &: \mathsf{Tm}\,(\Gamma, A)\,B \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \\
- \$ - &: \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,B
\end{aligned}
$$

[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)

$$
\begin{array}{ll}
\mathsf{Con} & : \mathsf{Set} \\
\mathsf{Ty} & : \mathsf{Con} \to \mathsf{Set} \\
\cdot & : \mathsf{Con} \\
-,- & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con} \\
\mathsf{Sub} & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
\ldots & \\
\mathsf{Tm} & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \\
-[-] & : \mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta \\
-[-] & : \mathsf{Tm}\,\Gamma\,A \to (\sigma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\sigma]) \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma, A)\,B \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \\
-\,\$\,- & : \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,B \\
\beta & : \mathsf{lam}\,t\,\$\,u = t[\mathsf{id}, u]
\end{array}
$$

T.T. in T.T. at level (7)

- higher order abstract syntax
  (Hofmann 1999, Awodey's natural models 2014, Bocquet–Kaposi–Sattler 2021)

$$
\begin{aligned}
\mathsf{Ty} &: \mathsf{Set} \\
\mathsf{Tm} &: \mathsf{Ty} \to \overline{\mathsf{Set}} \\
- \Rightarrow - &: \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{lam} &: (\mathsf{Tm}\,A \mathbin{\overline{\Rightarrow}} \mathsf{Tm}\,B) \to \mathsf{Tm}\,(A \Rightarrow B) \\
- \,\$\, - &: \mathsf{Tm}\,(A \Rightarrow B) \to \mathsf{Tm}\,A \to \mathsf{Tm}\,B \\
\beta &: \mathsf{lam}\,t \,\$\, u = t \;\overline{}\; u
\end{aligned}
$$

Going concrete (ii)

(1) string	strings
	↑
(2) list of lexical elements	lists
	"Gödel numbering" ↑
(3) syntax tree	inductive types (ITs)
	indexed W types $\to$ 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: $\mathsf{Tm}\,\Gamma\,A \cong \mathsf{Nf}\,\Gamma\,A$
  - normalisation by evaluation, logical predicates (Altenkirch–Kaposi 2016, Coquand 2019)
  - big-step normalisation (Altenkirch–Geniet TYPES 2019)
- Parametricity (Altenkirch–Kaposi 2016, Moeneclaey LICS 2021).
- Bidirectional type checking: only need to check equality of level (6) terms.
- Elaboration. Metavariables can be handled by a modality, they live at level (6). (e.g. Kovács ICFP 2020)
- Conservativity proofs (Hofmann 1995, Capriotti 2017).
- 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) 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

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.?