Type Theory in Type Theory using Quotient Inductive Types

Thorsten Altenkirch, Ambrus Kaposi

University of Nottingham

POPL, St Petersburg, Florida

20 January 2016

Goal

- To represent the syntax of Type Theory inside Type Theory
- Why?
  - Study the metatheory in a nice language
  - Template type theory

Expressing the judgements of Type Theory

$$\Gamma \vdash t : A$$

will be formalised as

$$t' : \mathrm{Tm}\ \Gamma\ A$$

(We are not interested in untyped terms)

Simple type theory in Agda (i)

$$
\begin{array}{lll}
\textbf{data}\ \mathsf{Ty} & : & \mathsf{Set}\ \textbf{where} \\
\quad \iota & : & \mathsf{Ty} \\
\quad \_{\Rightarrow}\_ & : & \mathsf{Ty} \ \to\ \mathsf{Ty} \ \to\ \mathsf{Ty} \\
\textbf{data}\ \mathsf{Con} & : & \mathsf{Set}\ \textbf{where} \\
\quad \bullet & : & \mathsf{Con} \\
\quad \_{,}\_ & : & \mathsf{Con} \ \to\ \mathsf{Ty} \ \to\ \mathsf{Con} \\
\textbf{data}\ \mathsf{Var} & : & \mathsf{Con} \ \to\ \mathsf{Ty} \ \to\ \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{zero} & : & \mathsf{Var}\ (\Gamma\ , \mathsf{A})\ \mathsf{A} \\
\quad \mathsf{suc} & : & \mathsf{Var}\ \Gamma\ \mathsf{A} \ \to\ \mathsf{Var}\ (\Gamma\ , \mathsf{B})\ \mathsf{A} \\
\textbf{data}\ \mathsf{Tm} & : & \mathsf{Con} \ \to\ \mathsf{Ty} \ \to\ \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{var} & : & \mathsf{Var}\ \Gamma\ \mathsf{A} \ \to\ \mathsf{Tm}\ \Gamma\ \mathsf{A} \\
\quad \mathsf{app} & : & \mathsf{Tm}\ \Gamma\ (\mathsf{A} \Rightarrow \mathsf{B}) \ \to\ \mathsf{Tm}\ \Gamma\ \mathsf{A} \ \to\ \mathsf{Tm}\ \Gamma\ \mathsf{B} \\
\quad \mathsf{lam} & : & \mathsf{Tm}\ (\Gamma\ , \mathsf{A})\ \mathsf{B} \ \to\ \mathsf{Tm}\ \Gamma\ (\mathsf{A} \Rightarrow \mathsf{B})
\end{array}
$$

Simple type theory in Agda (ii)

- In addition, we need substitutions:

$$
\begin{aligned}
\mathsf{Tms} &: \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
\_[\_] &: \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tms}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta\,A
\end{aligned}
$$

- Now we can define a conversion relation:

$$\_\tilde{}\_ : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Set}$$

eg. $\mathsf{app}\,(\mathsf{lam}\,t)\,u \sim t\,[\,\mathsf{id}\,,\,u\,]$

- The intended syntax is a quotient:

$$\mathsf{Tm}\,\Gamma\,A \;/\; \sim$$

The syntax of Dependent Type Theory (i)

- Types depend on contexts
- Substitutions are mentioned in the application rule:
  $\mathsf{app} : \mathsf{Tm}\ \Gamma\ (\Pi\ A\ B)\ (a : \mathsf{Tm}\ \Gamma\ A) \rightarrow \mathsf{Tm}\ \Gamma\ (B\ [\ a\ ])$
- We need an inductive-inductive definition:

$$
\begin{array}{lcl}
\mathbf{data}\ \mathsf{Con} & : & \mathsf{Set} \\
\mathbf{data}\ \mathsf{Ty} & : & \mathsf{Con} \rightarrow \mathsf{Set} \\
\mathbf{data}\ \mathsf{Tms} & : & \mathsf{Con} \rightarrow \mathsf{Con} \rightarrow \mathsf{Set} \\
\mathbf{data}\ \mathsf{Tm} & : & (\Gamma : \mathsf{Con}) \rightarrow \mathsf{Ty}\ \Gamma \rightarrow \mathsf{Set}
\end{array}
$$

The syntax of Dependent Type Theory (ii)

- In addition, there is a coercion rule for terms:

$$\frac{\Gamma \vdash A \sim B \qquad \Gamma \vdash t : A}{\Gamma \vdash t : B}$$

- This forces us to define conversion relations mutually:

$$
\begin{array}{lcl}
\textbf{data}\ \mathsf{Con} & : & \mathsf{Set} \\
\textbf{data}\ \mathsf{Ty} & : & \mathsf{Con} \ \rightarrow\ \mathsf{Set} \\
\textbf{data}\ \mathsf{Tms} & : & \mathsf{Con} \ \rightarrow\ \mathsf{Con} \ \rightarrow\ \mathsf{Set} \\
\textbf{data}\ \mathsf{Tm} & : & (\Gamma\ :\ \mathsf{Con}) \ \rightarrow\ \mathsf{Ty}\ \Gamma \ \rightarrow\ \mathsf{Set} \\
\textbf{data}\ \_\tilde{}\mathsf{Con}\_ & : & \mathsf{Con} \ \rightarrow\ \mathsf{Con} \ \rightarrow\ \mathsf{Set} \\
\textbf{data}\ \_\tilde{}\mathsf{Ty}\_ & : & \mathsf{Ty}\ \Gamma \ \rightarrow\ \mathsf{Ty}\ \Gamma \ \rightarrow\ \mathsf{Set} \\
\textbf{data}\ \_\tilde{}\mathsf{Tms}\_ & : & \mathsf{Tms}\ \Delta\ \Gamma \ \rightarrow\ \mathsf{Tms}\ \Delta\ \Gamma \ \rightarrow\ \mathsf{Set} \\
\textbf{data}\ \_\tilde{}\mathsf{Tm}\_ & : & \mathsf{Tm}\ \Gamma\ \mathsf{A} \ \rightarrow\ \mathsf{Tm}\ \Gamma\ \mathsf{A} \ \rightarrow\ \mathsf{Set}
\end{array}
$$

Lots of boilerplate

- The _~X_ relations are equivalence relations
- Coercion rules
- Congruence rules
- We need to work with setoids

The identity type $\_ \equiv \_$

- Equality (the identity type) is an equivalence relation
- We can coerce between equal types
- Equality is a congruence
- What about the extra equalities (eg. β, η for Π)?

Higher inductive types

- An idea from homotopy type theory: constructors for equalities.
- Example:

$$\begin{aligned} &\textbf{data}\ \mathsf{I} && : \mathsf{Set}\ \textbf{where} \\ &\quad \mathsf{zero} && : \mathsf{I} \\ &\quad \mathsf{one} && : \mathsf{I} \\ &\quad \mathsf{segment} && : \mathsf{zero} \equiv \mathsf{one} \end{aligned}$$

Higher inductive types

- An idea from homotopy type theory: constructors for equalities.
- Example:

$$\begin{aligned}
&\textbf{data}\ \mathsf{I} && : \mathsf{Set}\ \textbf{where} \\
&\quad \mathsf{zero} && : \mathsf{I} \\
&\quad \mathsf{one} && : \mathsf{I} \\
&\quad \mathsf{segment} && : \mathsf{zero} \equiv \mathsf{one}
\end{aligned}$$

$$\begin{aligned}
\mathsf{RecI}\ :\ & (\mathsf{I}^{\mathsf{M}} : \mathsf{Set}) \\
& (\mathsf{zero}^{\mathsf{M}} : \mathsf{I}^{\mathsf{M}}) \\
& (\mathsf{one}^{\mathsf{M}} : \mathsf{I}^{\mathsf{M}}) \\
& (\mathsf{segment}^{\mathsf{M}} : \mathsf{zero}^{\mathsf{M}} \equiv \mathsf{one}^{\mathsf{M}}) \\
& \to \mathsf{I} \to \mathsf{I}^{\mathsf{M}}
\end{aligned}$$

Quotient inductive types (QITs)

- A higher inductive type which is truncated to an h-set.
- They are *not* the same as quotient types: equality constructors are defined at the same time
- QITs can be simulated in Agda

The syntax of Dependent Type Theory (iii)

- We defined the syntax of a basic Type Theory as a quotient inductive inductive type (with Π and an uninterpreted family of types U, El)
- We don't need to state the equivalence relation, coercion, congruence laws anymore
- We collect the arguments of the recursor into a record:

$$
\begin{array}{lll}
\textbf{record}\ \mathsf{Model} & : & \mathsf{Set}\ \textbf{where} \\
\quad \textbf{field}\ \mathsf{Con}^{\mathsf{M}} & : & \mathsf{Set} \\
\qquad\quad \mathsf{Ty}^{\mathsf{M}} & : & \mathsf{Con}^{\mathsf{M}} \to \mathsf{Set} \\
\qquad\quad \ldots & &
\end{array}
$$

- which is the type of algebras for the QIT
  = the type of models of Type Theory, close to CwF

Applications (i): standard model

- A sanity check
- Every syntactic construct is interpreted as the corresponding metatheoretic construction.

$$
\begin{array}{llcl}
\mathsf{Con}^{\mathsf{M}} & & = & \mathsf{Set} \\
\mathsf{Ty}^{\mathsf{M}} & [\![\Gamma]\!] & = & [\![\Gamma]\!] \to \mathsf{Set} \\
\Pi^{\mathsf{M}} & [\![A]\!]\ [\![B]\!]\ \gamma & = & (\mathsf{x} : [\![A]\!]\ \gamma) \to [\![B]\!]\ (\gamma , \mathsf{x}) \\
\mathsf{lam}^{\mathsf{M}} & [\![t]\!] \qquad \gamma & = & \lambda\, \mathsf{x} \to [\![t]\!]\ (\gamma , \mathsf{x}) \\
\ldots & & &
\end{array}
$$

Applications (ii): logical predicate interpretation

- An interpretation from the syntax into the syntax
- Bernardy-Jansson-Paterson: Parametricity and Dependent Types, 2012
- A type is interpreted as a logical predicate over that type
- A term is interpreted as a proof that it satisfies the predicate
- Automated derivation of free theorems

Applications (iii): presheaf model

- Given a category $\mathcal{C}$
- Contexts are presheaves over $\mathcal{C}$
- Types are families of presheaves, terms are sections
- Normalisation by evaluation (NBE):
  - A presheaf over the category of renamings
  - We can generalise NBE from Simple Type Theory to Type Theory (formalisation in progress)

Further work

- We internalized a very basic type theory, but this can be extended easily with universes and inductive types.
- We used axioms (quotient inductive types, functional extensionality) in our metatheory. This can be solved by using cubical type theory.
- If we work within HoTT, we can only eliminate into h-sets. Hence, the standard model doesn't work as described.

Template type theory

- Given a model of type theory, together with new constants in that model
- We can interpret code that uses the new constants inside the model
- The code can use all the conveniences such as implicit arguments, pattern matching etc.
- This way we can justify extensions of type theory:
  - guarded type theory
  - local state monad
  - parametricity
  - homotopy type theory