Type theory in type theory using quotient inductive types

Ambrus Kaposi
(joint work with Thorsten Altenkirch)
University of Nottingham

Theoretical CS Seminar, Birmingham
26 February 2016

Goal

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

Structure

1. Simple type theory
2. Dependent type theory
3. Standard model
4. Logical predicate interpretation
5. Presheaf models, normalisation by evaluation
6. The future

Expressing the judgements of type theory

$$\Gamma \vdash t : A$$

will be formalised as

$$t : \mathsf{Tm}\ \Gamma\ A$$

(We have a **typed** presentation, no preterms)

Simple type theory with preterms

$$\begin{aligned}
\mathsf{x} &::= \mathsf{zero} \;\mid\; \mathsf{suc}\;\mathsf{x} \\
\mathsf{t} &::= \mathsf{x} \;\mid\; \mathsf{lam}\;\mathsf{t} \;\mid\; \mathsf{app}\;\mathsf{t}\;\mathsf{t} \\
\mathsf{A} &::= \iota \;\mid\; \mathsf{A} \Rightarrow \mathsf{A} \\
\Gamma &::= \bullet \;\mid\; \Gamma\;,\;\mathsf{A}
\end{aligned}$$

We define the relations $\vdash_v$ and $\vdash$.

$$\frac{}{\Gamma, A \vdash_v \mathsf{zero} : A} \qquad \frac{\Gamma \vdash_v x : A}{\Gamma, B \vdash_v \mathsf{suc}\, x : A}$$

$$\frac{\Gamma \vdash_v x : A}{\Gamma \vdash x : A} \quad \frac{\Gamma, A \vdash t : B}{\Gamma \vdash \mathsf{lam}\, t : A \to B} \quad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{app}\, t\, u : B}$$

Simple type theory in idealised Agda (i)

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

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:

$$
\begin{aligned}
\mathsf{app} \ : \quad & \mathsf{Tm}\ \Gamma\ (\Pi\ \mathsf{A}\ \mathsf{B}) \ \to\ (\mathsf{a}\ :\ \mathsf{Tm}\ \Gamma\ \mathsf{A}) \\
\to\ \ & \mathsf{Tm}\ \Gamma\ (\mathsf{B}\ [\ \mathsf{a}\ ])
\end{aligned}
$$

- We need an inductive-inductive definition:

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Con} && :\ \mathsf{Set} \\
&\textbf{data}\ \mathsf{Ty} && :\ \mathsf{Con}\ \to\ \mathsf{Set} \\
&\textbf{data}\ \mathsf{Tms} && :\ \mathsf{Con}\ \to\ \mathsf{Con}\ \to\ \mathsf{Set} \\
&\textbf{data}\ \mathsf{Tm} && :\ (\Gamma\ :\ \mathsf{Con})\ \to\ \mathsf{Ty}\ \Gamma\ \to\ \mathsf{Set}
\end{aligned}
$$

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:

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

(1) Contexts:

$$\frac{}{\cdot \vdash}\ \text{C-empty} \qquad \frac{\Gamma \vdash \qquad \Gamma \vdash A : \mathsf{U}}{\Gamma.x : A \vdash}\ \text{C-ext}$$

(2) Terms:

$$\frac{\Gamma \vdash A : \mathsf{U}}{\Gamma.x : A \vdash x : A}\ \text{var} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : \mathsf{U}}{\Gamma.x : B \vdash t : A}\ \text{wk} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{U} : \mathsf{U}}\ \text{U-I}$$

$$\frac{\Gamma \vdash A : \mathsf{U} \qquad \Gamma.x : A \vdash B : \mathsf{U}}{\Gamma \vdash \Pi(x : A).B : \mathsf{U}}\ \Pi\text{-F} \qquad \frac{\Gamma.x : A \vdash t : B}{\Gamma \vdash \lambda x.t : \Pi(x : A).B}\ \Pi\text{-I} \qquad \frac{\Gamma \vdash f : \Pi(x : A).B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ a : B[x \mapsto a]}\ \Pi\text{-E}$$

$$\frac{\Gamma \sim \Delta \vdash \quad \Gamma \vdash A \sim B : \mathsf{U} \quad \Gamma \vdash t : A}{\Delta \vdash t : B}\ \text{t-coe}$$

(3) Conversion for contexts:

$$\frac{\Gamma \vdash}{\Gamma \sim \Gamma \vdash}\ \text{C-eq-refl} \qquad \frac{\Gamma \sim \Delta \vdash}{\Delta \sim \Gamma \vdash}\ \text{C-eq-sym} \qquad \frac{\Gamma \sim \Delta \vdash \quad \Delta \sim \Theta \vdash}{\Gamma \sim \Theta \vdash}\ \text{C-eq-trans}$$

$$\frac{\Gamma \sim \Delta \vdash \quad \Gamma \vdash A \sim B : \mathsf{U}}{\Gamma.x : A \sim \Delta.x : B \vdash}\ \text{C-ext-cong}$$

(4) Conversion for terms:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \sim t : A}\ \text{t-eq-refl} \qquad \frac{\Gamma \vdash u \sim v : A}{\Gamma \vdash v \sim u : A}\ \text{t-eq-sym} \qquad \frac{\Gamma \vdash u \sim v : A \quad \Gamma \vdash v \sim w : A}{\Gamma \vdash u \sim w : A}\ \text{t-eq-trans}$$

$$\frac{\Gamma \sim \Delta \vdash \quad \Gamma \vdash A \sim B : \mathsf{U} \quad \Gamma \vdash u \sim v : A}{\Delta \vdash u \sim v : B}\ \text{t-eq-coe} \qquad \frac{\Gamma \vdash A \sim A' : \mathsf{U} \qquad \Gamma.x : A \vdash B \sim B' : \mathsf{U}}{\Gamma \vdash \Pi(x : A).B \sim \Pi(x : A').B' : \mathsf{U}}\ \Pi\text{-F-cong}$$

$$\frac{\Gamma.x : A \vdash t \sim t' : B}{\Gamma \vdash \lambda x.t \sim \lambda x.t' : \Pi(x : A).B}\ \Pi\text{-I-cong} \qquad \frac{\Gamma \vdash f \sim f' : \Pi(x : A).B \qquad \Gamma \vdash a \sim a' : A}{\Gamma \vdash f\ a \sim f'\ a' : B[x \mapsto a]}\ \Pi\text{-E-cong}$$

$$\frac{\Gamma.x : A \vdash t : B}{\Gamma \vdash (\lambda x.t)\ a \sim t[x \mapsto a] : B[x \mapsto a]}\ \Pi\text{-}\beta \qquad \frac{\Gamma \vdash f : \Pi(x : A).B}{\Gamma \vdash f \sim (\lambda x.f\ x) : \Pi(x : A).B}\ \Pi\text{-}\eta$$

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:

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

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:

$$\begin{array}{rll} \textbf{record}\ \mathsf{Model} & : & \mathsf{Set}\ \textbf{where} \\ \textbf{field}\ \mathsf{Con}^{\mathsf{M}} & : & \mathsf{Set} \\ \mathsf{Ty}^{\mathsf{M}} & : & \mathsf{Con}^{\mathsf{M}} \rightarrow \mathsf{Set} \ldots \end{array}$$

- which is the type of algebras for the QIT
  = the type of models of type theory, close to CwF.
  Initiality is given by the recursor

$$
\begin{aligned}
\cdot \quad &: \mathsf{Con}\\
-,- \quad &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}\\[1em]
-[-] \quad &: \mathsf{Ty}\,\Delta \to \mathsf{Tms}\,\Gamma\,\Delta \to \mathsf{Ty}\,\Gamma\\
\mathsf{U} \quad &: \mathsf{Ty}\,\Gamma\\
\mathsf{El} \quad &: \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma\\
\Pi \quad &: (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma, A) \to \mathsf{Ty}\,\Gamma\\[1em]
\mathsf{id} \quad &: \mathsf{Tms}\,\Gamma\,\Gamma\\
- \circ - \quad &: \mathsf{Tms}\,\Theta\,\Delta \to \mathsf{Tms}\,\Gamma\,\Theta \to \mathsf{Tms}\,\Gamma\,\Delta\\
\epsilon \quad &: \mathsf{Tms}\,\Gamma\,\cdot\\
-,- \quad &: (\sigma : \mathsf{Tms}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,A[\sigma] \to \mathsf{Tms}\,\Gamma\,(\Delta, A)\\
\pi_1 \quad &: \mathsf{Tms}\,\Gamma\,(\Delta, A) \to \mathsf{Tms}\,\Gamma\,\Delta\\[1em]
-[-] \quad &: \mathsf{Tm}\,\Delta\,A \to (\sigma : \mathsf{Tms}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,A[\sigma]\\
\pi_2 \quad &: (\sigma : \mathsf{Tms}\,\Gamma\,(\Delta, A)) \to \mathsf{Tm}\,\Gamma\,A[\pi_1\,\sigma]\\
\mathsf{lam} \quad &: \mathsf{Tm}\,(\Gamma, A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)\\
\mathsf{app} \quad &: \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to \mathsf{Tm}\,(\Gamma, A)\,B
\end{aligned}
$$

$$
\begin{aligned}
[\mathsf{id}] \quad &: A[\mathsf{id}] \equiv A\\
[][] \quad &: A[\sigma][\nu] \equiv A[\sigma \circ \nu]\\
\mathsf{U}[] \quad &: \mathsf{U}[\sigma] \equiv \mathsf{U}\\
\mathsf{El}[] \quad &: (\mathsf{El}\,\hat{A})[\sigma] \equiv \mathsf{El}\,({}_{\mathsf{U}[]*}\hat{A}[\sigma])\\
\Pi[] \quad &: (\Pi\,A\,B)[\sigma] \equiv \Pi\,(A[\sigma])\,(B[\sigma^A])\\[1em]
\mathsf{id}\circ \quad &: \mathsf{id} \circ \sigma \equiv \sigma\\
\circ\mathsf{id} \quad &: \sigma \circ \mathsf{id} \equiv \sigma\\
\circ\circ \quad &: (\sigma \circ \nu) \circ \delta \equiv \sigma \circ (\nu \circ \delta)\\
\epsilon\eta \quad &: \{\sigma : \mathsf{Tms}\,\Gamma\,\cdot\} \to \sigma \equiv \epsilon\\
\pi_1\beta \quad &: \pi_1\,(\sigma, t) \equiv \sigma\\
\pi\eta \quad &: (\pi_1\,\sigma, \pi_2\,\sigma) \equiv \sigma\\
,\circ \quad &: (\sigma, t) \circ \nu \equiv (\sigma \circ \nu), ({}_{[][]*}t[\nu])\\[1em]
\pi_2\beta \quad &: \pi_2\,(\sigma, t) \equiv^{\pi_1\beta} t\\
\Pi\beta \quad &: \mathsf{app}\,(\mathsf{lam}\,t) \equiv t\\
\Pi\eta \quad &: \mathsf{lam}\,(\mathsf{app}\,t) \equiv t\\
\mathsf{lam}[] \quad &: (\mathsf{lam}\,t)[\sigma] \equiv^{\Pi[]} \mathsf{lam}\,(t[\sigma^A])
\end{aligned}
$$

Standard model

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

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

Logical predicate interpretation (i)

- Unary parametricity says that terms respect logical predicates. Example:

$$A : \mathsf{U}, x : A \vdash t : A$$

- For any predicate on A, if x respects it, so will t.
- Given a type B and $u : B$, we define $B^M\, x :\equiv (x \equiv u)$.
- $A :\equiv B, A^M :\equiv B^M, x :\equiv u, x^M :\equiv \mathsf{refl}$
- Now we get $A^M\, t = B^M\, t = (t \equiv u)$.

Logical predicate interpretation (ii)

- Bernardy-Jansson-Paterson: Parametricity and Dependent Types, 2012
- A type is interpreted as a logical predicate over that type

$$\frac{\Gamma \text{ valid}}{\Gamma^P \text{ valid}} \qquad \frac{\Gamma \vdash A : \mathsf{Set}}{\Gamma^P \vdash A^P : A \to \mathsf{Set}}$$

- A term is interpreted as a proof that it satisfies the predicate

$$\frac{\Gamma \vdash t : A}{\Gamma^P \vdash t^P : (A^P)\, t}$$

Logical predicate interpretation (iii)

An interpretation from the syntax into the syntax:

$$
\begin{aligned}
\bullet^{P} &= \bullet \\
(\Gamma, x : A)^{P} &= \Gamma^{P}, x : A, x^{M} : A^{P}\, x \\
U^{P} &= \lambda\, A \to (A \to U) \\
((x : A) \to B)^{P} &= \lambda\, f \to ((x : A)\,(x^{M} : A^{P}\, x) \\
&\qquad\qquad \to B^{P}\, (f\, x)) \\
(\lambda\, x \to t)^{P} &= \lambda\, x\, x^{M} \to t^{P} \\
(t\, u)^{P} &= t^{P}\, u\, (u^{P}) \\
x^{P} &= x^{M}
\end{aligned}
$$

These equations are all typed. Template type theory: automated derivation of free theorems

Normalisation

$$\text{completeness} \circlearrowleft \quad \text{norm} \downarrow \ \frac{\mathsf{Tm}\,\Gamma\,A}{\mathsf{Nf}\,\Gamma\,A} \uparrow \ulcorner - \urcorner \quad \curvearrowright \text{stability}$$

$$\begin{aligned}
&\mathsf{data}\ \mathsf{Ne} &&: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}\\
&\quad \mathsf{var} &&: \mathsf{Var}\,\Gamma\,A \to \mathsf{Ne}\,\Gamma\,A\\
&\quad \mathsf{app} &&: \mathsf{Ne}\,\Gamma\,(\Pi\,A\,B) \to (v : \mathsf{Nf}\,\Gamma\,A) \to \mathsf{Ne}\,\Gamma\,(B[\ulcorner v \urcorner])\\
&\mathsf{data}\ \mathsf{Nf} &&: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}\\
&\quad \mathsf{neuU} &&: \mathsf{Ne}\,\Gamma\,\mathsf{U} \to \mathsf{Nf}\,\Gamma\,\mathsf{U}\\
&\quad \mathsf{neuEl} &&: \mathsf{Ne}\,\Gamma\,(\mathsf{El}\,\hat{A}) \to \mathsf{Nf}\,\Gamma\,(\mathsf{El}\,\hat{A})\\
&\quad \mathsf{lam} &&: \mathsf{Nf}\,(\Gamma, A)\,B \to \mathsf{Nf}\,\Gamma\,(\Pi\,A\,B)
\end{aligned}$$

Presheaf model

- Proof relevant version of Kripke model: category instead of poset
- Given a category $\mathcal{C}$
- Contexts are presheaves over $\mathcal{C}$: for every object of $\mathcal{C}$ we have a set and for morphisms we get maps between the sets
- Types are families of presheaves, terms are sections
- We need to give interpretations to the base type

NBE for simple type theory (i)

- Presheaf model over the category of renamings $\mathsf{REN}^{\mathsf{op}}$
  - Objects are contexts
  - Morphisms are renamings (lists of variables)
- The base type $\bullet$ at Γ is interpreted as $\mathsf{Ne}\,\Gamma\,\bullet$
- We denote the interpretation $[\![-]\!]$
- We define quote and unquote mutually:

$$\mathsf{u}_A : \mathsf{NE}_A \dot{\to} [\![A]\!] \qquad \mathsf{q}_A : [\![A]\!] \dot{\to} \mathsf{NF}_A$$

$$\mathsf{norm}_A\,(t : \mathsf{Tm}\,\Gamma\,A) :\equiv \mathsf{q}_A\,([\![t]\!]\,\mathsf{id}_\Gamma)$$

NBE for simple type theory (ii)

- Presheaf model over $\mathsf{REN}^{\mathsf{op}}$, base type $\bullet$ is $\mathsf{NE}\ \bullet$
- For completeness, we need a logical relation
  - metatheoretic
  - Kripke (base category $\mathsf{REN}^{\mathsf{op}}$)
  - binary
  - proof-irrelevant
  - relation at $\bullet$ is equality

$$
\begin{tikzcd}
\mathsf{NE}_A \arrow[r, "\mathsf{u}_A"] \arrow[dr, "\ulcorner \_ \urcorner"'] & \Sigma\,(\mathsf{TM}_A \times [\![A]\!])\,\mathsf{R}_A \arrow[r, "\mathsf{q}_A"] \arrow[d, "\mathsf{proj}"] & \mathsf{NF}_A \arrow[dl, "\ulcorner \_ \urcorner"] \\
 & \mathsf{TM}_A &
\end{tikzcd}
$$

NBE for type theory

- No need for presheaf model
- Instead we have a logical predicate
  - metatheoretic
  - Kripke (base category $\mathrm{REN}^{\mathrm{op}}$)
  - unary
  - proof-relevant
  - predicate at $\bullet$: $\lambda\, t\, .\, \Sigma\, (n\ :\ \mathsf{Ne}\, \Gamma\, \bullet)\, .\, n\ \equiv\ t$

$$\begin{tikzcd}
\mathsf{NE}_\Delta \arrow[r, "u_\Delta"] \arrow[dr, "\ulcorner_\urcorner"'] & \Sigma\, \mathsf{TM}_\Delta\, \Delta^{\mathsf{P}} \arrow[r, "q_\Delta"] \arrow[d, "\mathsf{proj}"] & \mathsf{NF}_\Delta \arrow[dl, "\ulcorner_\urcorner"] \\
& \mathsf{TM}_\Delta &
\end{tikzcd}$$

Further work

- We internalized a very basic type theory, this can be extended easily with universes and inductive types. How to do it a nice categorical way?
- We used axioms (quotient inductive types, functional extensionality) in our metatheory. This can be solved by cubical type theory.
- Still lots of boilerplate equality reasoning. Solution: informally extensional type theory, formally 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