Normalisation by Evaluation for Dependent Types

Ambrus Kaposi
Eötvös Loránd University, Budapest, Hungary

(j.w.w. Thorsten Altenkirch, University of Nottingham)

FSCD, Porto
24 June 2016

Introduction

- Goal:
  - Prove normalisation for a type theory with dependent types
  - Using the metalanguage of type theory itself
- Structure of the talk:
  - Representing type theory in type theory
  - Specifying normalisation
  - NBE for simple types
  - NBE for dependent types

Representing type theory in type theory

Simple type theory the traditional way

Set of variables, alphabet including $\Rightarrow$, λ etc.

Well-formed expressions:

$$
\begin{aligned}
A &::= \iota \mid A \Rightarrow A' \\
\Gamma &::= \cdot \mid \Gamma, x : A \\
t &::= x \mid \lambda x.t \mid t\, t'
\end{aligned}
$$

An inductively defined relation:

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash t : A}{\Gamma.x : B \vdash t : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}$$

Simple type theory in idealised Agda

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

Rules for dependent function space and a base type

$$\frac{\Gamma \vdash A \qquad \Gamma.x : A \vdash B}{\Gamma \vdash \Pi(x : A).B}$$

$$\frac{\Gamma.x : A \vdash t : B}{\Gamma \vdash \lambda x.t : \Pi(x : A).B} \qquad \frac{\Gamma \vdash f : \Pi(x : A).B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\, a : B[x \mapsto a]}$$

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{U}} \qquad \frac{\Gamma \vdash \hat{A} : \mathsf{U}}{\Gamma \vdash \mathsf{El}\, \hat{A}}$$

A typed syntax of dependent types (i)

- Types depend on contexts
  $\Rightarrow$ We need induction induction.

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Con} &:&\ \mathsf{Set} \\
&\textbf{data}\ \mathsf{Ty} &:&\ \mathsf{Con} \rightarrow \mathsf{Set}
\end{aligned}
$$

A typed syntax of dependent types (ii)

- Types depend on contexts
  $\Rightarrow$ We need induction induction.

- Substitutions are mentioned in the application rule:

$$\mathsf{app} : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[a])$$

  $\Rightarrow$ We define an explicit substitution calculus.

$$
\begin{array}{lcl}
\textbf{data}\ \mathsf{Con} & : & \mathsf{Set} \\
\textbf{data}\ \mathsf{Ty} & : & \mathsf{Con} \to \mathsf{Set} \\
\textbf{data}\ \mathsf{Tms} & : & \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
\textbf{data}\ \mathsf{Tm} & : & (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \\
\quad \_[\_] & : & \mathsf{Ty}\,\Gamma \to \mathsf{Tms}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta \\
\quad \ldots & &
\end{array}
$$

A typed syntax of dependent types (iii)

- Types depend on contexts.
  $\Rightarrow$ We need induction induction.
- Substitutions are mentioned in the application rule:
  $\Rightarrow$ We define an explicit substitution calculus.
- The following conversion rule for terms:

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

  $\Rightarrow$ Conversion (the relation including β, η) needs to be defined mutually with the syntax.

  - We need to add 4 new members to the inductive inductive definition: $\sim$ for contexts, types, substitutions and terms.

Representing conversion

- Lots of boilerplate:
  - The $\sim$ relations are equivalence relations
  - Coercion rules
  - Congruence rules
  - We need to work with setoids
- What we really want is to redefine equality $\_ \equiv \_$ for the types representing the syntax.

Higher inductive types (HITs)

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

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

Higher inductive types (HITs)

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

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

$$
\begin{array}{lll}
\mathsf{RecI} : & (\mathsf{I}^{\mathsf{M}} : \mathsf{Set}) & \\
& (\mathsf{left}^{\mathsf{M}} & : \mathsf{I}^{\mathsf{M}}) \\
& (\mathsf{right}^{\mathsf{M}} & : \mathsf{I}^{\mathsf{M}}) \\
& (\mathsf{segment}^{\mathsf{M}} & : \mathsf{left}^{\mathsf{M}} \equiv \mathsf{right}^{\mathsf{M}}) \\
\quad \to \mathsf{I} \to \mathsf{I}^{\mathsf{M}} & &
\end{array}
$$

Using the syntax

- We define the syntax as a HIIT, the conversion rules are constructors: e.g. $\beta : \mathsf{app}\,(\mathsf{lam}\,t)\,u \equiv t[u]$.
- The arguments of the non-dependent eliminator form a model of type theory, equivalent to Categories with Families.

$$
\begin{array}{rll}
\textbf{record}\ \mathsf{Model} & : & \mathsf{Set}\ \textbf{where} \\
\textbf{field}\ \mathsf{Con}^{\mathsf{M}} & : & \mathsf{Set} \\
\mathsf{Ty}^{\mathsf{M}} & : & \mathsf{Con}^{\mathsf{M}} \to \mathsf{Set} \\
\mathsf{Tm}^{\mathsf{M}} & : & (\Gamma : \mathsf{Con}^{\mathsf{M}}) \to \mathsf{Ty}^{\mathsf{M}}\,\Gamma \to \mathsf{Set} \\
\mathsf{lam}^{\mathsf{M}} & : & \mathsf{Tm}^{\mathsf{M}}\,(\Gamma\,,^{\mathsf{M}}\,\mathsf{A})\,\mathsf{B}^{\mathsf{M}} \to \mathsf{Tm}^{\mathsf{M}}\,\Gamma\,(\Pi^{\mathsf{M}}\,\mathsf{A}\,\mathsf{B}) \\
\beta^{\mathsf{M}} & : & \mathsf{app}^{\mathsf{M}}\,(\mathsf{lam}^{\mathsf{M}}\,\mathsf{t})\,\mathsf{u} \equiv \mathsf{t}\,[\,\mathsf{u}\,]^{\mathsf{M}} \\
\ldots & &
\end{array}
$$

- The eliminator says that the syntax is the initial model.

Specifying normalisation

Specifying normalisation

Neutral terms and normal forms (typed!):

$$\begin{array}{ll} \mathsf{n} ::= \mathsf{x} \mid \mathsf{n}\ \mathsf{v} & \mathsf{Ne}\ \Gamma\ \mathsf{A} \\ \mathsf{v} ::= \mathsf{n} \mid \lambda\, \mathsf{x} \,.\, \mathsf{v} & \mathsf{Nf}\ \Gamma\ \mathsf{A} \end{array}$$

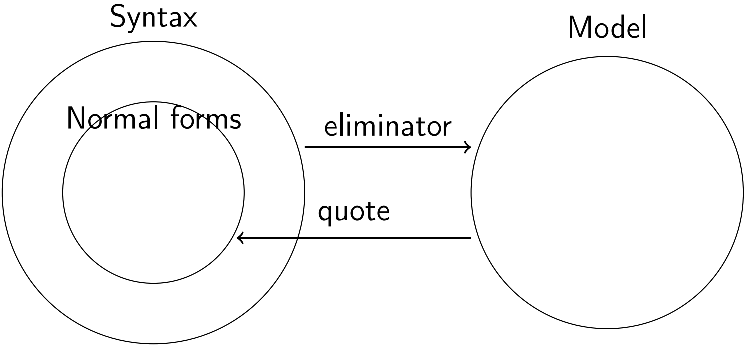
Normalisation is an isomorphism:

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

Soundness is given by congruence of equality:

$$t \equiv t' \to \mathsf{norm}\ t \equiv \mathsf{norm}\ t'$$

Normalisation by Evaluation (NBE)



- First formulation (Berger and Schwichtenberg, 1991)
- Simply typed case (Altenkirch, Hofmann, Streicher 1995)
- Dependent types using untyped realizers (Abel, Coquand, Dybjer, 2007)

NBE for simple types

The presheaf model

- Presheaf models are proof-relevant versions of Kripke models.
- They are parameterised over a category, we choose REN: objects are contexts, morphisms are lists of variables.

The presheaf model

- Presheaf models are proof-relevant versions of Kripke models.
- They are parameterised over a category, we choose REN: objects are contexts, morphisms are lists of variables.
- A type A is interpreted as a presheaf $[\![A]\!] : \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set}$.
  - Given a context Γ we have $[\![A]\!]_\Gamma : \mathsf{Set}$.
  - Given a renaming $\beta : \mathsf{REN}(\Delta, \Gamma)$, there is a $[\![A]\!]_\Gamma \to [\![A]\!]_\Delta$.

The presheaf model

- Presheaf models are proof-relevant versions of Kripke models.
- They are parameterised over a category, we choose REN: objects are contexts, morphisms are lists of variables.
- A type A is interpreted as a presheaf $[\![A]\!] : \mathrm{REN}^{\mathrm{op}} \to \mathrm{Set}$.
  - Given a context Γ we have $[\![A]\!]_\Gamma : \mathrm{Set}$.
  - Given a renaming $\beta : \mathrm{REN}(\Delta, \Gamma)$, there is a $[\![A]\!]_\Gamma \to [\![A]\!]_\Delta$.
- The function type is interpreted as the "possible world" function space: $[\![A \Rightarrow B]\!]_\Gamma = \forall \Delta . \mathrm{REN}(\Delta, \Gamma) \to [\![A]\!]_\Delta \to [\![B]\!]_\Delta$.

The presheaf model

- Presheaf models are proof-relevant versions of Kripke models.
- They are parameterised over a category, we choose REN: objects are contexts, morphisms are lists of variables.
- A type A is interpreted as a presheaf $[\![A]\!] : \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set}$.
  - Given a context Γ we have $[\![A]\!]_\Gamma : \mathsf{Set}$.
  - Given a renaming $\beta : \mathsf{REN}(\Delta, \Gamma)$, there is a $[\![A]\!]_\Gamma \to [\![A]\!]_\Delta$.
- The function type is interpreted as the "possible world" function space: $[\![A \Rightarrow B]\!]_\Gamma = \forall \Delta.\mathsf{REN}(\Delta, \Gamma) \to [\![A]\!]_\Delta \to [\![B]\!]_\Delta$.
- The interpretation of the base type is another parameter. We choose $[\![\iota]\!]_\Gamma = \mathsf{Nf}\, \Gamma\, \iota$.

Quotation

The quote function is a natural transformation

$$\mathsf{quote}_A : [\![A]\!] \mathbin{\dot{\to}} \mathsf{Nf} - A$$

i.e.

$$\mathsf{quote}_{A\,\Gamma} : [\![A]\!]_\Gamma \;\to\; \mathsf{Nf}\,\Gamma\,A$$

Defined mutually with unquote:

$$\mathsf{unquote}_A : \mathsf{Ne} - A \mathbin{\dot{\to}} [\![A]\!]$$

Quote and unquote

$$\mathsf{Ne} - A \xrightarrow{\quad \mathsf{unquote}_A \quad} \qquad\qquad [\![A]\!] \qquad \xrightarrow{\quad \mathsf{quote}_A \quad} \mathsf{Nf} - A$$

With completeness

$$\begin{tikzcd} \mathsf{Ne} - A \arrow[r, "\mathsf{unquote}'_A"] \arrow[dr, "\ulcorner \_ \urcorner"'] & \Sigma\,(\mathsf{Tm} - A \times [\![A]\!])\,\mathsf{R}_A \arrow[r, "\mathsf{quote}'_A"] \arrow[d, "\mathsf{proj}"] & \mathsf{Nf} - A \arrow[dl, "\ulcorner \_ \urcorner"] \\ & \mathsf{Tm} - A & \end{tikzcd}$$

R_A is a presheaf logical relation between the syntax and the presheaf model. It says equality at the base type.

NBE for dependent types

The presheaf model and quote

Types are interpreted as families of presheaves.

$$
\begin{aligned}
[\![\Gamma]\!] \quad &: \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set} \\
[\![\Gamma \vdash A]\!] &: (\Delta : \mathsf{REN}) \to [\![\Gamma]\!]_\Delta \to \mathsf{Set}
\end{aligned}
$$

The presheaf model and quote

Types are interpreted as families of presheaves.

$$\begin{aligned} [\![\Gamma]\!] &: \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set} \\ [\![\Gamma \vdash A]\!] &: (\Delta : \mathsf{REN}) \to [\![\Gamma]\!]_\Delta \to \mathsf{Set} \end{aligned}$$

We define quote for contexts and types mutually.

$$\begin{aligned} \mathsf{quote}_\Gamma &: [\![\Gamma]\!] \dot{\to} \mathsf{Nfs} - \Gamma \\ \mathsf{quote}_{\Gamma \vdash A} &: (\alpha : [\![\Gamma]\!]_\Delta) \to [\![A]\!]_\Delta\, \alpha \to \mathsf{Nf}\, \Delta\, \big(A[\mathsf{quote}_{\Gamma,\Delta}\, \alpha]\big) \end{aligned}$$

Defining quote, first try

$$\mathsf{Nes} - \Gamma \xrightarrow{\mathsf{unquote}_\Gamma} \qquad [\![\Gamma]\!] \qquad \xrightarrow{\mathsf{quote}_\Gamma} \mathsf{Nfs} - \Gamma$$

Defining quote, first try

$$\mathsf{Nes} - \Gamma \xrightarrow{\mathsf{unquote}_\Gamma} \qquad [\![\Gamma]\!] \qquad \xrightarrow{\mathsf{quote}_\Gamma} \mathsf{Nfs} - \Gamma$$

Quote for function space needs $\mathsf{quote}_A \circ \mathsf{unquote}_A \equiv \mathsf{id}$.

Defining quote, first try

$$\mathsf{Nes} - \Gamma \xrightarrow{\mathsf{unquote}_\Gamma} \qquad\qquad \llbracket \Gamma \rrbracket \qquad \xrightarrow{\mathsf{quote}_\Gamma} \mathsf{Nfs} - \Gamma$$

Quote for function space needs $\mathsf{quote}_A \circ \mathsf{unquote}_A \equiv \mathsf{id}$.
This follows from the logical relation R_A.

Defining quote, first try

$$\mathrm{Nes} - \Gamma \xrightarrow{\mathrm{unquote}_\Gamma} \qquad \llbracket \Gamma \rrbracket \qquad \xrightarrow{\mathrm{quote}_\Gamma} \mathrm{Nfs} - \Gamma$$

Quote for function space needs $\mathrm{quote}_A \circ \mathrm{unquote}_A \equiv \mathrm{id}$.
This follows from the logical relation R_A.
Let's define quote and completeness mutually!

Defining quote, second try

$$
\begin{tikzcd}
\mathsf{Nes} - \Gamma \arrow[r, "\mathsf{unquote}_\Gamma"] \arrow[dr, "\ulcorner \_ \urcorner"'] & \Sigma\,(\mathsf{Tms} - \Gamma \times \llbracket \Gamma \rrbracket)\,\mathsf{R}_\Gamma \arrow[r, "\mathsf{quote}_\Gamma"] \arrow[d, "\mathsf{proj}"] & \mathsf{Nfs} - \Gamma \arrow[dl, "\ulcorner \_ \urcorner"] \\
& \mathsf{Tms} - \Gamma &
\end{tikzcd}
$$

Defining quote, second try

$$
\begin{tikzcd}
\mathsf{Nes} - \Gamma \arrow[r, "\mathsf{unquote}_\Gamma"] \arrow[dr, "\ulcorner \_ \urcorner"'] & \Sigma\,(\mathsf{Tms} - \Gamma \times [\![\Gamma]\!])\,\mathsf{R}_\Gamma \arrow[r, "\mathsf{quote}_\Gamma"] \arrow[d, "\mathsf{proj}"] & \mathsf{Nfs} - \Gamma \arrow[dl, "\ulcorner \_ \urcorner"] \\
& \mathsf{Tms} - \Gamma &
\end{tikzcd}
$$

For unquote at the function space we need to define a semantic function which works for every input, not necessarily related by the relation. But quote needs ones which are related!

Defining quote, last try

$$
\begin{tikzcd}
\mathsf{Nes} - \Gamma \arrow[r, "\mathsf{unquote}_\Gamma"] \arrow[dr, "\ulcorner \_ \urcorner"'] & \Sigma\,(\mathsf{Tms} - \Gamma)\,\mathsf{P}_\Gamma \arrow[r, "\mathsf{quote}_\Gamma"] \arrow[d, "\mathsf{proj}"] & \mathsf{Nfs} - \Gamma \arrow[dl, "\ulcorner \_ \urcorner"] \\
& \mathsf{Tms} - \Gamma &
\end{tikzcd}
$$

Use a presheaf logical predicate.

Presheaf logical predicate

- The Yoneda embedding of the syntax:

$$
\begin{aligned}
\mathsf{Y}_\Gamma &: \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set} &&:= \mathsf{Tms} - \Gamma \\
\mathsf{Y}_A &: \Sigma_{\mathsf{REN}}\, \mathsf{Y}_\Gamma \to \mathsf{Set} &&:= \mathsf{Tm} - A[-] \\
\mathsf{Y}_\sigma &: \mathsf{Y}_\Gamma \overset{\cdot}{\to} \mathsf{Y}_\Delta &&:= \sigma \circ - \\
\mathsf{Y}_t &: \mathsf{Y}_\Gamma \xrightarrow{\mathsf{s}} \mathsf{Y}_A &&:= t[-]
\end{aligned}
$$

Presheaf logical predicate

- The Yoneda embedding of the syntax.
- P is a dependent version of the presheaf model:

$$
\begin{array}{llll}
\mathsf{Y}_\Gamma : \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set} & := \mathsf{Tms} - \Gamma & \qquad \mathsf{P}_\Gamma : \Sigma_{\mathsf{REN}}\, \mathsf{Y}_\Gamma \to \mathsf{Set} \\
\mathsf{Y}_A : \Sigma_{\mathsf{REN}}\, \mathsf{Y}_\Gamma \to \mathsf{Set} & := \mathsf{Tm} - A[-] & \qquad \mathsf{P}_A : \Sigma_{\mathsf{REN}, \mathsf{Y}_\Gamma, \mathsf{Y}_A}\, \mathsf{P}_\Gamma \to \mathsf{Set} \\
\mathsf{Y}_\sigma : \mathsf{Y}_\Gamma \stackrel{.}{\to} \mathsf{Y}_\Delta & := \sigma \circ - & \qquad \mathsf{P}_\sigma : \Sigma_{\mathsf{Y}_\Gamma}\, \mathsf{P}_\Gamma \stackrel{\mathsf{s}}{\to} \mathsf{P}_\Delta[\mathsf{Y}_\sigma] \\
\mathsf{Y}_t : \mathsf{Y}_\Gamma \stackrel{\mathsf{s}}{\to} \mathsf{Y}_A & := t[-] & \qquad \mathsf{P}_t : \Sigma_{\mathsf{Y}_\Gamma}\, \mathsf{P}_\Gamma \stackrel{\mathsf{s}}{\to} \mathsf{P}_A[\mathsf{Y}_t]
\end{array}
$$

Presheaf logical predicate

- The Yoneda embedding of the syntax.
- P is a dependent version of the presheaf model:

$$
\begin{array}{llll}
\mathsf{Y}_\Gamma : \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set} & := \mathsf{Tms} - \Gamma & \qquad \mathsf{P}_\Gamma : \Sigma_{\mathsf{REN}}\, \mathsf{Y}_\Gamma \to \mathsf{Set} \\
\mathsf{Y}_A : \Sigma_{\mathsf{REN}}\, \mathsf{Y}_\Gamma \to \mathsf{Set} & := \mathsf{Tm} - A[-] & \qquad \mathsf{P}_A : \Sigma_{\mathsf{REN}, \mathsf{Y}_\Gamma, \mathsf{Y}_A}\, \mathsf{P}_\Gamma \to \mathsf{Set} \\
\mathsf{Y}_\sigma : \mathsf{Y}_\Gamma \dot{\to} \mathsf{Y}_\Delta & := \sigma \circ - & \qquad \mathsf{P}_\sigma : \Sigma_{\mathsf{Y}_\Gamma}\, \mathsf{P}_\Gamma \xrightarrow{\mathsf{s}} \mathsf{P}_\Delta[\mathsf{Y}_\sigma] \\
\mathsf{Y}_t : \mathsf{Y}_\Gamma \xrightarrow{\mathsf{s}} \mathsf{Y}_A & := t[-] & \qquad \mathsf{P}_t : \Sigma_{\mathsf{Y}_\Gamma}\, \mathsf{P}_\Gamma \xrightarrow{\mathsf{s}} \mathsf{P}_A[\mathsf{Y}_t]
\end{array}
$$

- We need the dependent eliminator to define it.
- At the base type:
  - We had: $[\![\iota]\!]_\Gamma = \mathsf{Nf}\,\Gamma\,\iota$ and $\mathsf{R}_\iota\, t\, n = (t \equiv \ulcorner n \urcorner)$
  - Now we have: $\mathsf{P}_\iota\, t = \Sigma(n : \mathsf{Nf}\,\Gamma\,\iota).(t \equiv \ulcorner n \urcorner)$

Summary

- We defined the typed syntax of type theory as an explicit substitution calculus using a quotient inductive inductive type
- Normalisation is specified as an isomorphism between terms and normal forms
- We proved normalisation and completeness using a proof-relevant presheaf logical predicate
- Most of this has been formalised in Agda
- Stability, injectivity of type constructors can be proven
- Question: how to prove decidability of conversion? N.b. normal forms are indexed by non-normal types