Normalisation by Evaluation for Dependent Types

Ambrus Kaposi
Eötvös Loránd University, Budapest, Hungary
(joint work with Thorsten Altenkirch of Nottingham)

TYPES, Нови Сад
25 May 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 in idealised Agda

```
data Ty    : Set where
  ι        : Ty
  _⇒_      : Ty → Ty → Ty
data Con   : Set where
  •        : Con
  _,_      : Con → Ty → Con
data Var   : Con → Ty → Set where
  zero     : Var (Γ , A) A
  suc      : Var Γ A → Var (Γ , B) A
data Tm    : Con → Ty → Set where
  var      : Var Γ A → Tm Γ A
  lam      : Tm (Γ , A) B → Tm Γ (A ⇒ B)
  app      : Tm Γ (A ⇒ B) → Tm Γ A → Tm Γ B
```

Simple type theory in idealised Agda

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

No preterms!

A typed syntax of dependent types (i)

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

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

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.

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

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
- The identity type $\_ \equiv \_$ is an equivalence relation with coercion and congruence laws.
- Higher inductive types are an idea from homotopy type theory: constructors for equalities.
- We add the conversion rules as constructors: e.g. $\beta : \mathsf{app}\,(\mathsf{lam}\,t)\,u \equiv t[u]$.

QIITs

We formalise the syntax of type theory as a quotient inductive inductive type (QIIT).

- A QIT is a HIT which is a set
- QITs are not the same as quotient types

Using the syntax

- One defines functions from a QIIT using its eliminator.
- The arguments of the non-dependent eliminator form a model of type theory, equivalent to Categories with Families.

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

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

Specifying normalisation

Specifying normalisation

Neutral terms and normal forms (typed!):

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

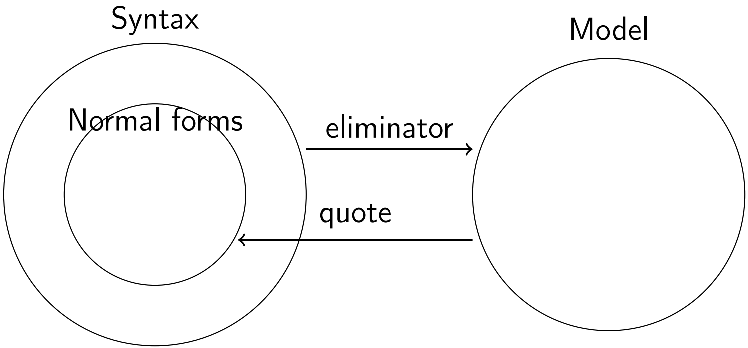
Normalisation is an isomorphism:

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

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, here 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, here we choose REN: objects are contexts, morphisms are lists of variables.
- A context Γ is interpreted as a presheaf $[\![\Gamma]\!] : \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set}$.
  - Given another context Δ we have $[\![\Gamma]\!]_\Delta : \mathsf{Set}$.
  - Given a renaming $\Delta \xrightarrow{\beta} \Theta$, there is a $[\![\Gamma]\!]_\Theta \xrightarrow{[\![\Gamma]\!]\,\beta} [\![\Gamma]\!]_\Delta$.

The presheaf model

- Presheaf models are proof-relevant versions of Kripke models.
- They are parameterised over a category, here we choose REN: objects are contexts, morphisms are lists of variables.
- A context Γ is interpreted as a presheaf $[\![\Gamma]\!] : \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set}$.
  - Given another context Δ we have $[\![\Gamma]\!]_\Delta : \mathsf{Set}$.
  - Given a renaming $\Delta \xrightarrow{\beta} \Theta$, there is a $[\![\Gamma]\!]_\Theta \xrightarrow{[\![\Gamma]\!]\,\beta} [\![\Gamma]\!]_\Delta$.
- Types are presheaves too: $[\![A]\!] : \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set}$
  - $[\![\iota]\!]_\Delta := \mathsf{Nf}\,\Delta\,\iota$

The presheaf model

- Presheaf models are proof-relevant versions of Kripke models.
- They are parameterised over a category, here we choose REN: objects are contexts, morphisms are lists of variables.
- A context Γ is interpreted as a presheaf $[\![\Gamma]\!] : \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set}$.
  - Given another context Δ we have $[\![\Gamma]\!]_\Delta : \mathsf{Set}$.
  - Given a renaming $\Delta \xrightarrow{\beta} \Theta$, there is a $[\![\Gamma]\!]_\Theta \xrightarrow{[\![\Gamma]\!]\,\beta} [\![\Gamma]\!]_\Delta$.
- Types are presheaves too: $[\![A]\!] : \mathsf{REN}^{\mathsf{op}} \to \mathsf{Set}$
  - $[\![\iota]\!]_\Delta := \mathsf{Nf}\,\Delta\,\iota$

Quotation

The quote function is a natural transformation.

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

At a given context we have:

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

It is defined mutually with unquote:

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

Quote and unquote

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

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 is equality at the base type.

NBE for dependent types

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

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

When we try to define this quote for function space, we need the equation $\mathsf{quote}_A \circ \mathsf{unquote}_A \equiv \mathsf{id}$.

Defining quote, first try

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

When we try to define this quote for function space, we need the equation $\mathsf{quote}_A \circ \mathsf{unquote}_A \equiv \mathsf{id}$.
Let's define quote and its 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

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

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, third 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 proof-relevant logical predicate. At the base type it says that there exists a normal form which is equal to the term.
Instance of categorical glueing.

Extra slides

The presheaf model and quote

For dependent types, 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

For dependent types, types are interpreted as families of presheaves.

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

Quote for contexts is the same, but for types it is more subtle:

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

Quotation

The quote function is a natural transformation.

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

For the base type it is the identity.

$\mathsf{quote}_\iota\ v := v$

For function types:

$\mathsf{quote}_{A \to B\, \Delta} \left(f : \forall \Theta.(\beta : \Theta \to \Delta) \to [\![A]\!]_\Theta \to [\![B]\!]_\Theta \right) : \mathsf{Nf}\, \Delta\, (A \to B)$

$:= \mathsf{lam} \Big($

$\uparrow \mathsf{Nf}\, (\Delta, A)\, B$

Quotation

The quote function is a natural transformation.

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

For the base type it is the identity.

$\mathsf{quote}_\iota\ v := v$

For function types:

$\mathsf{quote}_{A \to B\,\Delta} \left(f : \forall \Theta.(\beta : \Theta \to \Delta) \to [\![A]\!]_\Theta \to [\![B]\!]_\Theta\right) : \mathsf{Nf}\,\Delta\,(A \to B)$

$:= \mathsf{lam}\left(\mathsf{quote}_{B,(\Delta,A)}\ (\right.$

$\uparrow [\![B]\!]_{\Delta,A}$

Quotation

The quote function is a natural transformation.

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

For the base type it is the identity.

$\mathsf{quote}_\iota\ v := v$

For function types:

$\mathsf{quote}_{A \to B\,\Delta} \left(f : \forall \Theta.(\beta : \Theta \to \Delta) \to [\![A]\!]_\Theta \to [\![B]\!]_\Theta\right) : \mathsf{Nf}\,\Delta\,(A \to B)$

$:= \mathsf{lam}\left(\mathsf{quote}_{B,(\Delta,A)}\left(f_{\Delta,A}\right.\right.$

$$\uparrow \Delta, A \to \Delta$$

Quotation

The quote function is a natural transformation.

$$\mathsf{quote}_A : [\![A]\!] \stackrel{.}{\rightarrow} \mathsf{Nf} - A$$

For the base type it is the identity.

$\mathsf{quote}_\iota\ v := v$

For function types:

$\mathsf{quote}_{A \rightarrow B\,\Delta} \left(f : \forall\Theta.(\beta : \Theta \rightarrow \Delta) \rightarrow [\![A]\!]_\Theta \rightarrow [\![B]\!]_\Theta\right) : \mathsf{Nf}\,\Delta\,(A \rightarrow B)$

$:= \mathsf{lam}\left(\mathsf{quote}_{B,(\Delta,A)}\left(f_{\Delta,A} \quad \mathsf{wk} \quad (\right.\right.$

$\uparrow [\![A]\!]_{\Delta,A}$

Quotation

The quote function is a natural transformation.

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

For the base type it is the identity.

$\mathsf{quote}_\iota\ v := v$

For function types:

$$\mathsf{quote}_{A \to B\,\Delta}\left(f : \forall\Theta.(\beta : \Theta \to \Delta) \to [\![A]\!]_\Theta \to [\![B]\!]_\Theta\right) : \mathsf{Nf}\,\Delta\,(A \to B)$$

$$:= \mathsf{lam}\left(\mathsf{quote}_{B,(\Delta,A)}\left(f_{\Delta,A} \quad \mathsf{wk} \qquad (\right.\right.$$

$$\uparrow [\![A]\!]_{\Delta,A}$$

We need to unquote neutral terms: $\mathsf{unquote}_A : \mathsf{Ne} - A \mathrel{\dot{\to}} [\![A]\!]$.

Quotation

The quote function is a natural transformation.

$$\mathsf{quote}_A : [\![A]\!] \stackrel{.}{\to} \mathsf{Nf} - A$$

For the base type it is the identity.

$\mathsf{quote}_\iota\ v := v$

For function types:

$\mathsf{quote}_{A \to B\,\Delta}\left(f : \forall\Theta.(\beta : \Theta \to \Delta) \to [\![A]\!]_\Theta \to [\![B]\!]_\Theta\right) : \mathsf{Nf}\,\Delta\,(A \to B)$

$:= \mathsf{lam}\left(\mathsf{quote}_{B,(\Delta,A)}\left(f_{\Delta,A} \quad \mathsf{wk} \qquad\qquad (\mathsf{unquote}_{A\,(\Delta,A)}\,\mathsf{zero})\right)\right)$

We need to unquote neutral terms: $\mathsf{unquote}_A : \mathsf{Ne} - A \stackrel{.}{\to} [\![A]\!]$.