Normalisation by Evaluation for Dependent Types

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

TYPES, Novi Sad
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

```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

```agda
data Ty : Set where
  λ        : Ty
  _⇒_      : Ty → Ty → Ty

data Con : Set where
  •        : Con
  _,_      : 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
```

No preterms!
A typed syntax of dependent types (i)

- Types depend on contexts.
  \[\Rightarrow\] We need induction induction.

```agda
data Con : Set
data Ty : Con \to Set
```
A typed syntax of dependent types (ii)

- Types depend on contexts.
  ⇒ We need induction induction.

- Substitutions are mentioned in the application rule:

\[
\text{app} : \text{Tm} \Gamma (\Pi A B) \rightarrow (a : \text{Tm} \Gamma A) \rightarrow \text{Tm} \Gamma (B[a])
\]

⇒ We define an explicit substitution calculus.

```
data Con : Set
data Ty : Con → Set
data Tms : Con → Con → Set
data Tm : (Γ : Con) → Ty Γ → Set
  _[_] : Ty Γ → Tms Δ Γ → Ty Δ
...
```
A typed syntax of dependent types (iii)

- Types depend on contexts.
  ⇒ We need induction induction.

- Substitutions are mentioned in the application rule:
  ⇒ We define an explicit substitution calculus.

- The following conversion rule for terms:
  \[
  \Gamma \vdash A \sim B \quad \Gamma \vdash t : A \\
  \Gamma \vdash t : B
  \]

  ⇒ Conversion (the relation including $\beta$, $\eta$) 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 $\_ \equiv \_ \equiv$ 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. $eta : \text{app (lam } t \text{) } u \equiv t[u]$. 
QIIITs

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

- A QIT is a HIT which is a set
- QIIITs 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.

```agda
record Model : Set where
  field Con^M : Set
  Ty^M : Con^M → Set
  Tm^M : (Γ : Con^M) → Ty^M Γ → Set
  lam^M : Tm^M (Γ ,^M A) B^M → Tm^M Γ (Π^M A B)
  β^M : app^M (lam^M t) a ≡ t [ a ]^M
...
```

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

Neutral terms and normal forms (typed!):

\[ n ::= x \mid n \, v \quad \text{Ne} \, \Gamma \, A \]
\[ v ::= n \mid \lambda x . v \quad \text{Nf} \, \Gamma \, A \]

Normalisation is an isomorphism:

\[
\begin{array}{c}
\text{completeness} \cup \quad \text{norm} \downarrow \\
\text{Soundness is given by congruence of equality:}
\end{array}
\]

\[ t \equiv t' \rightarrow \text{norm} \, t \equiv \text{norm} \, t' \]
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 $\Gamma$ is interpreted as a presheaf $\llbracket \Gamma \rrbracket : \text{REN}^{\text{op}} \to \text{Set}$.
  - Given another context $\Delta$ we have $\llbracket \Gamma \rrbracket_\Delta : \text{Set}$.
  - Given a renaming $\Delta \xrightarrow{\beta} \Theta$, there is a $\llbracket \Gamma \rrbracket_\Theta \xrightarrow{\llbracket \Gamma \rrbracket_\beta} \llbracket \Gamma \rrbracket_\Delta$. 
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 $\Gamma$ is interpreted as a presheaf $[\Gamma] : \text{REN}^{\text{op}} \to \text{Set}$.
  
  - Given another context $\Delta$ we have $[\Gamma]_\Delta : \text{Set}$.
  
  - Given a renaming $\Delta \xrightarrow{\beta} \Theta$, there is a $[\Gamma]_\Theta \xrightarrow{[\Gamma]_\beta} [\Gamma]_\Delta$.

- Types are presheaves too: $[A] : \text{REN}^{\text{op}} \to \text{Set}$
  
  - $[\iota]_\Delta := Nf \Delta \iota$
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 $\Gamma$ is interpreted as a presheaf $[\Gamma] : \text{REN}^{\text{op}} \to \text{Set}$.
  - Given another context $\Delta$ we have $[\Gamma]_{\Delta} : \text{Set}$.
  - Given a renaming $\Delta \xrightarrow{\beta} \Theta$, there is a $[\Gamma]_{\Theta} \xrightarrow{[\Gamma]_{\Theta} \beta} [\Gamma]_{\Delta}$.
- Types are presheaves too: $[A] : \text{REN}^{\text{op}} \to \text{Set}$
  - $[\iota]_{\Delta} := \text{Nf} \Delta \iota$
Quotation

The quote function is a natural transformation.

\[ \text{quote}_A : [A] \overset{\cdot}{\rightarrow} \text{Nf} - A \]

At a given context we have:

\[ \text{quote}_{A \Gamma} : [A]_{\Gamma} \overset{\cdot}{\rightarrow} \text{Nf} \Gamma A \]

It is defined mutually with unquote:

\[ \text{unquote}_A : \text{Ne} - A \overset{\cdot}{\rightarrow} [A] \]
Quote and unquote

\[ \text{Ne} - A \xrightarrow{\text{unquote} \ A} \ [A] \xrightarrow{\text{quote} \ A} \text{Nf} - A \]
With completeness

\[ \text{Ne} - A \xrightarrow{\text{unquote}_A'} \Sigma (\text{Tm} - A \times \llbracket A \rrbracket) R_A \xrightarrow{\text{quote}_A'} \text{Nf} - A \]

\[ \text{Tm} - A \]

\( R_A \) 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

\[
\text{Nes} - \Gamma \xrightarrow{\text{unquote}_\Gamma} [\Gamma] \xrightarrow{\text{quote}_\Gamma} \text{Nfs} - \Gamma
\]
Defining quote, first try

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

\[ \text{Nes} - \Gamma \xrightarrow{\text{unquote}_\Gamma} \llbracket \Gamma \rrbracket \xrightarrow{\text{quote}_\Gamma} \text{Nfs} - \Gamma \]

When we try to define this quote for function space, we need the equation \( \text{quote}_A \circ \text{unquote}_A \equiv \text{id} \).
Let’s define quote and its completeness mutually!
Defining quote, second try

\[
\text{Nes} - \Gamma \xrightarrow{\text{unquote}_\Gamma} \Sigma (\text{Tms} - \Gamma \times \llbracket \Gamma \rrbracket) R_\Gamma \xrightarrow{\text{quote}_\Gamma} \text{Nfs} - \Gamma
\]

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, second try

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

\[
\text{Nes} \rightarrow \Gamma \quad \xrightarrow{\text{unquote}_\Gamma} \quad \sum (\text{Tms} \rightarrow \Gamma) \quad P_\Gamma \quad \xrightarrow{\text{quote}_\Gamma} \quad \text{Nfs} \rightarrow \Gamma
\]

\[\Gamma \rightarrow \Gamma\]

\[\Gamma \rightarrow \Gamma\]

Tms \rightarrow \Gamma

\text{proj}

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.

$$\left[\Gamma\right] : \text{REN}^{\text{op}} \to \text{Set}$$

$$\left[\Gamma \vdash A\right] : (\Delta : \text{REN}) \to \left[\Gamma\right]_{\Delta} \to \text{Set}$$
The presheaf model and quote

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

\[
\llbracket \Gamma \rrbracket : \text{REN}^{\text{op}} \to \text{Set} \\
\llbracket \Gamma \vdash A \rrbracket : (\Delta : \text{REN}) \to \llbracket \Gamma \rrbracket_{\Delta} \to \text{Set}
\]

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

\[
\text{quote}\_\Gamma : \llbracket \Gamma \rrbracket \rightarrow \text{Tms} - \Gamma \\
\text{quote}\_\Gamma \vdash A : (\alpha : \llbracket \Gamma \rrbracket_{\Delta}) \rightarrow \llbracket A \rrbracket_{\Delta} \alpha \rightarrow \text{Nf} \; \Delta \left( A[\text{quote}\_\Gamma,_{\Delta} \alpha] \right)
\]
Quotation

The quote function is a natural transformation.

\[ \text{quote}_A : \llbracket A \rrbracket \rightarrow \text{Nf} - A \]

For the base type it is the identity.

\[ \text{quote}_\iota \, \nu := \nu \]

For function types:

\[ \text{quote}_{A \rightarrow B} \Delta \left( f : \forall \Theta. (\beta : \Theta \rightarrow \Delta) \rightarrow \llbracket A \rrbracket_\Theta \rightarrow \llbracket B \rrbracket_\Theta \right) : \text{Nf} \, \Delta \, (A \rightarrow B) \]

\[ := \text{lam} \left( \uparrow \text{Nf} \, (\Delta, A) \, B \right) \]
Quotation

The quote function is a natural transformation.

\[ \text{quote}_A : [A] \rightarrow \text{Nf} - A \]

For the base type it is the identity.

\[ \text{quote}_\iota \, \nu := \nu \]

For function types:

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

\[ := \text{lam} \left( \text{quote}_{B, (\Delta, A)} \left( \rightarrow [B]_{\Delta, A} \right) \right) \]
**Quotation**

The quote function is a natural transformation.

\[ \text{quote}_A : [A] \to \text{Nf} - A \]

For the base type it is the identity.

\[ \text{quote}_I \, \nu := \nu \]

For function types:

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

\[ := \text{lam} \left(\text{quote}_{B, (\Delta, A)} \left(f_{\Delta, A} \uparrow \Delta, A \to \Delta\right)\right) \]
Quotation

The quote function is a natural transformation.

\[ \text{quote}_A : [A] \rightarrow Nf - A \]

For the base type it is the identity.

\[ \text{quote}_I \nu := \nu \]

For function types:

\[ \text{quote}_{A \rightarrow B} \Delta (f : \forall \Theta. (\beta : \Theta \rightarrow \Delta) \rightarrow [A]_\Theta \rightarrow [B]_\Theta) : Nf \Delta (A \rightarrow B) \]

\[ := \text{lam} \left( \text{quote}_{B,(\Delta,A)} (f_{\Delta,A} \ \ \text{wk} \ \ \uparrow [A]_{\Delta,A} \right) \]
Quotation

The quote function is a natural transformation.

\[ \text{quote}_A : [A] \to \text{Nf} - A \]

For the base type it is the identity.

\[ \text{quote}_v \ v := v \]

For function types:

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

\[ := \text{lam} \left( \text{quote}_{B,(\Delta,A)} \left( f_{\Delta,A} \right) \text{ wk} \right) \]

We need to unquote neutral terms: \[ \text{unquote}_A : \text{Ne} - A \rightarrow [A] \].
Quotation

The quote function is a natural transformation.

\[
\text{quote}_A : \llbracket A \rrbracket \to \text{Nf} - A
\]

For the base type it is the identity.

\[
\text{quote}_v v := v
\]

For function types:

\[
\text{quote}_{A \to B} \Delta \left( f : \forall \Theta. (\beta : \Theta \to \Delta) \to \llbracket A \rrbracket_\Theta \to \llbracket B \rrbracket_\Theta \right) : \text{Nf} \ \Delta (A \to B)
\]

\[
:= \text{lam} \left( \text{quote}_{B,(\Delta,A)} \left( f_{\Delta,A} \ \text{wk} \ (\text{unquote}_{A(\Delta,A)} \text{ zero}) \right) \right)
\]

We need to unquote neutral terms: \( \text{unquote}_A : \text{Ne} - A \to \llbracket A \rrbracket. \)