

Hereditary Substitutions for Simple Types, Formalized

Chantal Keller

Thorsten Altenkirch

INRIA – LIX – ENS de Lyon – University of Nottingham

September, 25th 2010



Presentation

Objectives:

- a $\beta\eta$ -normalizer in Total Type Theory
- **formally** establish the decidability of the $\beta\eta$ -equivalence for the simply-typed λ -calculus

Tools:

- Agda and dependently typed programming
- terms parameterized by types and contexts
- type-checking
- hereditary substitutions

Agda

Agda:

- functional programming language
- dependent types
- proof assistant
- Total Type Theory

Termination checker:

- lexicographic order on inductive objects
- must decrease for any cycle in the call graph

Hereditary substitutions [Watkins,Cervesato,Pfenning03]

Idea:

- substitute and normalize at the same time
- **structurally recursive** on types and terms

Outline

- 1 Calculus
- 2 Normalization
- 3 Technical details
- 4 Conclusion

Simply-typed λ -calculus

Types:

data Ty : Set **where**

○ : Ty

 $_ \Rightarrow _ : \text{Ty} \rightarrow \text{Ty} \rightarrow \text{Ty}$

Terms:

data Var : Set **where**

vz : Var

vs : Var \rightarrow Var**data** Tm : Set **where**var : Var \rightarrow Tm Λ : Tm \rightarrow Tmapp : Tm \rightarrow Tm \rightarrow Tm

Simply-typed λ -calculus

Types:

data Ty : Set **where**

○ : Ty

$_ \Rightarrow _$: Ty \rightarrow Ty \rightarrow Ty

Terms:

data Var : Con \rightarrow Ty \rightarrow Set **where**

vz : Var (Γ, σ) σ

vs : Var Γ σ \rightarrow Var (Γ, τ) σ

data Tm : Con \rightarrow Ty \rightarrow Set **where**

var : Var Γ σ \rightarrow Tm Γ σ

Λ : Tm (Γ, σ) τ \rightarrow Tm Γ ($\sigma \Rightarrow \tau$)

app : Tm Γ ($\sigma \Rightarrow \tau$) \rightarrow Tm Γ σ \rightarrow Tm Γ τ

Simply-typed λ -calculus

Types:

data Ty : Set **where**

○ : Ty

 $_ \Rightarrow _ : \text{Ty} \rightarrow \text{Ty} \rightarrow \text{Ty}$ What can we do
with this syntax?

Terms:

data Var : Con \rightarrow Ty \rightarrow Set **where**vz : Var (Γ, σ) σ vs : Var Γ $\sigma \rightarrow$ Var (Γ, τ) σ **data** Tm : Con \rightarrow Ty \rightarrow Set **where**var : Var Γ $\sigma \rightarrow$ Tm Γ σ $\Lambda : \text{Tm } (\Gamma, \sigma) \tau \rightarrow \text{Tm } \Gamma (\sigma \Rightarrow \tau)$ app : Tm Γ ($\sigma \Rightarrow \tau$) \rightarrow Tm Γ $\sigma \rightarrow$ Tm Γ τ

Weakening

Minus:

“ $\Gamma - x$ ” means “the context Γ from which the variable x is removed”

Variable weakening:

$$\text{wkv} : (x : \text{Var } \Gamma \sigma) \rightarrow \text{Var } (\Gamma - x) \tau \rightarrow \text{Var } \Gamma \tau$$

Term weakening:

$$\text{wkTm} : (x : \text{Var } \Gamma \sigma) \rightarrow \text{Tm } (\Gamma - x) \tau \rightarrow \text{Tm } \Gamma \tau$$

Substitution

Term substitution:

$$\text{subst} : \text{Tm } \Gamma \tau \rightarrow (x : \text{Var } \Gamma \sigma) \rightarrow \text{Tm } (\Gamma - x) \sigma \rightarrow \text{Tm } (\Gamma - x) \tau$$

$\text{subst } t \ x \ u$

- $t [x:=u]$
- x not free either in u nor in $t [x:=u]$

$\beta\eta$ -convertibility

Convertibility:

```
data  $\_ \beta\eta\text{-}\equiv \_$  : Tm  $\Gamma$   $\sigma$   $\rightarrow$  Tm  $\Gamma$   $\sigma$   $\rightarrow$  Set where  
  brefl : {t : Tm  $\Gamma$   $\sigma$ }  $\rightarrow$  t  $\beta\eta\text{-}\equiv$  t  
  bsym  : t1  $\beta\eta\text{-}\equiv$  t2  $\rightarrow$  t2  $\beta\eta\text{-}\equiv$  t1  
  btrans : t1  $\beta\eta\text{-}\equiv$  t2  $\rightarrow$  t2  $\beta\eta\text{-}\equiv$  t3  $\rightarrow$  t1  $\beta\eta\text{-}\equiv$  t3  
  cong $\Lambda$  : t1  $\beta\eta\text{-}\equiv$  t2  $\rightarrow$   $\Lambda$  t1  $\beta\eta\text{-}\equiv$   $\Lambda$  t2  
  congApp  : t1  $\beta\eta\text{-}\equiv$  t2  $\rightarrow$  u1  $\beta\eta\text{-}\equiv$  u2  $\rightarrow$  app t1 u1  $\beta\eta\text{-}\equiv$  app t2 u2  
  beta     : app ( $\Lambda$  t) u  $\beta\eta\text{-}\equiv$  subst t vz u  
  eta     :  $\Lambda$  (app (wkTm vz t) (var vz))  $\beta\eta\text{-}\equiv$  t
```

The subset of normal forms

β -short η -long normal forms:

mutual

data Nf : Con \rightarrow Ty \rightarrow Set **where**

λn : Nf (Γ, σ) $\tau \rightarrow$ Nf Γ $(\sigma \Rightarrow \tau)$

ne : Ne Γ $\circ \rightarrow$ Nf Γ \circ

data Ne : Con \rightarrow Ty \rightarrow Set **where**

$\rightarrow, -$: Var Γ $\sigma \rightarrow$ Sp Γ σ $\tau \rightarrow$ Ne Γ τ

data Sp : Con \rightarrow Ty \rightarrow Ty \rightarrow Set **where**

ε : Sp Γ σ σ

$\rightarrow, -$: Nf Γ $\tau \rightarrow$ Sp Γ σ $\rho \rightarrow$ Sp Γ $(\tau \Rightarrow \sigma)$ ρ

Examples

Neutral terms:

- $x u_1 u_2 u_3 u_4 : \text{Ne } \Gamma \circ$
- $x : \text{Var } \Gamma (\sigma_1 \Rightarrow \sigma_2 \Rightarrow \sigma_3 \Rightarrow \sigma_4 \Rightarrow \circ)$
- $u_i : \text{Nf } \Gamma \sigma_i$

Examples:

- $\lambda n x^\circ. x$
- $\lambda n x^\circ \Rightarrow^\circ. (\lambda n u^\circ. x u)$

Terms and normal forms

Canonical injection from normal forms to terms:

- $[_] : \text{Nf } \Gamma \sigma \rightarrow \text{Tm } \Gamma \sigma$

Normalization function from terms to normal forms:

- $\text{nf} : \text{Tm } \Gamma \sigma \rightarrow \text{Nf } \Gamma \sigma$
- using hereditary substitutions

Outline

- 1 Calculus
- 2 Normalization
- 3 Technical details
- 4 Conclusion

η -expansion

mutual

$nvar : \text{forall } \{\sigma\} \rightarrow \text{Var } \Gamma \sigma \rightarrow \text{Nf } \Gamma \sigma$

$nvar \{\sigma\} x = ne2nf \{\sigma\} (x, \varepsilon)$

$ne2nf : \text{forall } \{\sigma\} \rightarrow \text{Ne } \Gamma \sigma \rightarrow \text{Nf } \Gamma \sigma$

$ne2nf \{\circ\} xns = ne xns$

$ne2nf \{\sigma \Rightarrow \tau\} (x, ns) =$

$\lambda n (ne2nf \{\tau\} (vs x, appSp (wkSp vz ns) (nvar \{\sigma\} vz)))$

η -expansion

mutual

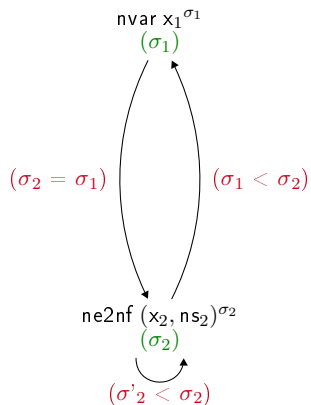
$\text{nvar} : \text{forall } \{\sigma\} \rightarrow \text{Var } \Gamma \sigma \rightarrow \text{Nf } \Gamma \sigma$
 $\text{nvar } \{\sigma\} x = \text{ne2nf } \{\sigma\} (x, \varepsilon)$

$\text{ne2nf} : \text{forall } \{\sigma\} \rightarrow \text{Ne } \Gamma \sigma \rightarrow \text{Nf } \Gamma \sigma$

$\text{ne2nf } \{\circ\} xns = \text{ne } xns$

$\text{ne2nf } \{\sigma \Rightarrow \tau\} (x, ns) =$

$\lambda n (\text{ne2nf } \{\tau\} (\text{vs } x, \text{appSp } (\text{wkSp } \text{vz } ns) (\text{nvar } \{\sigma\} \text{vz})))$



β -reduction

$\text{napp} : \text{Nf } \Gamma (\sigma \Rightarrow \tau) \rightarrow \text{Nf } \Gamma \sigma \rightarrow \text{Nf } \Gamma \tau$
 $\text{napp } t \ u = ?$

β -reduction
$$\text{napp} : \text{Nf } \Gamma (\sigma \Rightarrow \tau) \rightarrow \text{Nf } \Gamma \sigma \rightarrow \text{Nf } \Gamma \tau$$

$$\text{napp } (\lambda n \ t) \ u = ?$$

β -reduction
$$\text{napp} : \text{Nf } \Gamma (\sigma \Rightarrow \tau) \rightarrow \text{Nf } \Gamma \sigma \rightarrow \text{Nf } \Gamma \tau$$

$$\text{napp } (\lambda n \ t) \ u = t \ [vz := u]$$

β -reduction
$$\text{napp} : \text{Nf } \Gamma (\sigma \Rightarrow \tau) \rightarrow \text{Nf } \Gamma \sigma \rightarrow \text{Nf } \Gamma \tau$$

$$\text{napp } (\lambda n \ t) \ u = t [vz := u]$$

$$- [_ := _] : \text{Nf } \Gamma \tau \rightarrow (x : \text{Var } \Gamma \sigma) \rightarrow \text{Nf } (\Gamma - x) \sigma \rightarrow \text{Nf } (\Gamma - x) \tau$$

β -reduction
$$\text{napp} : \text{Nf } \Gamma (\sigma \Rightarrow \tau) \rightarrow \text{Nf } \Gamma \sigma \rightarrow \text{Nf } \Gamma \tau$$

$$\text{napp } (\lambda n \ t) \ u = t [vz := u]$$

$$_ [_ := _] : \text{Nf } \Gamma \tau \rightarrow (x : \text{Var } \Gamma \sigma) \rightarrow \text{Nf } (\Gamma - x) \sigma \rightarrow \text{Nf } (\Gamma - x) \tau$$

$$(\lambda n \ t) [x := u] = \lambda n \ (t [(vs \ x) := (\text{wkNf } vz \ u)])$$

$$(\text{ne } (y, \text{ts})) [x := u] \text{ with eq } x \ y$$

$$(\text{ne } (y, \text{ts})) [x := u] \mid \text{false} = \text{ne } (y, \text{ts} < x := u >)$$

$$(\text{ne } (x, \text{ts})) [x := u] \mid \text{true} = u \diamond (\text{ts} < x := u >)$$

β -reduction
$$\text{napp} : \text{Nf } \Gamma (\sigma \Rightarrow \tau) \rightarrow \text{Nf } \Gamma \sigma \rightarrow \text{Nf } \Gamma \tau$$

$$\text{napp } (\lambda n \ t) \ u = t [vz := u]$$

$$_ [_ := _] : \text{Nf } \Gamma \tau \rightarrow (x : \text{Var } \Gamma \sigma) \rightarrow \text{Nf } (\Gamma - x) \sigma \rightarrow \text{Nf } (\Gamma - x) \tau$$

$$(\lambda n \ t) [x := u] = \lambda n \ (t [(vs \ x) := (wkNf \ vz \ u)])$$

$$(ne \ (y, ts)) [x := u] \text{ with } eq \ x \ y$$

$$(ne \ (y, ts)) [x := u] \mid \text{false} = ne \ (y, ts < x := u >)$$

$$(ne \ (x, ts)) [x := u] \mid \text{true} = u \diamond (ts < x := u >)$$

$$_ \diamond _ : \text{Nf } \Gamma \sigma \rightarrow \text{Sp } \Gamma \sigma \tau \rightarrow \text{Nf } \Gamma \tau$$

$$t \diamond \varepsilon = t$$

$$t \diamond (u, us) = (\text{napp } t \ u) \diamond us$$

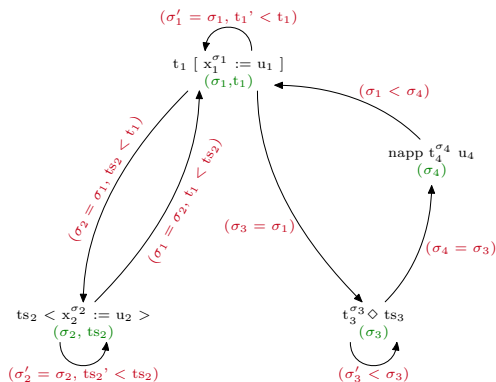
β -reduction
$$\text{napp } (\lambda n t) u = t [vz := u]$$

$$\begin{aligned} (\lambda n t) [x := u] &= \lambda n (t [(vs x) := (\text{wkNf } vz u)]) \\ (\text{ne } (y, ts)) [x := u] &\text{ with eq } x y \\ (\text{ne } (y, ts)) [x := u] \mid \text{false} &= \text{ne } (y, ts < x := u >) \\ (\text{ne } (x, ts)) [x := u] \mid \text{true} &= u \diamond (ts < x := u >) \end{aligned}$$

$$t \diamond \varepsilon = t$$

$$t \diamond (u, us) = (\text{napp } t u) \diamond us$$

$$\varepsilon < x := u > = \varepsilon$$

$$(t, ts) < x := u > = (t [x := u]), (ts < x := u >)$$


The normalization function

$$\text{nf} : \text{Tm } \Gamma \sigma \rightarrow \text{Nf } \Gamma \sigma$$

$$\text{nf} (\text{var } x) = \text{nvar } x$$

$$\text{nf} (\Lambda t) = \lambda n (\text{nf } t)$$

$$\text{nf} (\text{app } t u) = \text{napp} (\text{nf } t) (\text{nf } u)$$

Decidability of $\beta\eta\equiv$

Two properties:

- soundness : $\{t \ u : \text{Tm } \Gamma \ \sigma\} \rightarrow t \ \beta\eta\equiv \ u \rightarrow \text{nf } t \equiv \text{nf } u$
- completeness : $(t : \text{Tm } \Gamma \ \sigma) \rightarrow [\text{nf } t] \ \beta\eta\equiv \ t$

Consequences

- $\{t \ u : \text{Tm } \Gamma \ \sigma\} \rightarrow t \ \beta\eta\equiv \ u \leftrightarrow \text{nf } t \equiv \text{nf } u$
- normalization and checking the equality of normal forms use only first order primitive recursion
- $\beta\eta\equiv$ is primitive recursive

Outline

- 1 Calculus
- 2 Normalization
- 3 Technical details
- 4 Conclusion

Very confusing!

Example

- $\lambda f. \lambda z. f ((\lambda x. f x) z)$
- $\lambda \lambda 1 ((\lambda 2 0) 0)$

Very confusing!

Example

- $\lambda f.\lambda z.f ((\lambda x.f x) z)$
- $\lambda\lambda 1 ((\lambda 2 0) 0)$
- but no α -renaming, a unique representation for any closed term, nice presentation of well-typed terms...

Very confusing!

Example

- $\lambda f.\lambda z.f ((\lambda x.f x) z)$
- $\lambda\lambda 1 ((\lambda 2 0) 0)$
- but no α -renaming, a unique representation for any closed term, nice presentation of well-typed terms...
- equality?

A nice approach to equality

$\text{eq} : (x : \text{Var } \Gamma \sigma) \rightarrow (y : \text{Var } \Gamma \tau) \rightarrow \text{Bool}$

$\text{eq } \text{vz } \text{vz} = \text{true}$

$\text{eq } \text{vz } (\text{vs } x) = \text{false}$

$\text{eq } (\text{vs } x) \text{vz} = \text{false}$

$\text{eq } (\text{vs } x) (\text{vs } y) = \text{eq } x y$

A nice approach to equality

$\text{eq} : (x : \text{Var } \Gamma \sigma) \rightarrow (y : \text{Var } \Gamma \tau) \rightarrow \text{Bool}$

$\text{eq } \text{vz } \text{vz} = \text{true}$

$\text{eq } \text{vz } (\text{vs } x) = \text{false}$

$\text{eq } (\text{vs } x) \text{vz} = \text{false}$

$\text{eq } (\text{vs } x) (\text{vs } y) = \text{eq } x y$

data $\text{EqV} : \text{Var } \Gamma \sigma \rightarrow \text{Var } \Gamma \tau \rightarrow \text{Set}$ **where**

$\text{same} : \{x : \text{Var } \Gamma \sigma\} \rightarrow \text{EqV } x x$

$\text{diff} : (x : \text{Var } \Gamma \sigma) \rightarrow (y : \text{Var } (\Gamma - x) \tau) \rightarrow \text{EqV } x (\text{wkv } x y)$

$\Gamma = [\sigma_1; \sigma; \sigma_2; \sigma_3; \tau; \sigma_4]$

A nice approach to equality

$\text{eq} : (x : \text{Var } \Gamma \sigma) \rightarrow (y : \text{Var } \Gamma \tau) \rightarrow \text{EqV } x \ y$

$\text{eq } \text{vz } \text{vz} = \text{same}$

$\text{eq } \text{vz } (\text{vs } x) = \text{diff } \text{vz } x$

$\text{eq } (\text{vs } x) \text{vz} = \text{diff } (\text{vs } x) \text{vz}$

$\text{eq } (\text{vs } x) (\text{vs } y) \textbf{with } \text{eq } x \ y$

$\text{eq } (\text{vs } x) (\text{vs } .x) \mid \text{same} = \text{same}$

$\text{eq } (\text{vs } .x) (\text{vs } .(\text{wkv } x \ y)) \mid (\text{diff } x \ y) = \text{diff } (\text{vs } x) (\text{vs } y)$

data EqV : Var Γ σ \rightarrow Var Γ τ \rightarrow Set **where**

same : {x : Var Γ σ } \rightarrow EqV x x

diff : (x : Var Γ σ) \rightarrow (y : Var (Γ - x) τ) \rightarrow EqV x (wkv x y)

Application

Substitution for normal forms

$$\begin{aligned} & _ [_ := _] : \text{Nf } \Gamma \tau \rightarrow (x : \text{Var } \Gamma \sigma) \rightarrow \text{Nf } (\Gamma - x) \sigma \rightarrow \text{Nf } (\Gamma - x) \tau \\ & (\lambda n \ t) [x := u] = \lambda n \ (t [(vs \ x) := (wkNf \ vz \ u)]) \\ & (ne \ (y, \ ts)) [x := u] \text{ with eq } x \ y \\ & (ne \ (y, \ ts)) [x := u] \mid \text{false} = ne \ (y, \ ts < x := u >) \\ & (ne \ (x, \ ts)) [x := u] \mid \text{true} = u \diamond (ts < x := u >) \end{aligned}$$

Application

Substitution for normal forms

~~$$\begin{aligned}
 & _ [_ := _] : \text{Nf } \Gamma \tau \rightarrow (x : \text{Var } \Gamma \sigma) \rightarrow \text{Nf } (\Gamma - x) \sigma \rightarrow \text{Nf } (\Gamma - x) \tau \\
 & (\lambda n \ t) [x := u] = \lambda n \ (t [(vs \ x) := (wkNf \ vz \ u)]) \\
 & (ne \ (y, \ ts)) [x := u] \text{ with eq } x \ y \\
 & (ne \ (y, \ ts)) [x := u] \mid \text{false} = ne \ (y, \ ts < x := u >) \\
 & (ne \ (x, \ ts)) [x := u] \mid \text{true} = u \diamond (ts < x := u >)
 \end{aligned}$$~~

$$\begin{aligned}
 & _ [_ := _] : \text{Nf } \Gamma \tau \rightarrow (x : \text{Var } \Gamma \sigma) \rightarrow \text{Nf } (\Gamma - x) \sigma \rightarrow \text{Nf } (\Gamma - x) \tau \\
 & (\lambda n \ t) [x := u] = \lambda n \ (t [(vs \ x) := (wkNf \ vz \ u)]) \\
 & (ne \ (y, \ ts)) [x := u] \text{ with eq } x \ y \\
 & (ne \ (.(wkv \ x \ y'), \ ts)) [x := u] \mid \text{diff } x \ y' = ne \ (y', \ ts < x := u >) \\
 & (ne \ (x, \ ts)) [x := u] \mid \text{same} = u \diamond (ts < x := u >)
 \end{aligned}$$

Outline

- 1 Calculus
- 2 Normalization
- 3 Technical details
- 4 Conclusion

Conclusion

- A normalizer for the simply-typed λ -calculus in Total Type Theory
- Decidability of $\beta\eta$ - \equiv
- Does not easily scale-up

Agda

Pros

- Manipulation of dependent types
- Termination checker
- Write programs

Cons

- Write proofs

Thanks

Thank you for your attention!

Any questions?