Hereditary Substitutions for Simple Types, Formalized

Chantal Keller Thorsten Altenkirch

INRIA – LIX – ENS de Lyon – University of Nottingham

September, 25th 2010

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

Simply-typed -calculus

Types:

```
data Ty : Set where
    ∘   :  Ty
  _⇒_   :  Ty  →  Ty  →  Ty
```

Terms:

```
data Var :                Set where
    vz   :  Var
    vs   :  Var                →  Var
data Tm :                 Set where
   var   :  Var                →  Tm
     Λ   :  Tm                 →  Tm
   app   :  Tm                 →  Tm          →  Tm
```

Simply-typed -calculus

Types:

```
data Ty : Set where
  ∘   :  Ty
  _⇒_ :  Ty → Ty → Ty
```

Terms:

```
data Var : Con → Ty → Set where
  vz  :  Var (Γ , σ) σ
  vs  :  Var Γ σ → Var (Γ , τ) σ
data Tm : Con → Ty → Set where
  var :  Var Γ σ → Tm Γ σ
  Λ   :  Tm (Γ , σ) τ → Tm Γ (σ ⇒ τ)
  app :  Tm Γ (σ ⇒ τ) → Tm Γ σ → Tm Γ τ
```

Simply-typed -calculus

What can we do with this syntax?

Types:

```
data Ty : Set where
  ∘     : Ty
  _⇒_   : Ty → Ty → Ty
```

Terms:

```
data Var : Con → Ty → Set where
  vz   : Var (Γ , σ) σ
  vs   : Var Γ σ → Var (Γ , τ) σ
data Tm : Con → Ty → Set where
  var  : Var Γ σ → Tm Γ σ
  Λ    : Tm (Γ , σ) τ → Tm Γ (σ ⇒ τ)
  app  : Tm Γ (σ ⇒ τ) → Tm Γ σ → Tm Γ τ
```

Weakening

Minus:

"Γ - x" means "the context Γ from which the variable x is removed"

Variable weakening:

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

Term weakening:

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

Substitution

Term substitution:

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

subst t x u

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

-convertibility

The subset of normal forms

-s ort -long normal forms:

```
mutual
  data Nf : Con → Ty → Set where
    λn : Nf (Γ, σ) τ → Nf Γ (σ ⇒ τ)
    ne : Ne Γ ∘ → Nf Γ ∘

  data Ne : Con → Ty → Set where
    _,_ : Var Γ σ → Sp Γ σ τ → Ne Γ τ

  data Sp : Con → Ty → Ty → Set where
    ε : Sp Γ σ σ
    _,_ : Nf Γ τ → Sp Γ σ ρ → Sp Γ (τ ⇒ σ) ρ
```

Examples

Neutral terms:

- $\mathsf{x}\ \mathsf{u}_1\ \mathsf{u}_2\ \mathsf{u}_3\ \mathsf{u}_4\ :\ \mathsf{Ne}\ \Gamma\ \circ$
- $\mathsf{x}\ :\ \mathsf{Var}\ \Gamma\ (\sigma_1 \Rightarrow \sigma_2 \Rightarrow \sigma_3 \Rightarrow \sigma_4 \Rightarrow \circ)$
- $\mathsf{u}_i : \mathsf{Nf}\ \Gamma\ \sigma_i$

Examples:

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

Terms and normal forms

Canonical injection from normal forms to terms:

- $\lceil \_ \rceil : \mathsf{Nf}\ \Gamma\ \sigma \to \mathsf{Tm}\ \Gamma\ \sigma$

Normalization function from terms to normal forms:

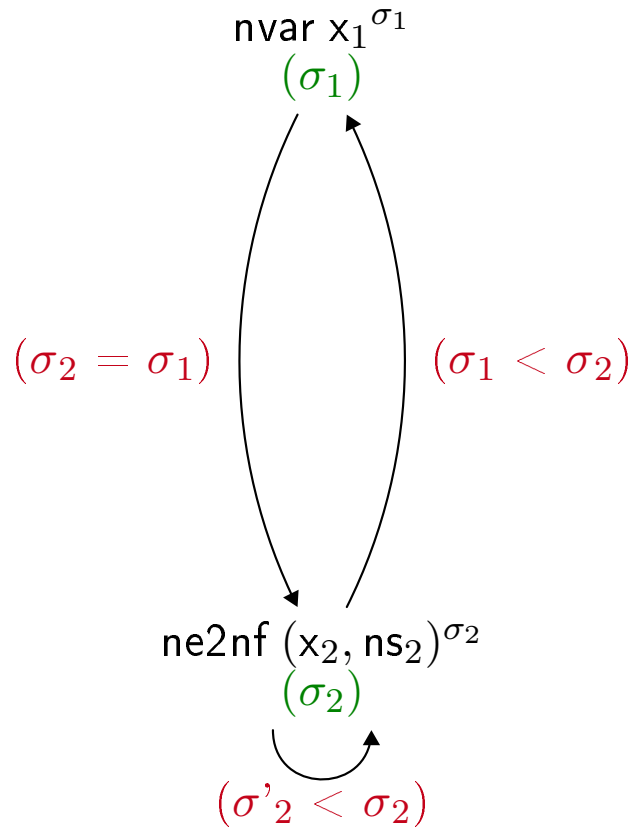
- $\mathsf{nf} : \mathsf{Tm}\ \Gamma\ \sigma \to \mathsf{Nf}\ \Gamma\ \sigma$
- using hereditary substitutions

-expansion

```
mutual
  nvar : forall {σ} → Var Γ σ → Nf Γ σ
  nvar {σ} x = ne2nf {σ} (x, ε)

  ne2nf : forall {σ} → Ne Γ σ → Nf Γ σ
  ne2nf {○} xns = ne xns
  ne2nf {σ ⇒ τ} (x, ns) =
λn (ne2nf {τ} (vs x, appSp (wkSp vz ns) (nvar {σ} vz)))
```

-expansion



mutual

nvar : **forall** $\{\sigma\} \to$ Var Γ $\sigma \to$ Nf Γ σ
nvar $\{\sigma\}$ x = ne2nf $\{\sigma\}$ (x, ε)

ne2nf : **forall** $\{\sigma\} \to$ Ne Γ $\sigma \to$ Nf Γ σ
ne2nf $\{\circ\}$ xns = ne xns
ne2nf $\{\sigma \Rightarrow \tau\}$ (x, ns) =
λn (ne2nf $\{\tau\}$ (vs x, appSp (wkSp vz ns) (nvar $\{\sigma\}$ vz)))

-reduction

napp : Nf Γ $(\sigma \Rightarrow \tau) \to$ Nf Γ $\sigma \to$ Nf Γ τ
napp t u = ?

-reduction

napp : Nf Γ $(\sigma \Rightarrow \tau) \to$ Nf Γ $\sigma \to$ Nf Γ τ
napp $(\lambda$n t) u = ?

-reduction

napp : Nf Γ $(\sigma \Rightarrow \tau) \to$ Nf Γ $\sigma \to$ Nf Γ τ
napp $(\lambda$n t) u $=$ t [vz := u]

-reduction

napp : Nf Γ $(\sigma \Rightarrow \tau) \to$ Nf Γ $\sigma \to$ Nf Γ τ
napp $(\lambda$n t) u $=$ t [vz := u]

_ [_:=_] : Nf Γ $\tau \to$ (x : Var Γ $\sigma) \to$ Nf (Γ - x) $\sigma \to$ Nf (Γ - x) τ

-reduction

```
napp : Nf Γ (σ ⇒ τ) → Nf Γ σ → Nf Γ τ
napp (λn t) u = t [vz := u]

_ [_:=_] : Nf Γ τ → (x : Var Γ σ) → Nf (Γ - x) σ → Nf (Γ - x) τ
(λn t) [x := u] = λn (t [(vs x) := (wkNf vz u)])
(ne (y, ts)) [x := u] with eq x y
(ne (y, ts)) [x := u] | false = ne (y, ts < x := u >)
(ne (x, ts)) [x := u] | true = u ◇ (ts < x := u >)
```

-reduction

```
napp : Nf Γ (σ ⇒ τ) → Nf Γ σ → Nf Γ τ
napp (λn t) u = t [vz := u]

_ [_:=_] : Nf Γ τ → (x : Var Γ σ) → Nf (Γ - x) σ → Nf (Γ - x) τ
(λn t) [x := u] = λn (t [(vs x) := (wkNf vz u)])
(ne (y, ts)) [x := u] with eq x y
(ne (y, ts)) [x := u] | false = ne (y, ts < x := u >)
(ne (x, ts)) [x := u] | true = u ◇ (ts < x := u >)

_◇_ : Nf Γ σ → Sp Γ σ τ → Nf Γ τ
t ◇ ε = t
t ◇ (u, us) = (napp t u) ◇ us
```

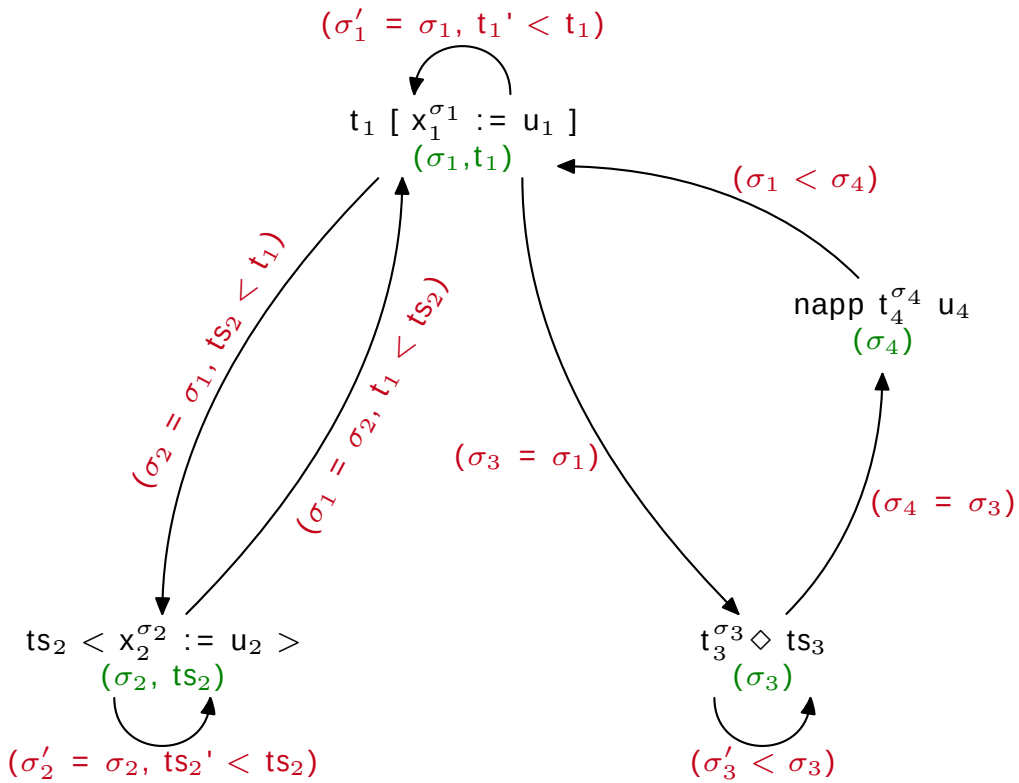
-reduction

napp (n t) u = t [vz := u]

(n t) [x := u] = n (t [(vs x) := (wkNf vz u)])
(ne (y; ts)) [x := u] with eq x y
(ne (y; ts)) [x := u] | false = ne (y; ts < x := u >)
(ne (x; ts)) [x := u] | true = u ◇ (ts < x := u >)

t ◇ " = t
t ◇ (u; us) = (napp t u) ◇ us

" < x := u > = "
(t; ts) < x := u > = (t [x := u]); (ts < x := u >)



The normalization function

$$
\begin{aligned}
&\mathsf{nf} : \mathsf{Tm}\ \Gamma\ \sigma \to \mathsf{Nf}\ \Gamma\ \sigma \\
&\mathsf{nf}\ (\mathsf{var}\ x) = \mathsf{nvar}\ x \\
&\mathsf{nf}\ (\Lambda\ t) = \lambda n\ (\mathsf{nf}\ t) \\
&\mathsf{nf}\ (\mathsf{app}\ t\ u) = \mathsf{napp}\ (\mathsf{nf}\ t)\ (\mathsf{nf}\ u)
\end{aligned}
$$

Decidability of -≡

Two properties:

- soundness : $\{\mathsf{t}\ \mathsf{u} : \mathsf{Tm}\ \Gamma\ \sigma\} \to \mathsf{t}\ \beta\eta\text{-}{\equiv}\ \mathsf{u} \to \mathsf{nf}\ \mathsf{t} \equiv \mathsf{nf}\ \mathsf{u}$
- completeness : $(\mathsf{t} : \mathsf{Tm}\ \Gamma\ \sigma) \to \lceil\ \mathsf{nf}\ \mathsf{t}\ \rceil\ \beta\eta\text{-}{\equiv}\ \mathsf{t}$

Consequences

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

Outline

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

$$\begin{aligned}
&\mathsf{eq} : (\mathsf{x} : \mathsf{Var}\,\Gamma\,\sigma) \to (\mathsf{y} : \mathsf{Var}\,\Gamma\,\tau) \to \mathsf{Bool}\\
&\mathsf{eq}\ \mathsf{vz}\ \mathsf{vz} = \mathsf{true}\\
&\mathsf{eq}\ \mathsf{vz}\ (\mathsf{vs}\ \mathsf{x}) = \mathsf{false}\\
&\mathsf{eq}\ (\mathsf{vs}\ \mathsf{x})\ \mathsf{vz} = \mathsf{false}\\
&\mathsf{eq}\ (\mathsf{vs}\ \mathsf{x})\ (\mathsf{vs}\ \mathsf{y}) = \mathsf{eq}\ \mathsf{x}\ \mathsf{y}
\end{aligned}$$

A nice approach to equality

```
eq : (x : Var Γ σ) → (y : Var Γ τ) → Bool
eq vz vz = true
eq vz (vs x) = false
eq (vs x) vz = false
eq (vs x) (vs y) = eq x y
```

```
data EqV : Var Γ σ → Var Γ τ → Set where
  same : {x : Var Γ σ} → EqV x x
  diff : (x : Var Γ σ) → (y : Var (Γ - x) τ) → EqV x (wkv x y)
Γ = [σ1; σ; σ2; σ3; τ; σ4]
```

A nice approach to equality

```
eq : (x : Var Γ σ) → (y : Var Γ τ) → EqV x y
eq vz vz = same
eq vz (vs x) = diff vz x
eq (vs x) vz = diff (vs x) vz
eq (vs x) (vs y) with eq x y
eq (vs x) (vs .x) | same = same
eq (vs .x) (vs .(wkv x y)) | (diff x y) = diff (vs x) (vs y)
```

```
data EqV : Var Γ σ → Var Γ τ → Set where
  same : {x : Var Γ σ} → EqV x x
  diff : (x : Var Γ σ) → (y : Var (Γ - x) τ) → EqV x (wkv x y)
```

Application

Substitution for normal forms

```
_ [ _ := _ ] : Nf Γ τ → (x : Var Γ σ) → Nf (Γ - x) σ → Nf (Γ - x) τ
(λn t) [x := u] = λn (t [(vs x) := (wkNf vz u)])
(ne (y, ts)) [x := u] with eq x y
(ne (y, ts)) [x := u] | false = ne (y, ts < x := u >)
(ne (x, ts)) [x := u] | true = u ◇ (ts < x := u >)
```

Application

Substitution for normal forms

~~_ [_:=_] : Nf Γ τ → (x : Var Γ σ) → Nf (Γ - x) σ → Nf (Γ - x) τ~~
~~(λn t) [x := u] = λn (t [(vs x) := (wkNf vz u)])~~
~~(ne (y, ts)) [x := u] **with** eq x y~~
~~(ne (y, ts)) [x := u] | false = ne (y, ts < x := u >)~~
~~(ne (x, ts)) [x := u] | true = u ◇ (ts < x := u >)~~

_ [_:=_] : Nf Γ τ → (x : Var Γ σ) → Nf (Γ - x) σ → Nf (Γ - x) τ
(λn t) [x := u] = λn (t [(vs x) := (wkNf vz u)])
(ne (y, ts)) [x := u] **with** eq x y
(ne (.(wkv x y'), ts)) [.x := u] | diff x y' = ne (y', ts < x := u >)
(ne (x, ts)) [.x := u] | same = u ◇ (ts < x := u >)

Outline

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?