The Simply Typed λ-Calculus

(In Agda)

Jonathan Prieto-Cubides

Master in Applied Mathematics
Logic and Computation Group
Universidad EAFIT
Medellín, Colombia

1th June 2017

Contents

Definition

The set of λ-terms denoted by Λ is built up from a set of variables V using application and (function) abstraction.

$$
\begin{aligned}
x \in V &\Rightarrow x \in \Lambda, \\
M \in \Lambda, x \in V &\Rightarrow (\lambda x.M) \in \Lambda, \\
M, N \in \Lambda &\Rightarrow (M\,N) \in \Lambda, \\
M, N \in \Lambda, x \in V &\Rightarrow ((\lambda x.M)\,N) \in \Lambda.
\end{aligned}
$$

Definition

The set of λ-terms denoted by Λ is built up from a set of variables V using application and (function) abstraction.

$$
\begin{aligned}
x \in V &\Rightarrow x \in \Lambda, \\
M \in \Lambda, x \in V &\Rightarrow (\lambda x.M) \in \Lambda, \\
M, N \in \Lambda &\Rightarrow (M\,N) \in \Lambda, \\
M, N \in \Lambda, x \in V &\Rightarrow ((\lambda x.M)\;N) \in \Lambda.
\end{aligned}
$$

```
Name : Set
Name = String

data Expr : Set where
  var : Name → Expr
  lam : Name → Expr → Expr
  _•_ : Expr → Expr → Expr
```

Definition

$$\frac{M : \sigma \longrightarrow \tau \qquad N : \sigma}{M N : \tau}$$

$$\cfrac{\cfrac{\cfrac{[x : \sigma]^1}{\vdots}}{M : \tau}}{\lambda x . M : \sigma \to \tau} \; 6\_1$$

Problem	**Question**
Type-checking	Given M and τ, $\Gamma \vdash M : \tau$?
Typability	Given M does exists a σ, $\Gamma \vdash M : \sigma$?
Inhabitation	Given τ, does exists an M such that $\Gamma \vdash M : \sigma$?

Theorem

- *It is decidable whether a term is typable in $\lambda \to$.*
- *If a term M is typable in $\lambda \to$, then M has a principal type scheme, i.e. a type σ such that every possible type for M is a subsitution instance of σ. Moreover σ is computable from M.*

Theorem

Type checking for $\lambda \to$ is decidable.

```
module Typing (U : Set) where

data Type : Set where
  base : U     → Type
  _→_  : Type → Type → Type
```

```
module Syntax (Type : Set) where

open import Data.String

Name : Set
Name = String

-- Type Judgments (x : A).
data Formal : Set where
  _:_ : Name → Type → Formal

data Expr : Set where
  var : Name   → Expr
  lam : Formal → Expr → Expr
  _•_ : Expr   → Expr → Expr
```

module Examples (Type : Set) where

```
open import Syntax Type

postulate A : Type

x = var "x"
y = var "y"
z = var "z"

-- I, K, S : Expr

I = lam ("x" : A) x                       -- λx.x, x ∈ A
K = lam ("x" : A) (lam ("y" : A) x)       -- λxy.x, x,y ∈ A
S =
  lam ("x" : A)
    (lam ("y" : A)
      (lam ("z" : A)
        ((x • z) • (y • z))))             -- λxyz.xz(yz), x,y,z ∈ A
```

```
data Expr (n : ℕ) : Set where
  var : Fin n  → Expr n
  lam : Type   → Expr (suc n) → Expr n
  _•_ : Expr n → Expr n       → Expr n

Binder : ℕ → Set
Binder = Vec Name

data _⊢_⇝_ : ∀ {n} → Binder n → S.Expr → Expr n → Set where
  var-zero : ∀ {n x} {Γ : Binder n}
           → Γ , x ⊢ var x ⇝ var (# 0)

  var-suc  : ∀ {n x y k} {Γ : Binder n} {p : False (x ≟ y)}
           → Γ ⊢ var x ⇝ var k
           → Γ , y ⊢ var x ⇝ var (suc k)

  lam      : ∀ {n x τ t t′} {Γ : Binder n}
           → Γ , x ⊢ t ⇝ t′
           → Γ ⊢ lam (x : τ) t ⇝ lam τ t′

  _•_      : ∀ {n t₁ t₁′ t₂ t₂′} {Γ : Binder n}
           → Γ ⊢ t₁ ⇝ t₁′
           → Γ ⊢ t₂ ⇝ t₂′
           → Γ ⊢ t₁ • t₂ ⇝ t₁′ • t₂′
```

```
∅ : Binder 0
∅ = []

Γ : Binder 2
Γ = "x" ∷ "y" ∷ []

e1 : "x" ∷ "y" ∷ [] ⊢ var "x" ⇝ var (# 0)
e1 = var-zero

I : [] ⊢ lam ("x" : A) (var "x")
       ⇝ lam A (var (# 0))
I = lam var-zero

K : [] ⊢ lam ("x" : A) (lam ("y" : A) (var "x"))
       ⇝ lam A (lam A (var (# 1)))
K = lam (lam (var-suc var-zero))

K₂ : [] ⊢ lam ("x" : A) (lam ("y" : A) (var "y"))
        ⇝ lam A (lam A (var (# 0)))
K₂ = lam (lam var-zero)

P : Γ ⊢ lam ("x" : A) (lam ("y" : A) (lam ("z" : A) (var "x")))
      ⇝ lam A (lam A (lam A (var (# 2))))
P = {!!}
```

```
open import Bound Type hiding (_,_)

Ctxt : ℕ → Set
Ctxt = Vec Type

_,_ : ∀ {n} → Ctxt n → Type → Ctxt (suc n)
Γ , x = x ∷ Γ

data _⊢_:_ : ∀ {n} → Ctxt n → Expr n → Type → Set where
  tVar : ∀ {n Γ} {x : Fin n}
       → Γ ⊢ var x : lookup x Γ

  tLam : ∀ {n} {Γ : Ctxt n} {t} {τ τ′}
       → Γ , τ ⊢ t : τ′
       → Γ ⊢ lam τ t : τ ⟶ τ′

  _•_  : ∀ {n} {Γ : Ctxt n} {t₁ t₂} {τ τ′}
       → Γ ⊢ t₁ : τ ⟶ τ′
       → Γ ⊢ t₂ : τ
       → Γ ⊢ t₁ • t₂ : τ′
```