

# The Simply Typed $\lambda$ -Calculus

(In Agda)

Jonathan Prieto-Cubides

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

1th June 2017



```
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 --  $\lambda x.x, x \in A$ 
```

```
K = lam ("x" : A) (lam ("y" : A) x) --  $\lambda xy.x, x,y \in A$ 
```

```
S =
```

```
  lam ("x" : A)
```

```
    (lam ("y" : A)
```

```
      (lam ("z" : A)
```

```
        ((x • z) • (y • z))))
```

```
--  $\lambda xyz.xz(yz), x,y,z \in A$ 
```

```
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'
```

$\emptyset$  : Binder 0

$\emptyset = []$

$\Gamma$  : Binder 2

$\Gamma = "x" :: "y" :: []$

$e_1 : "x" :: "y" :: [] \vdash \text{var } "x" \rightarrow \text{var } (\# 0)$

$e_1 = \text{var-zero}$

$I : [] \vdash \text{lam } ("x" : A) (\text{var } "x")$

$\rightarrow \text{lam } A (\text{var } (\# 0))$

$I = \text{lam var-zero}$

$K : [] \vdash \text{lam } ("x" : A) (\text{lam } ("y" : A) (\text{var } "x"))$

$\rightarrow \text{lam } A (\text{lam } A (\text{var } (\# 1)))$

$K = \text{lam } (\text{lam } (\text{var-suc var-zero}))$

$K_2 : [] \vdash \text{lam } ("x" : A) (\text{lam } ("y" : A) (\text{var } "y"))$

$\rightarrow \text{lam } A (\text{lam } A (\text{var } (\# 0)))$

$K_2 = \text{lam } (\text{lam } \text{var-zero})$

$P : \Gamma \vdash \text{lam } ("x" : A) (\text{lam } ("y" : A) (\text{lam } ("z" : A) (\text{var } "x")))$

$\rightarrow \text{lam } A (\text{lam } A (\text{lam } A (\text{var } (\# 2))))$

$P = \{\!\!\{!\!\}$

## module Typing (U : Set) where

```
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'
```