Tutorial Implementation of a Dependently Typed Lambda Calculus

Tobias Hoffmann

What even are Dependent Types?

- The Function type `t → t'` is extended
- Return type can depend on argument type
- More than just polymorphism

```
cons_fixed :: Bool → List Bool → List Bool

cons_polymorphic :: a → List a → List a

cons_dependent :: (n :: Nat) → a → Vec a n → Vec a (n + 1)
```

Dependent Types in Practice

- Proof Assistant
  - Agda
  - Coq
- General Functional Programming
  - Idris

Proving things with Dependent Types

Proving associativity of addition on natural numbers in Agda:

```
data Nat : Set where
  zero : Nat
  suc  : Nat → Nat

_+_ : Nat → Nat → Nat
zero + y = y
(suc x) + y = suc (x + y)

data _==_ (x : Nat) → Set where
  refl : x == x

assoc : (x : Nat) → (y : Nat) → (z : Nat) → (x + y) + z == x + (y + z)
assoc x y z = ?
```

Programming with Dependent Types

Known-length vectors and the functions `append` *and* `head` *on them:*

```
data Vec : Set → Nat → Set where
    nil : {a : Set} → Vec a 0
    _::_ : {a : Set} → {n : Nat} → a → Vec a n → Vec a (1 + n)

append : {a : Set} → {n m : Nat} → Vec a n → Vec a m → Vec a (n + m)
append nil v' = v'
append (x :: v) v' = x :: (append v v')

head : {a : Set} → {n : Nat} → {1 ⩽ n} → Vec a n → a
head (x :: v) = x
```

...

Annotated Term

$$\frac{\Gamma \vdash \tau :: * \quad \Gamma \vdash e ::_{\downarrow} \tau}{\Gamma \vdash (e :: \tau) ::_{\uparrow} \tau}$$

```
typeInfer i g (Ann e t) =
  do
    kindCheck g t Star
    typeCheck i g e t
    return t
```

$$\frac{\Gamma \vdash \rho ::_{\downarrow} * \quad \rho \Downarrow \tau \quad \Gamma \vdash e ::_{\downarrow} \tau}{\Gamma \vdash (e :: \rho) ::_{\uparrow} \tau}$$

```
typeInfer i g (Ann e r) =
  do
    typeCheck i g r VStar
    let t = evalCheck [] r
    typeCheck i g e t
    return t
```

Conclusion

- Dependent types aren't as scary as they seem