Chapter 23: Universal Types

System F (polymorphic lambda calculus)

Power of System F

Properties (Soundness, decidability, paramertricity)

Abstraction

```
doubleNat = λf:Nat→Nat. λx:Nat. f (f x);
doubleRcd = λf:{l:Bool}→{l:Bool}. λx:{l:Bool}. f (f x);
doubleFun = λf:(Nat→Nat)→(Nat→Nat). λx:Nat→Nat. f (f x);
```

Can we do abstraction over types so that we can apply to different types?

```
double = λX. λf:X→X. λx:X. f (f x)
```

Polymorphism

- Parametric polymorphism
  - $\lambda x{:}\ T.\ x : T \to T$
- Ad-hoc polymorphism (overloading)
  - 1 + 2
  - 1.0 + 2.0
  - "we " + "you"

System F

- First discovered by Jean-Yves Girard (1972)
- Independently developed by John Reynolds (1974) as polymorphic lambda calculus (or second order lambda calculus)
- A natural extension of λ→ with a new form of abstract and application over types:

$$\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \lambda X.t_2 : \forall X.T_2}$$

$$\frac{\Gamma \vdash t_1 : \forall X.T_{12}}{\Gamma \vdash t_1\ [T_2] : [X \mapsto T_2]T_{12}}$$

Syntax and Evaluation

Syntax

$\mathtt{t}$	::=		*terms:*
		$\mathtt{x}$	*variable*
		$\lambda\mathtt{x}{:}\mathtt{T}.\mathtt{t}$	*abstraction*
		$\mathtt{t}\ \mathtt{t}$	*application*
		$\lambda\mathtt{X}.\mathtt{t}$	*type abstraction*
		$\mathtt{t}\ [\mathtt{T}]$	*type application*
$\mathtt{v}$	::=		*values:*
		$\lambda\mathtt{x}{:}\mathtt{T}.\mathtt{t}$	*abstraction value*
		$\lambda\mathtt{X}.\mathtt{t}$	*type abstraction value*

Evaluation $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-APP1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-APP2)}$$

$$(\lambda\mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-APPABS)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ [\mathtt{T}_2] \longrightarrow \mathtt{t}_1'\ [\mathtt{T}_2]} \quad \text{(E-TAPP)}$$

$$(\lambda\mathtt{X}.\mathtt{t}_{12})\ [\mathtt{T}_2] \longrightarrow [\mathtt{X} \mapsto \mathtt{T}_2]\mathtt{t}_{12} \quad \text{(E-TAPPTABS)}$$

Types and Type Context

T	$::=$	*types:*
	X	*type variable*
	$T \to T$	*type of functions*
	$\forall X.T$	*universal type*
Γ	$::=$	*contexts:*
	$\varnothing$	*empty context*
	$\Gamma, x{:}T$	*term variable binding*
	Γ, X	*type variable binding*

Typing

Typing $\boxed{\Gamma \vdash t : T}$

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1 . t_2 : T_1 \to T_2} \qquad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-APP)}$$

$$\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \lambda X . t_2 : \forall X . T_2} \qquad \text{(T-TABS)}$$

$$\frac{\Gamma \vdash t_1 : \forall X . T_{12}}{\Gamma \vdash t_1\ [T_2] : [X \mapsto T_2] T_{12}} \qquad \text{(T-TAPP)}$$

Ex.: Defining Polymorphic Functions

- id = λX. λx:X. x
  - id : ∀X. X → X
  - id [Nat] 0 ➔ 0
- double = λX. λf:X→X. λa:X. f (f a)
  - double : ∀X. (X→X) → X → X
  - double [Nat] (λx:Nat. succ(succ(x))) 3 ➔ 7
- selfApp = λx:∀X.X→X. x [∀X.X→X] x
  - selfApp : (∀X. X→X) → (∀X. X → X)
- quadruple = λX. double [X→X] (double [X]);
  - quadruple : ∀X. (X→X) → X → X

Ex.: Polymorphic Lists

- nil : $\forall X.$ List X
- cons : $\forall X.$ X $\rightarrow$ List X $\rightarrow$ List X
- isnil : $\forall X.$ List X $\rightarrow$ Bool
- head : $\forall X.$ List X $\rightarrow$ X
- tail : $\forall X.$ List X $\rightarrow$ List X

```
map : ∀X. ∀Y. (X→Y) → List X → List Y
map = λX. λY. λf: X→Y.
            (fix (λm: (List X) → (List Y). λl: List X.
            if isnil [X] l then nil [Y]
            else cons [Y] (f (head [X] l)) (m (tail [X] l))))
```

Exercise: Can you write reverse?

Ex.: Church Encoding

- Church encodings can be carried out in System F.

- CBool = ∀X.X→X→X;
  - tru = λX. λt:X. λf:X. t;
  - fls = λX. λt:X. λf:X. f;
  - not = λb:CBool. λX. λt:X. λf:X. b [X] f t;
- !
- CNat = ∀X. (X→X) → X → X
  - c0 = λX. λs:X→X. λz:X. z
  - c1= λX. λs:X→X. λz:X. s z;
  - csucc = λn:CNat. λX. λs:X→X. λz:X. s (n [X] s z)
  - cplus = λm:CNat. λn:CNat. λX. λs:X→X. λz:X.
    m [X] s (n [X] s z)

Ex.: Encoding Lists

- List X = ∀R. (X→R→R) → R → R
  - nil = λX. (λR. λc:X→R→R. λn:R. n) as ∀X. List X
  - cons = λX. λhd:X. λtl:List X.

    (λR. λc:X→R→R. λn:R. c hd (tl [R] c n)) as List X;
  - isnil = λX. λl:List X.

    l [Bool] (λhd:X. λtl:Bool. false) true
  - head = λX. λl:List X.

    l [X] (λhd:X. λtl:X. hd) (diverge [X] unit)
  - sum : List Nat → Nat

    sum = … definition without using fix …?

sum = λl: List Nat. l (λhd: Nat, λtl:Nat. hd + tl) 0
(sum = foldr (λhd: Nat, λtl:Nat. hd + tl) 0)

Ex.: Encoding Pair

- Pair X Y = λR. (X→Y→R) → R;
  - pair : ∀X. ∀Y. X → Y → Pair X Y
  - fst : ∀X. ∀Y. Pair X Y → X
  - snd : ∀X. ∀Y. Pair X Y → Y

pair = λX. λY. λx:X. λy:Y. λR. λp. p x y

fst = λX. λY. λp. p [X] (λx. λy → x)
snd = λX. λY. λp. p [X] (λx. λy → y)

Basic Properties of System F

Very similar to those of the simply typed λ-calculus.

Theorem [Preservation]: If $\Gamma \vdash t : T$ and $t \rightarrow t'$, then $\Gamma \vdash t' : T$.

Theorem [Progress]: If t is a closed, well-typed term, then either t is a value or there is some t' with $t \rightarrow t'$.

Theorem [Normalization]: Well-typed System F terms are normalizing (i.e., the evaluation of every well-typed program terminates).

Erasure and Type Construction

$$\begin{aligned}
\mathit{erase}(\lambda \mathtt{x}{:}\mathtt{T}_1.\ \mathtt{t}_2) &= \lambda \mathtt{x}.\ \mathit{erase}(\mathtt{t}_2) \\
\mathit{erase}(\mathtt{t}_1\ \mathtt{t}_2) &= \mathit{erase}(\mathtt{t}_1)\ \mathit{erase}(\mathtt{t}_2) \\
\mathit{erase}(\lambda \mathtt{X}.\ \mathtt{t}_2) &= \mathit{erase}(\mathtt{t}_2)
\end{aligned}$$

Theorem [Wells, 1994]: It is undecidable whether, given a closed term m of the untyped lambda-calculus, there is some well-typed term t in System F such that erase(t) = m.

Partial Erasure and Type Construction

$$\begin{array}{lcl}
erase_p(\mathtt{x}) & = & \mathtt{x} \\
erase_p(\lambda \mathtt{x}{:}\mathtt{T}_1.\ \mathtt{t}_2) & = & \lambda \mathtt{x}{:}\mathtt{T}_1.\ erase_p(\mathtt{t}_2) \\
erase_p(\mathtt{t}_1\ \mathtt{t}_2) & = & erase_p(\mathtt{t}_1)\ erase_p(\mathtt{t}_2) \\
erase_p(\lambda \mathtt{X}.\ \mathtt{t}_2) & = & \lambda \mathtt{X}.\ erase_p(\mathtt{t}_2) \\
erase_p(\mathtt{t}_1\ [\mathtt{T}_2]) & = & erase_p(\mathtt{t}_1)\ \boxed{[]}
\end{array}$$

Theorem [Boehm 1985, 1989]: It is undecidable whether, given a closed term s in which type applications are marked but the arguments are omitted, there is some well-typed System F term t such that $erase_p(t) = s$.

Type reconstruction is as hard as higher-order unification.
(But many practical algorithms have been developed)

Erasure and Evaluation Order

Keep type abstraction

$$\begin{aligned}
erase_v(\mathtt{x}) &= \mathtt{x} \\
erase_v(\lambda\mathtt{x}{:}\mathtt{T}_1.\ \mathtt{t}_2) &= \lambda\mathtt{x}.\ erase_v(\mathtt{t}_2) \\
erase_v(\mathtt{t}_1\ \mathtt{t}_2) &= erase_v(\mathtt{t}_1)\ erase_v(\mathtt{t}_2) \\
erase_v(\lambda\mathtt{X}.\ \mathtt{t}_2) &= \lambda\_.\ erase_v(\mathtt{t}_2) \\
erase_v(\mathtt{t}_1\ [\mathtt{T}_2]) &= erase_v(\mathtt{t}_1)\ \mathtt{dummyv}
\end{aligned}$$

Theorem: If $erase_v(t) = u$, then either (1) both t and u are normal forms according to their respective evaluation relations, or (2) $t \to t'$ and $u \to u'$, with $erase_v(t') = u'$.

Fragments of System F

- Rank-1 (prenex) polymorphism
  - type variables should not be instantiated with polymorphic types

- Rank-2 polymorphism
  - A type is said to be of rank 2 if no path from its root to a ∀ quantifier passes to the left of 2 or more arrows.

$(\forall X.X\to X)\to \mathrm{Nat}$	OK
$\mathrm{Nat}\to((\forall X.X\to X)\to(\mathrm{Nat}\to\mathrm{Nat}))$	OK
$((\forall X.X\to X)\to \mathrm{Nat})\to \mathrm{Nat}$	X

Type reconstruction for ranks 2 and lower is decidable, and that for rank 3 and higher of System F is undecidable.

Parametricity

- Uniform behavior of polymorphic programs

```
CBool = ∀X.X→X→X;
tru = λX. λt:X. λf:X. t;
fls = λX. λt:X. λf:X. f;
```

(1) Tru and fls are the only two basic inhabitants of Cbool.

(2) Free Theorem:
e.g., for reverse: ∀X. List X -> List X, we have

```
map [X] [Y] f . reverse [List X] = reverse [List Y] . map [X] [Y] f
```

Homework

23.5.1 THEOREM [PRESERVATION]: If $\Gamma \vdash \mathtt{t} : \mathsf{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathsf{T}$. □

Proof: EXERCISE [RECOMMENDED, ★★★]. □

23.5.2 THEOREM [PROGRESS]: If $\mathtt{t}$ is a closed, well-typed term, then either $\mathtt{t}$ is a value or else there is some $\mathtt{t}'$ with $\mathtt{t} \longrightarrow \mathtt{t}'$. □

Proof: EXERCISE [RECOMMENDED, ★★★]. □