Chapter 23: Universal Types

System F (polymorphic lambda calculus)

Power of System F

Properties (Soundness, decidability, paramertricity, impredicativity)

Abstraction

$$\text{doubleNat} = \lambda f{:}\text{Nat}\to\text{Nat}.\ \lambda x{:}\text{Nat}.\ f\ (f\ x);$$

$$\text{doubleRcd} = \lambda f{:}\{l{:}\text{Bool}\}\to\{l{:}\text{Bool}\}.\ \lambda x{:}\{l{:}\text{Bool}\}.\ f\ (f\ x);$$

$$\text{doubleFun} = \lambda f{:}(\text{Nat}\to\text{Nat})\to(\text{Nat}\to\text{Nat}).\ \lambda x{:}\text{Nat}\to\text{Nat}.\ f\ (f\ x);$$

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

$$\text{double} = \lambda X.\ \lambda f{:}X\to X.\ \lambda x{:}X.\ f\ (f\ x)$$

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 $\lambda\to$ 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

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

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

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-APP1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-APP2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-APPABS)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ [T_2] \longrightarrow t_1'\ [T_2]} \quad \text{(E-TAPP)}$$

$$(\lambda X.t_{12})\ [T_2] \longrightarrow [X \mapsto T_2]t_{12} \quad \text{(E-TAPPTABS)}$$

Types and Type Context

T	::=		*types:*
		X	*type variable*
		$T \rightarrow T$	*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 : ∀X. List X
- cons : ∀X. X → List X → List X
- isnil : ∀X. List X → Bool
- head : ∀X. List X → X
- tail : ∀X. List X → 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 = $\forall X. X \to X \to X$;
  - tru = $\lambda X.\ \lambda t{:}X.\ \lambda f{:}X.\ t$;
  - fls = $\lambda X.\ \lambda t{:}X.\ \lambda f{:}X.\ f$;
  - not = $\lambda b{:}\mathrm{CBool}.\ \lambda X.\ \lambda t{:}X.\ \lambda f{:}X.\ b\ [X]\ f\ t$;

- CNat = $\forall X.\ (X \to X) \to X \to X$
  - c0 = $\lambda X.\ \lambda s{:}X \to X.\ \lambda z{:}X.\ z$
  - c1= $\lambda X.\ \lambda s{:}X \to X.\ \lambda z{:}X.\ s\ z$;
  - csucc = $\lambda n{:}\mathrm{CNat}.\ \lambda X.\ \lambda s{:}X \to X.\ \lambda z{:}X.\ s\ (n\ [X]\ s\ z)$
  - cplus = $\lambda m{:}\mathrm{CNat}.\ \lambda n{:}\mathrm{CNat}.\ \lambda X.\ \lambda s{:}X \to X.\ \lambda z{:}X.$
    $m\ [X]\ s\ (n\ [X]\ s\ z)$

Ex.: Encoding Lists

- $\text{List } X = \forall R.\ (X \to R \to R) \to R \to R$
  - nil = $\lambda X.\ (\lambda R.\ \lambda c{:}X \to R \to R.\ \lambda n{:}R.\ n)$ as $\forall X.$ List X
  - cons = $\lambda X.\ \lambda hd{:}X.\ \lambda tl{:}\text{List } X.$

    $(\lambda R.\ \lambda c{:}X \to R \to R.\ \lambda n{:}R.\ c\ hd\ (tl\ [R]\ c\ n))$ as List X;
  - isnil = $\lambda X.\ \lambda l{:}\text{List } X.$

    $l\ [\text{Bool}]\ (\lambda hd{:}X.\ \lambda tl{:}\text{Bool}.\ \text{false})\ \text{true}$
  - head = $\lambda X.\ \lambda l{:}\text{List } X.$

    $l\ [X]\ (\lambda hd{:}X.\ \lambda tl{:}X.\ hd)\ (\text{diverge}\ [X]\ \text{unit})$
  - sum : List Nat → Nat

    sum = … definition without using fix …?

Ex.: Encoding Pair

- $\text{Pair } X\ Y = \lambda R.\ (X \to Y \to R) \to R;$
  - $\text{pair} : \forall X.\ \forall Y.\ X \to Y \to \text{Pair } X\ Y$
  - $\text{fst} : \forall X.\ \forall Y.\ \text{Pair } X\ Y \to X$
  - $\text{snd} : \forall X.\ \forall Y.\ \text{Pair } X\ Y \to Y$

$\text{pair} = \lambda X.\ \lambda Y.\ \lambda x{:}X.\ \lambda y{:}Y.\ \lambda R.\ \lambda p.\ \ p\ x\ y$

$\text{fst} = \lambda X.\ \lambda Y.\ \lambda p.\ p\ [X]\ (\lambda x.\ \lambda y \to x)$
$\text{snd} = \lambda X.\ \lambda Y.\ \lambda p.\ p\ [X]\ (\lambda x.\ \lambda y \to y)$

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

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

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

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

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.

$(\forall X.X \to X) \to Nat$	OK
$Nat \to ((\forall X.X \to X) \to (Nat \to Nat))$	OK
$((\forall X.X \to X) \to Nat) \to Nat$	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

$$\begin{aligned}
&\mathsf{CBool} = \forall X.X{\rightarrow}X{\rightarrow}X;\\
&\mathsf{tru} = \lambda X.\ \lambda t{:}X.\ \lambda f{:}X.\ t;\\
&\mathsf{fls} = \lambda X.\ \lambda t{:}X.\ \lambda f{:}X.\ f;
\end{aligned}$$

(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, ★★★]. □