Design Principles of Programming Languages

Universal Types

Zhenjiang Hu, Haiyan Zhao, Yingfei Xiong
Peking University, Spring Term

Reminder: Final Presentation

- Presentation: 10 minutes / group
- Question and Answer: 4 minutes / group

System F

- The foundation for polymorphism in modern languages
  - C++, Java, C#, Modern Haskell
- Discovered by
  - Jean-Yves Girard (1972)
  - John Reynolds (1974)
- Also known as
  - Polymorphic λ-calculus
  - Second-order λ-calculus
    - (Curry-Howard) Corresponds to second-order intuitionistic logic
  - Impredicative polymorphism (for the polymorphism mechanism)

Exercise

- Considering HM-System. What is the type of this program?
- let f = λx.x in
  let g = λx.f (f x) in
  {g 5, g true}

Exercise

- Considering HM-System. What is the type of this program?
- (λf.
  let g = λx.f (f x) in
  {g 5, g true}
) (λx.x)

Exercise

- Considering HM-System. What is the type of this program?
- let h= λx.x in
  (λf.
   let g = λx.f (f x) in
   {g 5, g true}
) h

System F by Examples

```
  id = λX. λx:X. x;
▸ id : ∀X. X → X
  id [Nat];
▸ <fun> : Nat → Nat
  id [Nat] 0;
▸ 0 : Nat
```

Exercise

- What are the types of the following terms?
  - double=λX. λf:X→X. λa:X.f (f a)
  - double [Nat]
  - double [Nat→Nat]

Key to Exercise

- What are the types of the following terms?
  - double=λX. λf:X→X. λa:X.f (f a)
    - ∀X. (X→X) → X →X
  - double [Nat]
    - (Nat→ Nat) →Nat→ Nat
  - double [Nat→Nat]
    - ((Nat→ Nat) → Nat→ Nat) → (Nat→ Nat) → Nat→ Nat

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

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

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-App2)}$$

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

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

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

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

$$\frac{\mathtt{x}{:}\mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \quad \text{(T-Var)}$$

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda\mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1\to\mathtt{T}_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11}\to\mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \quad \text{(T-App)}$$

$$\frac{\Gamma, \mathtt{X} \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda\mathtt{X}.\mathtt{t}_2 : \forall\mathtt{X}.\mathtt{T}_2} \quad \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \forall\mathtt{X}.\mathtt{T}_{12}}{\Gamma \vdash \mathtt{t}_1\ [\mathtt{T}_2] : [\mathtt{X} \mapsto \mathtt{T}_2]\mathtt{T}_{12}} \quad \text{(T-TApp)}$$

Exercise

- Can we type this term in simple typed λ-calculus?
  - $\lambda x.\, x\ x$

Exercise

- Can we type this term in system F (by adding type declarations and arguments)?
  - $\lambda x.\, x\ x$

Exercise

- Can we type this term in system F (by adding type declarations and arguments)?
  - $\lambda x. x\ x$

- $\lambda x: \forall X. X \to X.$ x $[\forall X. X \to X]$ x

- $\mathrm{double} = \lambda \mathrm{X}. \lambda \mathrm{x}: \mathrm{X} \to \mathrm{X}. \lambda \mathrm{y}: \mathrm{X}. \mathrm{x}\ \mathrm{x}\ \mathrm{y}$
- $\mathrm{double}: \forall X. (\mathrm{X} \to \mathrm{X}) \to (\mathrm{X} \to \mathrm{X})$
- quadruple = λX. $\lambda \mathrm{double}: \forall \mathrm{X}. (\mathrm{X} \to \mathrm{X}) \to (\mathrm{X} \to \mathrm{X}).$ double [X→X] (double [X])

Exercise

- Implment csucc for CNat so that c_i = csucc c_{i-1}

```
CNat = ∀X. (X→X) → X → X;

c0  =  λX. λs:X→X. λz:X. z;
▸ c0 : CNat

c1   =  λX. λs:X→X. λz:X. s z;
▸ c1 : CNat

c2   =  λX. λs:X→X. λz:X. s (s z);
▸ c2 : CNat
```

Exercise

- Implment csucc for CNat so that c_i = csucc c_{i-1}

```
CNat = ∀X. (X→X) → X → X;
c0  =  λX. λs:X→X. λz:X. z;
▸ c0 : CNat
c1   =  λX. λs:X→X. λz:X. s z;
▸ c1 : CNat
c2   =  λX. λs:X→X. λz:X. s (s z);
▸ c2 : CNat
scc = λn. λs. λz. s (n s z);
```

Exercise

- Implment csucc for CNat so that c_i = csucc c_{i-1}

```
CNat = ∀X. (X→X) → X → X;
c0  =  λX. λs:X→X. λz:X. z;
▸ c0 : CNat
c1   =  λX. λs:X→X. λz:X. s z;
▸ c1 : CNat
c2   =  λX. λs:X→X. λz:X. s (s z);
▸ c2 : CNat
csucc = λn:CNat. λX. λs:X→X. λz:X. s (n [X] s z);
▸ csucc : CNat → CNat
```

Extending System F

- Introducing advanced types by directly copying the extra rules
  - Tuples, Records, Variants, References, Recursive types

- PolyPair = ∀X. ∀Y. {X, Y}

Can you define list in System F?

- List =…
- nil = …
- cons = …

Can you define list in System F?

- List = ∀X. μA. <nil:Unit, cons:{X, A}>;
- Let List X = μA. <nil:Unit, cons:{X, A}>
  - nil = λX. <nil:Unit> as List X
  - cons = λX. λn:X.λl:List X.<cons={n, l}> as List X

- cons [Nat] 2 (nil [Nat])
- tail = $\lambda X.\lambda l\colon List\ X.$ case l of
<nil=u> => nil
<cons=p> => p.2
- Problem: List X exposes the internal structure
  - Solving this problem requires System Fω

Church Encoding

- Read the book

Basic Properties

- Preservation
- Progress
- Normalization
  - Every typable term halts.
  - Y Combinator cannot be written in System F.

Efficiency Issue

- Additional evaluation rule adds runtime overhead.

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

- Solution:
  - Only use types in type checking
  - Erase types during compilation

Removing types

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

t reduces to t' ⇒ erase(t) reduces to erase(t')

A Problem in Extended System F

- Do the following two terms the same?
  - $\lambda x.\, x$ (λX.error);
  - $\lambda x.\, x$ error;

Review: Error

$$\Gamma \vdash \texttt{error} : T \qquad \text{(T-ERROR)}$$

New syntactic forms

t	::=	...	*terms:*
		error	*run-time error*

New evaluation rules $\boxed{t \longrightarrow t'}$

$$\texttt{error}\ t_2 \longrightarrow \texttt{error} \qquad \text{(E-APPERR1)}$$

$$v_1\ \texttt{error} \longrightarrow \texttt{error} \qquad \text{(E-APPERR2)}$$

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\Gamma \vdash \texttt{error} : T \qquad \text{(T-ERROR)}$$

A Problem in Extended System F

- Do the following two terms the same?
  - $\lambda x.x$ (λX.error); // a value
  - $\lambda x.x$ error; // reduce to error

- A new erase function

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

Wells' Theorem

- Can we construct types in System F?
  - One of the longest-standing problems in programming languages
  - 1970s – 1990s
- [Wells94] It is undecidable whether, given a closed term m of the untyped λ-calculus, there is some well-typed term t in System F such that $erase(t) = m$.

Rank-N Polymorphism

- In AST, any path from the root to an ∀ passes the left of no more than N-1 arrows
  - $\forall X. X \rightarrow X$:
    - Rank 1
  - $(\forall X. X \rightarrow X) \rightarrow Nat$:
    - Rank 2
  - $((\forall X. X \rightarrow X) \rightarrow Nat) \rightarrow Nat$:
    - Rank 3
  - $Nat \rightarrow (\forall X. X \rightarrow X) \rightarrow Nat \rightarrow Nat$:
    - Rank 2
  - $Nat \rightarrow (\forall X. X \rightarrow X) \rightarrow Nat$:
    - Rank 2

Rank-N Polymorphism

- Rank-1 is HM-system
  - Polymorphic types cannot be passed as parameters
- Type inference for rank-2 is decidable
  - Polymorphic types cannot be used in high-order functional parameters
- Type inference for rank-3 or more is undecidable

- What is the rank of C++ template, Java/C# generics?
  - Rank-1, because any generic parameters passed to a function must be instantiated