CSCI 740 - Programming Language Theory

Lecture 14

Hindley-Milner Type System

Instructor: Hossein Hojjat

October 2, 2017

Recap: System F

System F (or the polymorphic λ-calculus):
minimal language that illustrates the core concepts of polymorphic typing

$$\tau ::= \alpha \mid \mathsf{Int} \mid \tau \to \tau \mid \forall\alpha.\tau$$
$$e ::= n \mid x \mid \lambda x : \tau.e \mid e \;\; e \mid \Lambda\alpha.e \mid e[\tau]$$

Summary of new things:

- Type variables and Universal types
- Type abstraction (generalization)
- Type application (instantiation)

Example

Let id $= \Lambda\alpha.\lambda x : \alpha.x$

- id has type $\forall\alpha.\alpha \to \alpha$
- id [Int] has type Int $\to$ Int
- id [Int $\to$ Int] has type (Int $\to$ Int) $\to$ (Int $\to$ Int)

Example

$$
\begin{aligned}
&(\Lambda\alpha.\Lambda\beta.\lambda x : \alpha.\lambda f : \alpha \to \beta.f\ x)\ [\mathsf{Int}]\ [\mathsf{Int}]\ 5\ (\lambda y : \mathsf{Int}.y + y)\\
\to&(\Lambda\beta.\lambda x : \mathsf{Int}.\lambda f : \mathsf{Int} \to \beta.f\ x)\ [\mathsf{Int}]\ 5\ (\lambda y : \mathsf{Int}.y + y)\\
\to&(\lambda x : \mathsf{Int}.\lambda f : \mathsf{Int} \to \mathsf{Int}.f\ x)\ 5\ (\lambda y : \mathsf{Int}.y + y)\\
\to&(\lambda f : \mathsf{Int} \to \mathsf{Int}.f\ 5)(\lambda y : \mathsf{Int}.y + y)\\
\to&(\lambda y : \mathsf{Int}.y + y)\ 5\\
\to&(5 + 5)\\
\to&\ 10
\end{aligned}
$$

Type Rules

Type Abstraction (Generalization)

$$\frac{\Gamma, \alpha \vdash e : \tau}{\Gamma \vdash \Lambda\alpha.e \ : \ \forall\alpha.\tau}$$

Type Application (Instantiation)

$$\frac{\Gamma \vdash e : \forall\alpha.\tau}{\Gamma \vdash e[\tau'] : \tau[\alpha \mapsto \tau']}$$

Type inference with Polymorphism

let

id $= \lambda x.x$

in

... (id `true`) ... (id 1) ...

Constraints:

$$\begin{array}{lll} \mathsf{id} & : \alpha_1 & \to \alpha_1 \\ \mathsf{id} & : \mathsf{Bool} & \to \alpha_2 \\ \mathsf{id} & : \mathsf{Int} & \to \alpha_3 \end{array}$$

Does not unify!

Solution: Generalize the type variable

$$\mathsf{id} \ : \ \forall \alpha_1.\alpha_1 \to \alpha_1$$

Different uses of a generalized type variable may be instantiated differently

$$\mathsf{id}_1 : \mathsf{Bool} \to \mathsf{Bool}$$

$$\mathsf{id}_2 : \mathsf{Int} \to \mathsf{Int}$$

Attempting Type Inference

- Let's extend simply-typed calculus as follows:

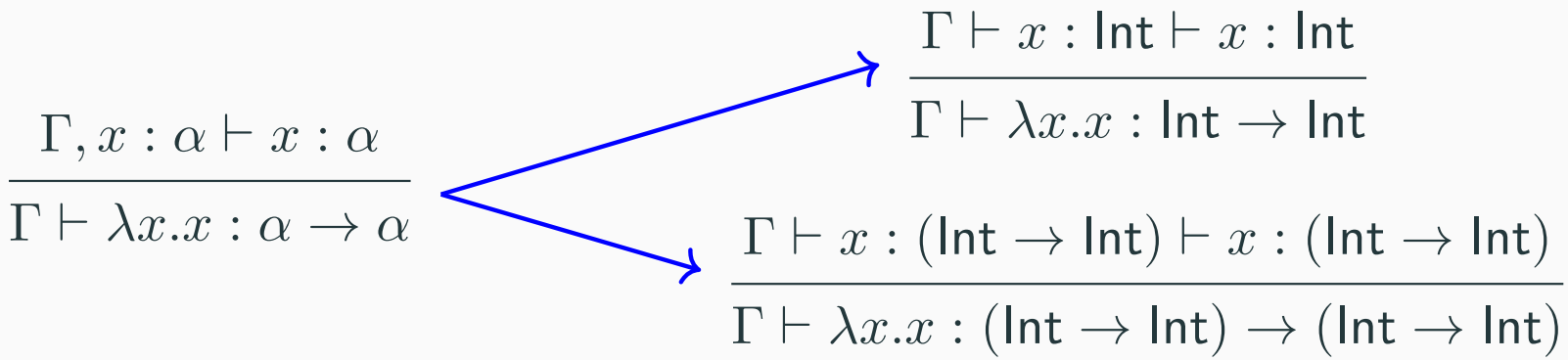
$$\tau ::= \alpha \mid \mathsf{Int} \mid \tau \to \tau \mid \forall \alpha.\tau$$
$$e ::= n \mid x \mid \lambda x.e \mid \ e\ e$$

- Type inference will automatically infer where to generalize a term, to introduce polymorphic types, and where to instantiate them

Generalization

- **Question:** When is it safe to generalize (quantify) a type variable α in the type of expression e?
- **Answer:** Whenever we can redo the typing proof for e, choosing α to be anything we want, and still have a valid typing proof

Generalization Example



- The choice of the type of x is purely local to type checking $\lambda x.x$
- There is no interaction with the outside environment
- Thus we can generalize the type of x

Generalization Example

Can we generalize the type of x?

$$\frac{\Gamma, x : \mathsf{Int} \vdash x : \mathsf{Int}}{\Gamma \vdash \lambda x.x + 3}$$

Generalization Example

Can we generalize the type of x?

$$\frac{\Gamma, x : \mathsf{Int} \vdash x : \mathsf{Int}}{\Gamma \vdash \lambda x.x + 3}$$

- The function restricts the type of x
- We cannot introduce a type variable
- Thus we cannot generalize the type of x
- We can only generalize when the function does not "look at" its parameter

Generalization Example

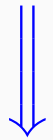
Can we generalize the type of x?

$$\frac{\Gamma, y : \alpha, x : \alpha \vdash \text{if } p \text{ then } x \text{ else } y : \alpha}{\Gamma, y : \alpha \vdash \lambda x.\text{if } p \text{ then } x \text{ else } y : \alpha \to \alpha}$$

Generalization Example

Can we generalize the type of x?

$$\frac{\Gamma, y : \alpha, x : \alpha \vdash \text{if } p \text{ then } x \text{ else } y : \alpha}{\Gamma, y : \alpha \vdash \lambda x.\text{if } p \text{ then } x \text{ else } y : \alpha \to \alpha}$$



$$\cancel{\frac{\Gamma, y : \alpha, x : \text{Int} \vdash \text{if } p \text{ then } x \text{ else } y : \text{Int}}{\Gamma, y : \alpha \vdash \lambda x.\text{if } p \text{ then } x \text{ else } y : \text{Int} \to \text{Int}}}$$

- The choice of the type of x depends on the type environment
- In the first rule, x and y have the same type
- If we generalize the type of x, they could have different types
- Thus we cannot generalize the type of x

Generalization Rule

- We can generalize any **type variable** that is unconstrained by the environment

$$\frac{\Gamma \vdash e : \tau \qquad \alpha \notin \mathsf{FV}(\Gamma)}{\Gamma \vdash e : \forall \alpha . \tau}$$

Inference for Polymorphism

- Type inference in System F (without explicit type annotations) is **undecidable**
- Can we at least perform some type inference for parametric polymorphism?
- Yes. A sweet spot was found by Hindley and Milner

Hindley-Milner Polymorphism

- Restrict polymorphism to only the **top level**
- Introduce polymorphism at **let**
- Fully instantiate at use of a polymorphic type
- Here is the new language
- Expression: $e ::= n \mid x \mid \lambda x.e \mid e\ e \mid \text{let } x = e \text{ in } e$
- Types: $\tau ::= \alpha \mid \text{Int} \mid \tau \to \tau$
- Types Schemes: $s ::= \tau \mid \forall \alpha.s$
- Notice that we do not instantiate α with a scheme s, only a type τ

Old Type Inference Rules

$$\frac{}{\Gamma \vdash n : \mathsf{Int}}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma, x : \alpha \vdash e : \tau' \qquad \alpha \text{ fresh}}{\Gamma \vdash \lambda x.e : \alpha \to \tau'}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2 \qquad \boxed{\tau_1 = \tau_2 \to \alpha} \qquad \alpha \text{ fresh}}{\Gamma \vdash e_1 \;\; e_2 : \alpha}$$

New Type Inference Rules

- At **let** generalize over all possible variable

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \forall \vec{\alpha}.\tau_1 \vdash e_2 : \tau_2 \qquad \vec{\alpha} = \mathsf{FV}(\tau_1) - \mathsf{FV}(\Gamma)}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2}$$

- At variable uses, instantiate to all fresh types

$$\frac{x : \forall \vec{\alpha}.\tau \qquad \vec{\beta}\ \mathsf{fresh}}{\Gamma \vdash x : \tau[\vec{\alpha} \mapsto \vec{\beta}]}$$

- Here $\vec{\alpha}$ denotes a list of type variables