Type Inference

Professor: Suman Saha

Introduction

- So far when we studied typing, we always assumed that the programmer annotated some types
- Example: We gave types to lambda variables
- But annotating types can be cumbersome.
- Goal of type inference: Automatically deduce the most general type for each expression
- Two key points:
  - Automatically inferring types: This means the programmer has to write no types, but still gets all the benefit from static typing
  - Inferring the most general type: This means we want to infer polymorphic types whenever possible

Examples

- Do we actually need these type annotation to infer the type of programs?
- Consider the following examples:

  - let $f_1 = \lambda x.\ x+2$
    - type of f_1 must be $\mathrm{Int} \rightarrow \mathrm{Int}$

  - let $f_2 = \lambda x.\ \lambda y.\ x + y$
    - type of f_2 must be $\mathrm{Int} \rightarrow \mathrm{Int} \rightarrow \mathrm{Int}$

  - let $f_3 = \lambda x.\ \lambda y.\ x + 1$
    - type of f_3 must be $\forall \alpha.\ \mathrm{Int} \rightarrow \alpha \rightarrow \mathrm{Int}$

  - let $f_4 = \lambda g.\ (g\ 0)$
    - type of f_4 is $\forall \alpha.\ (\mathrm{Int} \rightarrow \alpha) \rightarrow \alpha$

Type Inference Overview

- Goal is to develop an algorithm that can compute the most general type for any expression without any type annotation.
- Big Idea: Replace the concrete type Int annotation with a type variable and collect all constraints on this type variable.
- Specifically, pretend that the type of the argument is just some type variable called a

Example

- Type derivation tree for $\lambda x{:}\mathrm{int}.\ x+2$

- Type derivation tree for the same expression using type variable a

Generalizing Typing Rules

- The base case stay unchanged:

$\Gamma \vdash n : \mathtt{int}$ (T-NUM) $\quad \Gamma \vdash \mathtt{true} : \mathtt{bool}$ (T-TRUE) $\quad \Gamma \vdash s : \mathit{string}$ (T-String)

$\Gamma \vdash \mathtt{false} : \mathtt{bool}$ (T-FALSE) $\quad \Gamma, x : \tau \vdash x : \tau$ (T-VAR)

- Plus operator

$$\frac{\Gamma \vdash e_1 : \mathtt{int} \quad \Gamma \vdash e_2 : \mathtt{int}}{\Gamma \vdash (e_1 + e_2) : \mathtt{int}} \text{ (T-ADD)}$$

Generalizing Typing Rules

- Concatenation:

$$\frac{\Gamma \vdash e_1 : string \qquad \Gamma \vdash e_2 : string}{\Gamma \vdash (e_1 :: e_2) : string} \text{ (T-Con)}$$

- And operator

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash (e_1 \wedge e_2) : \texttt{bool}} \text{ (T-AND)}$$

Generalizing Typing Rules

- Abstraction:

$$\frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash (\lambda x : \tau \,.\, e) : \tau \rightarrow \tau'} \text{ (T-ABS)}$$

- Application

$$\frac{\Gamma \vdash e_1 : \tau \rightarrow \tau' \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \tau'} \text{ (T-APP)}$$

Top Hat

Example-1

- $(((\lambda x.\lambda y.x)2)true)$

Example-2

- "duck" + 7