CS 162 Programming languages

Lecture 13: Type Inference II

Yu Feng
Winter 2021

Type sytem in λ⁺

$$\frac{\Gamma, x : \mathsf{T}_1 \vdash e : \mathsf{T}_2}{\Gamma \vdash \mathsf{lambda}\ x : \mathsf{T}_1.\ e : \mathsf{T}_1 \to \mathsf{T}_2}\ \textsc{T-Lambda}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{T}_1 \to \mathsf{T}_2 \qquad \Gamma \vdash e_2 : \mathsf{T}_1}{\Gamma \vdash (e_1\ e_2) : \mathsf{T}_2}\ \textsc{T-App}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{Int} \qquad \Gamma \vdash e_2 : \mathsf{T} \qquad \Gamma \vdash e_3 : \mathsf{T}}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \mathsf{T}}\ \textsc{T-If} \qquad \frac{\Gamma \vdash e : \mathsf{List}[\mathsf{T}]}{\Gamma \vdash \mathsf{isnil}\ e : \mathsf{Int}}\ \textsc{T-IsNil}$$

$$\frac{x : \mathsf{T} \in \Gamma}{\Gamma \vdash x : \mathsf{T}}\ \textsc{T-Var} \qquad \frac{\Gamma, x : \mathsf{T}_1 \vdash e_1 : \mathsf{T}_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash e_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\ x : \mathsf{T}_1 = e_1\ \mathsf{in}\ e_2 : \mathsf{T}_2}\ \textsc{T-Let}$$

$$\frac{}{\Gamma \vdash \mathsf{Nil} : \mathsf{List}[\mathsf{T}]}\ \textsc{T-Nil} \qquad \frac{\Gamma \vdash e_1 : \mathsf{T} \qquad \Gamma \vdash e_2 : \mathsf{List}[\mathsf{T}]}{\Gamma \vdash e_1 @ e_2 : \mathsf{List}[\mathsf{T}]}\ \textsc{T-Cons}$$

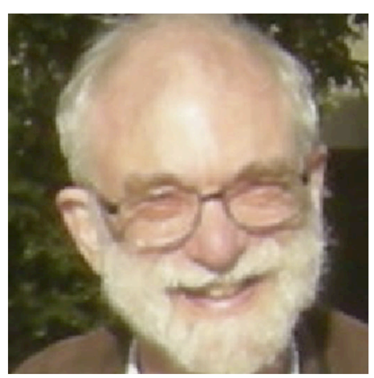
Type inference

- Goal of type inference: Automatically deduce the type for each expression

- Automatically inferring types: This means the programmer has to write no types, but still gets all the benefit from static typing

Hindley-Milner type inference

Develop an algorithm that can compute the most general type for any expression without any type annotations



J. Roger Hindley

Robin Milner
Turing Award (1991)

Type variables

- Big idea: Replace the concrete type ***Int*** annotated 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***

- And for all rules that have preconditions on ***a***, write these preconditions as constraints

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\textit{identifer } x\\ \Gamma(x)=Int\end{array}}{\Gamma[x\leftarrow Int]\vdash x:Int} \qquad \dfrac{\textit{integer } 2}{\Gamma[x\leftarrow Int]\vdash 2:Int}}{\Gamma[x\leftarrow \cancel{Int}]\vdash x+2:Int}}{\Gamma\vdash \lambda x{:}\underset{\boldsymbol{a}}{\cancel{Int}}.x+2:Int\to Int}$$

Type variables

- Here is the type derivation tree for this expression using type variable ***a***:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\textit{identifer } x\\ \Gamma(x) = a\end{array}}{\Gamma[x \leftarrow a] \vdash x : a} \qquad {\color{red} a = \mathit{Int}} \qquad \dfrac{\textit{integer } 2}{\Gamma[x \leftarrow a] \vdash 2 : \mathit{Int}}}{\Gamma[x \leftarrow a] \vdash x + 2 : \mathit{Int}}}{\Gamma \vdash \lambda x{:}a.x + 2 : a \to \mathit{Int}}$$

- Observe that we have one additional precondition on the plus rule: The type variable a must be equal to Int for this rule to apply.

- We now obtain the type: ***a*** → Int and the constraint ***a*** = Int

- Final type: Int → Int

Type variables in typing rules

- We dealt with not knowing the type of x in the following way:
- We introduced a type variable a for the type of x
- Every time a rule uses the type of x, we use a
- Since the plus rule has the precondition that both operands must be of type Int, we introduced a constraint $a = Int$
- After we typed the expression, we had a the type $a \to Int$ and the constraint $a = Int$
- Solving the collected constraint yields: $Int \to Int$

Generalizing this example

- This strategy generalizes!
- Introduce type variables for every type annotation
- Collect constraints on type variables during type checking
- Solve this type with respect to the collected constraints

Hindley–Milner Type Inference

Constraint typing rules

$$\frac{}{\Gamma \vdash i : \mathsf{Int}} \text{ CT-INT}$$

Any number has type int

$$\frac{\Gamma \vdash e_1 : \mathsf{T}_1 \qquad \Gamma \vdash e_2 : \mathsf{T}_2 \qquad \Box \in \{+, -, *\} \qquad \mathsf{T}_1 = \mathsf{Int} \qquad \mathsf{T}_2 = \mathsf{Int}}{\Gamma \vdash e_1 \Box e_2 : \mathsf{Int}} \text{ CT-ARITH}$$

e_1 and e_2 are of type int

Constraint typing rules

$$\frac{\mathsf{X}\ \text{fresh} \qquad \Gamma, x : \mathsf{X} \vdash e : \mathsf{T}}{\Gamma \vdash \mathsf{lambda}\, x.\ e : \mathsf{X} \to \mathsf{T}} \ \text{CT-LAMBDA}$$

Introduce a ***fresh*** type variable for parameter ***x***

The ones circled in red are constraints

$$\frac{\Gamma \vdash e_1 : \mathsf{T}_1 \qquad \Gamma \vdash e_2 : \mathsf{T}_2 \qquad \mathsf{X}_1, \mathsf{X}_2\ \text{fresh} \qquad \mathsf{T}_1 = \mathsf{X}_1 \to \mathsf{X}_2 \qquad \mathsf{T}_2 = \mathsf{X}_1}{\Gamma \vdash (e_1\ e_2) : \mathsf{X}_2} \ \text{CT-APP}$$

Introduce ***fresh*** type variables for functions e_1 and argument e_2

Constraint typing rules

$$\frac{\mathsf{X}\ \text{fresh} \qquad \Gamma, x : \mathsf{X} \vdash e : \mathsf{T}}{\Gamma \vdash \mathsf{lambda}\ x.\ e : \mathsf{X} \to \mathsf{T}}\ \text{CT-LAMBDA}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \mathsf{T}_1 \qquad \Gamma \vdash e_2 : \mathsf{T}_2 \\ \mathsf{X}_1, \mathsf{X}_2\ \text{fresh} \qquad \mathsf{T}_1 = \mathsf{X}_1 \to \mathsf{X}_2 \qquad \mathsf{T}_2 = \mathsf{X}_1\end{array}}{\Gamma \vdash (e_1\ e_2) : \mathsf{X}_2}\ \text{CT-APP}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \mathsf{T}_1 \qquad \Gamma \vdash e_2 : \mathsf{T}_2 \qquad \Box \in \{+, -, *\} \\ \mathsf{T}_1 = \mathsf{Int} \qquad \mathsf{T}_2 = \mathsf{Int}\end{array}}{\Gamma \vdash e_1 \Box e_2 : \mathsf{Int}}\ \text{CT-ARITH}$$

$$\frac{}{\Gamma \vdash i : \mathsf{Int}}\ \text{CT-INT}$$

```
(lambda x. x + 2) 5
```

constraint generation

$\mathrm{Int} = T_2$ (CT-INT)
$T_2 = X_1$ (CT-APP, CT-INT)
$T_1 = X_1 \to X_2$ (CT-APP)
$X_2 = \mathrm{Int}$ (CT-ARITH)

constraint solving

$X_1 = X_2 = T_2 = \mathrm{Int}$
$T_1 = \mathrm{Int} \to \mathrm{Int}$

Constraint solving

Algorithm 1 Unification Algorithm

procedure UNIFY(C)
  if $C = \emptyset$ **then**
    success
  else
    $C = \{S = T\} \cup C'$
    if $S = T$ **then**
      unify(C')
    else if $S = \tau \wedge \tau \notin FV(T)$ **then**
      unify($C'[\tau \mapsto T]$) $\circ [\tau \mapsto T]$
    else if $T = \tau \wedge \tau \notin FV(S)$ **then**
      unify($C'[\tau \mapsto T]$) $\circ [\tau \mapsto T]$
    else if $S = S_1 \rightarrow S_2 \wedge T = T_1 \rightarrow T_2$ **then**
      unify($C' \cup \{S_1 = T_1, S_2 = T_2\}$)
    else
      fail

Choose a constraint S=T from C and let C' denote the remaining constraints

Occur check to avoid generating a cyclic substitution such as $[X \mapsto X \rightarrow X]$

$\circ$ denotes composition of two substitutions

Constraint solving

`(lambda x. x + 2) 5`

constraint generatio

$Int = T_2$ (CT-INT)
$T_2 = X_1$ (CT-APP, CT-INT)
$T_1 = X_1 \rightarrow X_2$ (CT-APP)
$X_2 = Int$ (CT-ARITH)

constraint solving

$X_1 = X_2 = T_2 = Int$
$T_1 = Int \rightarrow Int$

$Int = T_2,\ T_2 = X_1,\ T_1 = X_1 \rightarrow X_2,\ X_2 = Int$

① $T_2 \mapsto Int$ — $Int = X_1,\ T_1 = X_1 \rightarrow X_2,\ X_2 = Int$

② $X_1 \mapsto Int,\ T_2 \mapsto Int$ — $T_1 = Int \rightarrow X_2,\ X_2 = Int$

③ $X_1 \mapsto Int\ \ T_2 \mapsto Int$, $T_1 \mapsto Int \rightarrow X_2$ — $X_2 = Int$

④ $X_1 \mapsto Int,\ T_2 \mapsto Int$, $X_2 \mapsto Int$ — $\emptyset$ (done!)

$(T_1 \mapsto (Int \rightarrow X_2)) \circ (X_2 \mapsto Int)$

$\Downarrow$

$T_1 \mapsto Int \rightarrow Int$