Lecture 11: Type Inference I

Yu Feng
Winter 2024

Type sytem in λ^+

$$\frac{\Gamma \vdash e : T_2 \qquad T_1 = T_2}{\Gamma \vdash (e @ T_1) : T_1} \text{ T-ANNOT}$$

$$\frac{}{\Gamma \vdash i : \text{Int}} \text{ T-INT} \qquad \frac{\Gamma \vdash e_1 : \text{Int} \qquad \Gamma \vdash e_2 : \text{Int} \qquad \square \in \{+, -, *\}}{\Gamma \vdash e_1 \square e_2 : \text{Int}} \text{ T-ARITH}$$

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \text{ T-VAR} \qquad \frac{\Gamma \vdash e_1 : T_1 \qquad x : T_1, \Gamma \vdash e_2 : T_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : T_2} \text{ T-LET}$$

$$\frac{x : T_1, \Gamma \vdash e : T_2}{\Gamma \vdash (\text{lambda } x : T_1 . e) : T_1 \to T_2} \text{ T-LAMBDA}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 \qquad \Gamma \vdash e_2 : T_3 \qquad T_1 = T_3}{\Gamma \vdash (e_1 \ e_2) : T_2} \text{ T-APP}$$

$$\frac{}{\Gamma \vdash \text{true} : \text{Bool}} \text{ T-TRUE} \qquad \frac{}{\Gamma \vdash \text{false} : \text{Bool}} \text{ T-FALSE}$$

$$\frac{\Gamma \vdash e_1 : \text{Bool} \qquad \Gamma \vdash e_2 : T_1 \qquad \Gamma \vdash e_3 : T_2 \qquad T_1 = T_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T_1} \text{ T-IF}$$

$$\frac{\Gamma \vdash e_1 : \text{Int} \qquad \Gamma \vdash e_2 : \text{Int} \qquad \square \in \{<, >, =\}}{\Gamma \vdash e_1 \square e_2 : \text{Bool}} \text{ T-REL}$$

$$\frac{f : T, \Gamma \vdash e : T}{\Gamma \vdash (\text{fix } f : T \text{ is } e) : T} \text{ T-FIX}$$

$$\frac{}{\Gamma \vdash \text{Nil}[T] : \text{List}[T]} \text{ T-NIL} \qquad \frac{\Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : \text{List}[T]}{\Gamma \vdash e_1 :: e_2 : \text{List}[T]} \text{ T-CONS}$$

Type annotations

- So far when we studied typing, we always assumed that the programmer annotated some types

- Example: We gave types to let bindings and lambda variables in class

- But annotating types can be cumbersome!

- Anyone who has ever written C++ code can really empathize: `vector<Map<int,string>>::const_iterator it...`

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

Type inference example 1

- Do we really need these type annotations?

- Consider the following example:

```
let f = lambda x.x+2 in..
```

- Here, we know that function f adds two to its argument

- We also know that plus is only defined on integers

- Therefore, the type of f must be $Int \rightarrow Int$

Type inference example 2

- Consider the following example:

let f = lambda x.lambda y.x+y in ..

- Here, we know that function f has two (curried) arguments, x and y

- We also know that plus is only defined on integers

- Therefore, the type of f must be $Int \rightarrow Int \rightarrow Int$

Hindley-Milner type inference

Develop an algorithm that can compute the most general type for any expression without any type annotations
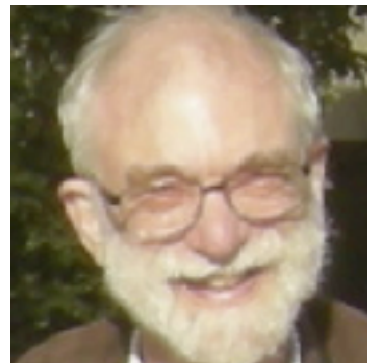
J. Roger Hindley

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

$$\frac{\dfrac{\begin{array}{c} identifer\ x \\ \Gamma(x) = Int \end{array}}{\Gamma[x \leftarrow Int] \vdash x : Int} \qquad \dfrac{integer\ 2}{\Gamma[x \leftarrow Int] \vdash 2 : Int}}{\dfrac{\Gamma[x \leftarrow Int] \vdash x + 2 : Int}{\Gamma \vdash \lambda x{:}Int.x + 2 : Int \rightarrow Int}}$$

a

Type variables

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

$$
\cfrac{
  \cfrac{
    \begin{array}{l} identifer\ x \\ \Gamma(x) = a \end{array}
  }{\Gamma[x \leftarrow a] \vdash x : a}
  \qquad a = Int \qquad
  \cfrac{
    integer\ 2
  }{\Gamma[x \leftarrow a] \vdash 2 : Int}
}{
  \cfrac{
    \Gamma[x \leftarrow a] \vdash x + 2 : Int
  }{\Gamma \vdash \lambda x{:}a.x + 2 : a \rightarrow Int}
}
$$

- 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: $a \rightarrow$ Int and the constraint a = Int

- Final type: Int $\rightarrow$ 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 $a \rightarrow Int$ and the constraint $a = Int$

- Solving the collected constraint yields: $Int \rightarrow Int$

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

Will talk about this in the next lecture

Constraint typing rules

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

Any number has type int

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

e_1 and e_2 are of type int

Constraint typing rules

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

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

The ones circled in red are constraints

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

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

Constraint typing rules

$$\frac{\textsf{X fresh} \quad \Gamma, x : \textsf{X} \vdash e : \textsf{T}}{\Gamma \vdash \textsf{lambda}\, x.\, e : \textsf{X} \rightarrow \textsf{T}} \; \text{CT-Lambda}$$

$$\frac{\Gamma \vdash e_1 : \textsf{T}_1 \quad \Gamma \vdash e_2 : \textsf{T}_2}{\textsf{X}_1, \textsf{X}_2 \textsf{ fresh} \quad \textsf{T}_1 = \textsf{X}_1 \rightarrow \textsf{X}_2 \quad \textsf{T}_2 = \textsf{X}_1}{\Gamma \vdash (e_1\ e_2) : \textsf{X}_2} \; \text{CT-App}$$

$$\frac{\Gamma \vdash e_1 : \textsf{T}_1 \quad \Gamma \vdash e_2 : \textsf{T}_2 \quad \square \in \{+, -, *\}}{\textsf{T}_1 = \textsf{Int} \quad \textsf{T}_2 = \textsf{Int}}{\Gamma \vdash e_1 \square e_2 : \textsf{Int}} \; \text{CT-Arith}$$

$$\frac{}{\Gamma \vdash i : \textsf{Int}} \; \text{CT-Int}$$

`(lambda x. x + 2) 5`

constraint generation

$\textsf{Int} = \textsf{T}_2$ (CT-INT)
$\textsf{T}_2 = \textsf{X}_1$ (CT-APP, CT-INT)
$\textsf{T1} = \textsf{X}_1 \rightarrow \textsf{X}_2$ (CT-APP)
$\textsf{X}_2 = \textsf{Int}$ (CT-ARITH)

constraint solving

$\textsf{X}_1 = \textsf{X}_2 = \textsf{T}_2 = \textsf{Int}$
$\textsf{T}_1 = \textsf{Int} \rightarrow \textsf{Int}$

Constraint typing rules

$$\frac{x : \mathsf{T} \in \Gamma}{\Gamma \vdash x : \mathsf{T}} \text{ CT-VAR}$$

Look up on the type environment

$$\frac{\mathsf{X} \text{ fresh} \qquad \Gamma, x : \mathsf{X} \vdash e_1 : \mathsf{T}_1 \qquad \Gamma, x : \mathsf{X} \vdash e_2 : \mathsf{T}_2 \qquad \mathsf{X} = \mathsf{T}_1}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \mathsf{T}_2} \text{ CT-LET}$$

Introduce fresh type variable for *x*