Lecture 11: Type Inference I

Yu Feng
Winter 2023

Type sytem in λ^+

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

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

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

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

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

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 → Int → Int*

Hindley-Milner type inference

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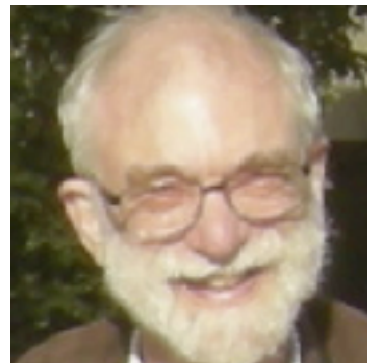
J. Roger Hindley

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**:

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

- 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** $\to$ Int and the constraint **a** = Int

- Final type: Int $\to$ 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

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

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

The ones circled in red are constraints

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

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

Constraint typing rules

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

$$\frac{\Gamma \vdash e_1 : T_1 \qquad \Gamma \vdash e_2 : T_2}{X_1, X_2 \text{ fresh} \qquad T_1 = X_1 \to X_2 \qquad T_2 = X_1}{\Gamma \vdash (e_1\ e_2) : X_2} \quad \text{CT-A\scriptsize{PP}}$$

$$\frac{\Gamma \vdash e_1 : T_1 \qquad \Gamma \vdash e_2 : T_2 \qquad \square \in \{+, -, *\}}{T_1 = \text{Int} \qquad T_2 = \text{Int}}{\Gamma \vdash e_1 \square e_2 : \text{Int}} \quad \text{CT-A\scriptsize{RITH}}$$

$$\frac{}{\Gamma \vdash i : \text{Int}} \quad \text{CT-I\scriptsize{NT}}$$

`(lambda x. x + 2) 5`

constraint generation

$\text{Int} = T_2$ (CT-INT)
$T_2 = X_1$ (CT-APP, CT-INT)
$T1 = X_1 \to X_2$ (CT-APP)
$X_2 = \text{Int}$ (CT-ARITH)

constraint solving

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

Constraint typing rules

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

Look up on the type environment

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

Introduce fresh type variable for x