CS 162 Programming languages

Lecture 11: Type Inference I

Yu Feng

Winter 2025

Type sytem in λ^+

$$\frac{\Gamma \vdash e : \mathsf{T}_2 \qquad \mathsf{T}_1 = \mathsf{T}_2}{\Gamma \vdash (e \,@\, \mathsf{T}_1) : \mathsf{T}_1}\ \textsc{T-Annot}$$

$$\frac{}{\Gamma \vdash i : \mathsf{Int}}\ \textsc{T-Int} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Int} \qquad \Gamma \vdash e_2 : \mathsf{Int} \qquad \square \in \{+, -, *\}}{\Gamma \vdash e_1 \,\square\, e_2 : \mathsf{Int}}\ \textsc{T-Arith}$$

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

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

$$\frac{\Gamma \vdash e_1 : \mathsf{T}_1 \to \mathsf{T}_2 \qquad \Gamma \vdash e_2 : \mathsf{T}_3 \qquad \mathsf{T}_1 = \mathsf{T}_3}{\Gamma \vdash (e_1\ e_2) : \mathsf{T}_2}\ \textsc{T-App}$$

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}}\ \textsc{T-True} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}}\ \textsc{T-False}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{Bool} \qquad \Gamma \vdash e_2 : \mathsf{T}_1 \qquad \Gamma \vdash e_3 : \mathsf{T}_2 \qquad \mathsf{T}_1 = \mathsf{T}_2}{\Gamma \vdash \mathsf{if}\, e_1 \,\mathsf{then}\, e_2 \,\mathsf{else}\, e_3 : \mathsf{T}_1}\ \textsc{T-If}$$

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

$$\frac{f : \mathsf{T}, \Gamma \vdash e : \mathsf{T}}{\Gamma \vdash (\mathsf{fix}\, f : \mathsf{T} \,\mathsf{is}\, e) : \mathsf{T}}\ \textsc{T-Fix}$$

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

Type 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 ..
```

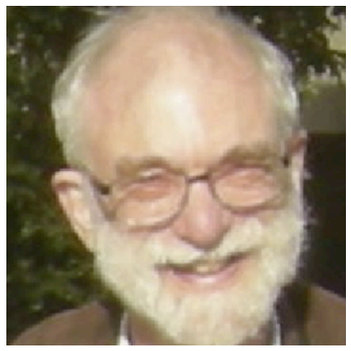
- Here, we know that function `f` has two (curried) arguments, x and y

- 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

$$\dfrac{\dfrac{\dfrac{identifer\ x \quad \Gamma(x) = Int}{\Gamma[x \leftarrow Int] \vdash x : Int} \quad \dfrac{integer\ 2}{\Gamma[x \leftarrow Int] \vdash 2 : Int}}{\Gamma[x \leftarrow Int] \vdash x + 2 : Int}}{\Gamma \vdash \lambda x{:}Int.x + 2 : Int \rightarrow Int}$$

a

Type variables

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

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

$$\Gamma \vdash \lambda x{:}a.x + 2 : a \rightarrow Int$$

- 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*** → 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*** → ***Int*** and the constraint ***a*** = ***Int***
- Solving the collected constraint yields: ***Int*** → ***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

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

Any number has type int

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

e_1 and e_2 are of type int

Constraint typing rules

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

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

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

The ones circled in red are constraints

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

Constraint typing rules

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

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

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

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

T1=X1->X2 **T2=X1=Int**

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

X2=Int

constraint generation

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

constraint solving

$X_1 = X_2 = T_2$ =Int
T_1 = Int → Int

Constraint typing rules

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

Look up on the type environment

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

Introduce fresh type variable for ***x***