

Lecture 13: Type Inference II

Yu Feng
Winter 2022

Type system in λ^+

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

$$\frac{\Gamma \vdash e_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash (e_1 \ e_2) : T_2} \text{ T-APP}$$

$$\frac{\Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T} \text{ T-IF}$$

$$\frac{\Gamma \vdash e : \text{List}[T]}{\Gamma \vdash \text{isnil } e : \text{Int}} \text{ T-ISNIL}$$

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

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

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

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : \text{List}[T]}{\Gamma \vdash e_1 @ e_2 : \text{List}[T]} \text{ T-CONS}$$

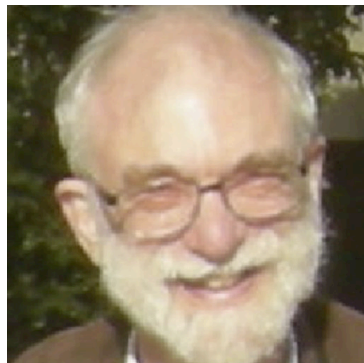
Type inference

- 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

$$\begin{array}{c}
 \text{identifier } x \\
 \hline
 \Gamma(x) = \text{Int} \\
 \hline
 \Gamma[x \leftarrow \text{Int}] \vdash x : \text{Int}
 \end{array}
 \quad
 \begin{array}{c}
 \text{integer } 2 \\
 \hline
 \Gamma[x \leftarrow \text{Int}] \vdash 2 : \text{Int}
 \end{array}$$

$$\begin{array}{c}
 \Gamma[x \leftarrow \text{Int}] \vdash x + 2 : \text{Int} \\
 \hline
 \Gamma \vdash \lambda x:\text{Int}. x + 2 : \text{Int} \rightarrow \text{Int}
 \end{array}$$

~~*a*~~

Type variables

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

$$\begin{array}{c}
 \text{identifier } x \\
 \hline
 \Gamma(x) = a \\
 \hline
 \Gamma[x \leftarrow a] \vdash x : a
 \end{array}
 \quad
 \textcolor{red}{a = Int}
 \quad
 \begin{array}{c}
 \text{integer } 2 \\
 \hline
 \Gamma[x \leftarrow a] \vdash 2 : Int
 \end{array}$$

$$\Gamma[x \leftarrow a] \vdash x + 2 : Int$$

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

- 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 $a = \mathbf{Int}$
- After we typed the expression, we had a the type $a \rightarrow \mathbf{Int}$ and the constraint $a = \mathbf{Int}$
- Solving the collected constraint yields: $\mathbf{Int} \rightarrow \mathbf{Int}$

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

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

Any number has type int

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

e_1 and e_2 are of type int

Constraint typing rules

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

Introduce a *fresh* type variable for parameter x

$$\frac{\begin{array}{c} \Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2 \\ X_1, X_2 \text{ fresh} \quad T_1 = X_1 \rightarrow X_2 \quad T_2 = X_1 \end{array}}{\Gamma \vdash (e_1 \ e_2) : X_2} \text{CT-APP}$$

The ones circled in red are constraints

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

Constraint typing rules

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

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2 \quad X_1, X_2 \text{ fresh} \quad T_1 = X_1 \rightarrow X_2 \quad T_2 = X_1}{\Gamma \vdash (e_1 e_2) : X_2} \text{CT-APP}$$

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

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

(lambda x. x + 2) 5

constraint generation

Int = T₂ (CT-INT)
 T₂ = X₁ (CT-APP, CT-INT)
 T₁ = X₁ → X₂ (CT-APP)
 X₂ = Int (CT-ARITH)

constraint solving

X₁ = X₂ = T₂ = Int
 T₁ = Int → 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( $C'[\tau \mapsto S]$ )  $\circ [\tau \mapsto S]$   
    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 performs substitutions over
the *type environment*

Perform substitutions over the
remaining constraints C'

Constraint solving

$(\text{lambda } x. x + 2) 5$

constraint generation

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

constraint solving

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

$\text{Int} = T_2, T_2 = X_1, T_1 = X_1 \rightarrow X_2, X_2 = \text{Int}$

$\text{Int} = X_1, T_1 = X_1 \rightarrow X_2, X_2 = \text{Int}$

① $T_2 \mapsto \text{Int}$

② $X_1 \mapsto \text{Int}, T_2 \mapsto \text{Int}$

③ $X_1 \mapsto \text{Int}, T_2 \mapsto \text{Int}$
 $T_1 \mapsto \text{Int} \rightarrow X_2$

$T_1 = \text{Int} \rightarrow X_2, X_2 = \text{Int}$

$X_2 = \text{Int}$

④ $X_1 \mapsto \text{Int}, T_2 \mapsto \text{Int}$

$X_2 \mapsto \text{Int} \quad (T_1 \mapsto (\text{Int} \rightarrow X_2))$
 $\circ (X_2 \mapsto \text{Int})$

\Downarrow

$T_1 \mapsto \text{Int} \rightarrow \text{Int}$

ϕ (done!)