CS 162 Programming languages

Lecture 10: Type Inference & Midterm Review

Yu Feng

Winter 2020

Hindley-Milner algorithm

For each top-level definition, traverse each AST to:

- **decorate** each AST node with preliminary type variable
- collect type constraints from each AST node
- use unification to solve constraints and produce a substitution
- use substitution to infer type of definition

Simple-typed lambda calculus

$$e ::= x \mid \lambda x : \tau \,.\, e \mid e_1\ e_2 \mid n$$

$$\tau ::= \textbf{int} \mid X \mid \tau_1 \rightarrow \tau_2$$

To formally define type inference, we introduce a new typing relation:

$$\Gamma \vdash e : \tau \triangleright C$$

Type environment

Meaning: Expression e has type τ provided that every constraint in the set C is satisfied.

Typing rules for STLC

$$\text{CT-VAR}\ \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau \triangleright \emptyset}$$

If x is of type τ in Env
then x has type τ

$$\text{CT-INT}\ \frac{}{\Gamma \vdash n : \textbf{int} \triangleright \emptyset}$$

All natural numbers have type int/number

$$\text{CT-ABS}\ \frac{\Gamma, x : \tau_1 \vdash e : \tau_2 \triangleright C}{\Gamma \vdash \lambda x : \tau_1 . e : \tau_1 \rightarrow \tau_2 \triangleright C}$$

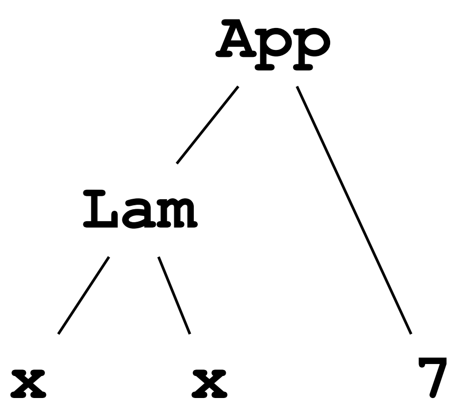
If formal parameter x is type τ1, and body e is type τ2, then the function is type τ1->τ2

$$\text{CT-APP}\ \frac{\begin{array}{c}\Gamma \vdash e_1 : \tau_1 \triangleright C_1 \quad \Gamma \vdash e_2 : \tau_2 \triangleright C_2 \\ C' = C_1 \cup C_2 \cup \{\tau_1 = \tau_2 \rightarrow X\}\end{array}}{\Gamma \vdash e_1\ e_2 : X \triangleright C'}\ X \text{ is fresh}$$

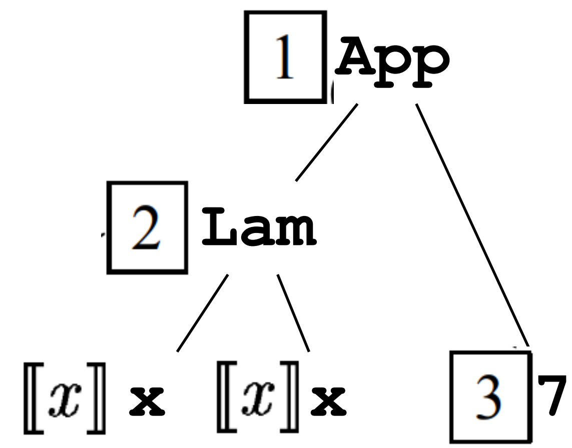
If function e1 is type τ1 which is in the form of τ2-> X,
and the argument e2 is type τ2, then the function call is type X

Example

What is the type of App(Lam(x, x), 7)?



Decorate AST with type vars



Generate constraints using typing rules

$$[\![\boxed{2}]\!] = [\![\boxed{3}]\!] \rightarrow [\![\boxed{1}]\!]$$

$$[\![\boxed{2}]\!] = [\![x]\!] \rightarrow [\![x]\!]$$

$$[\![\boxed{3}]\!] = \text{number}$$

How to solve those constraints?

Unification algorithm

$$
\begin{aligned}
\mathit{unify}(C) \quad = \quad & \text{if } C = \emptyset, \text{ then } \{\} \\
& \text{else let } \{\mathtt{S} = \mathtt{T}\} \cup C' = C \text{ in} \\
& \quad \text{if } \mathtt{S} = \mathtt{T} \\
& \quad \quad \text{then } \mathit{unify}(C') \\
& \quad \text{else if } \mathtt{S} = \mathtt{X} \text{ and } \mathtt{X} \notin FV(\mathtt{T}) \\
& \quad \quad \text{then } \mathit{unify}(\{\mathtt{X} \mapsto \mathtt{T}\}C') \circ \{\mathtt{X} \mapsto \mathtt{T}\} \\
& \quad \text{else if } \mathtt{T} = \mathtt{X} \text{ and } \mathtt{X} \notin FV(\mathtt{S}) \\
& \quad \quad \text{then } \mathit{unify}(\{\mathtt{X} \mapsto \mathtt{S}\}C') \circ \{\mathtt{X} \mapsto \mathtt{S}\} \\
& \quad \text{else if } \mathtt{S} = \mathtt{S}_1 \to \mathtt{S}_2 \text{ and } \mathtt{T} = \mathtt{T}_1 \to \mathtt{T}_2 \\
& \quad \quad \text{then } \mathit{unify}(C' \cup \{\mathtt{S}_1 = \mathtt{T}_1, \mathtt{S}_2 = \mathtt{T}_2\}) \\
& \quad \text{else} \\
& \quad \quad \text{fail}
\end{aligned}
$$

Occurrence check: make sure T does not contain type variable X

Apply substitution to both the remaining constraint C’ and the current mapping

Types and Programming Languages p326

Example

How to solve those constraints?

$$[\![\boxed{2}]\!] = [\![\boxed{3}]\!] \rightarrow [\![\boxed{1}]\!]$$

$$[\![\boxed{2}]\!] = [\![x]\!] \rightarrow [\![x]\!]$$

$$[\![\boxed{3}]\!] = \texttt{number}$$

Apply substitution to both the remaining constraint C’ and the current mapping

Action	Stack	Substitution
Initialize	$[\![\boxed{2}]\!] = [\![\boxed{3}]\!] \rightarrow [\![\boxed{1}]\!]$ $[\![\boxed{2}]\!] = [\![x]\!] \rightarrow [\![x]\!]$ $[\![\boxed{3}]\!] = \texttt{number}$	empty
Step 3	$[\![\boxed{3}]\!] \rightarrow [\![\boxed{1}]\!] = [\![x]\!] \rightarrow [\![x]\!]$ $[\![\boxed{3}]\!] = \texttt{number}$	$[\![\boxed{2}]\!] \mapsto [\![\boxed{3}]\!] \rightarrow [\![\boxed{1}]\!]$
Step 5	$[\![\boxed{3}]\!] = [\![x]\!]$ $[\![\boxed{1}]\!] = [\![x]\!]$ $[\![\boxed{3}]\!] = \texttt{number}$	$[\![\boxed{2}]\!] \mapsto [\![\boxed{3}]\!] \rightarrow [\![\boxed{1}]\!]$
Step 3	$[\![\boxed{1}]\!] = [\![x]\!]$ $[\![x]\!] = \texttt{number}$	$[\![\boxed{2}]\!] \mapsto [\![x]\!] \rightarrow [\![\boxed{1}]\!]$ $[\![\boxed{3}]\!] \mapsto [\![x]\!]$
Step 3	$[\![x]\!] = \texttt{number}$	$[\![\boxed{2}]\!] \mapsto [\![x]\!] \rightarrow [\![x]\!]$ $[\![\boxed{3}]\!] \mapsto [\![x]\!]$ $[\![\boxed{1}]\!] \mapsto [\![x]\!]$
Step 3	empty	$[\![\boxed{2}]\!] \mapsto \texttt{number} \rightarrow \texttt{number}$ $[\![\boxed{3}]\!] \mapsto \texttt{number}$ $[\![\boxed{1}]\!] \mapsto \texttt{number}$ $[\![x]\!] \mapsto \texttt{number}$

https://cs.brown.edu/courses/cs173/2002/Lectures/2002-11-13.pdf

Midterm review

- 75 min
- 75 points in total
- One A4 cheat sheet
- Two classrooms. ED-1213 (Yanju) and GIRV-2128 (Yimeng)
- Questions
  - Multiple choices
  - Short answers
  - Program output
  - Programming in OCaml and Lambda-calculus

Midterm review

- Lambda-calculus (α-renaming, β-reduction, evaluation)
- OCaml basics (let-binding, list, tuple, datatype, recursion)
- Datatypes (How to access? How to construct?)
- Recursion (tail recursion)
- Higher-order functions (map, fold, filter, etc.)
- Closure
- Type inference (provide the type of an expression)