Type Inference
CS 152 (Spring 2021)

Harvard University

Tuesday, March 23, 2021
Today, we will learn about

- Type inference
  - Type-checking vs type-inference
  - Constraint-based typing
  - Unification
Type annotations
Type inference

- Infer (or reconstruct) the types of a program
- Example: \( \lambda a. \lambda b. \lambda c. \text{if } a (b + 1) \text{ then } b \text{ else } c \)
Constraint-based Type Inference

- **Type variables** $X, Y, Z, \ldots$: placeholders for types.
- **Judgment** $\Gamma \vdash e : \tau \triangleright C$
  - Expression $e$ has type $\tau$ provided every constraint in set $C$ is satisfied
  - Constraints are of the form $\tau_1 \equiv \tau_2$
Language

\[ e ::= x \mid \lambda x : \tau. \ e \mid e_1 \ e_2 \mid n \mid e_1 + e_2 \]

\[ \tau ::= \text{int} \mid X \mid \tau_1 \to \tau_2 \]
Inference rules

CT-VAR \[ \Gamma \vdash x : \tau \triangleright \emptyset \]
\[ \Gamma \vdash x : \tau \in \Gamma \]

CT-Int \[ \Gamma \vdash n : \text{int} \triangleright \emptyset \]

CT-Add \[ \Gamma \vdash e_1 : \tau_1 \triangleright C_1 \quad \Gamma \vdash e_2 : \tau_2 \triangleright C_2 \]
\[ \Gamma \vdash e_1 + e_2 : \text{int} \triangleright C_1 \cup C_2 \cup \{ \tau_1 \equiv \text{int}, \tau_2 \equiv \text{int} \} \]
Inference rules, ctd.

**CT-ABS**

\[
\frac{\Gamma, x : \tau_1 \vdash e : \tau_2 \triangleright C}{\Gamma \vdash \lambda x : \tau_1. e : \tau_1 \to \tau_2 \triangleright C}
\]

\[
\begin{align*}
\Gamma &\vdash e_1 : \tau_1 \triangleright C_1 \\
\Gamma &\vdash e_2 : \tau_2 \triangleright C_2
\end{align*}
\]

\[C' = C_1 \cup C_2 \cup \{\tau_1 \equiv \tau_2 \rightarrow X\}\]

**CT-APP**

\[
\frac{C' = C_1 \cup C_2 \cup \{\tau_1 \equiv \tau_2 \rightarrow X\}}{\Gamma \vdash e_1 \ e_2 : X \triangleright C'}
\]

\(X\) is fresh
Example
Unification

- What does it mean for a set of constraints to be satisfied?
- How do we find a solution to a set of constraints (i.e., infer the types)?
- To answer these questions: we define *type substitutions* and *unification*
Type substitutions (aka substitutions)

- Map from type variables to types
- Substitution of type variables, formally:

\[
\sigma(X) = \begin{cases} 
\tau & \text{if } X \mapsto \tau \in \sigma \\
X & \text{if } X \text{ not in the domain of } \sigma
\end{cases}
\]

\[
\sigma(\text{int}) = \text{int}
\]

\[
\sigma(\tau \rightarrow \tau') = \sigma(\tau) \rightarrow \sigma(\tau')
\]
Substitution in constraints

Extended to substitution of constraints, and set of constrains:

\[ \sigma(\tau_1 \equiv \tau_2) = \sigma(\tau_1) \equiv \sigma(\tau_2) \]
\[ \sigma(C) = \{ \sigma(c) \mid c \in C \} \]
Unification

- Constraints are of form $\tau_1 \equiv \tau_2$
- Substitution $\sigma$ unifies $\tau_1 \equiv \tau_2$ if $\sigma(\tau_1)$ is the same as $\sigma(\tau_2)$
- Substitution $\sigma$ unifies (or satisfies) set of constraints $C$ if it unifies every constraint in $C$
- So given $\vdash e : \tau \triangleright C$, want substitution $\sigma$ that unifies $C$
  - Moreover, type of $e$ is $\sigma(\tau)$
Unification algorithm

\[ \text{unify}(C) = \sigma \]
unify(∅)

[]
\texttt{unify(\{\tau\equiv\tau'\} \cup C)}

\begin{itemize}
    \item if \(\tau = \tau'\) then \\
        \texttt{unify}(C)
    \item else if \(\tau = X\) and \(X\) not a free variable of \(\tau'\) then \\
        let \(\sigma = [X \mapsto \tau']\) in \\
        \texttt{unify}(\sigma(C)) \circ \sigma
    \item else if \(\tau' = X\) and \(X\) not a free variable of \(\tau\) then \\
        let \(\sigma = [X \mapsto \tau]\) in \\
        \texttt{unify}(\sigma(C)) \circ \sigma
    \item else if \(\tau = \tau_o \rightarrow \tau_1\) and \(\tau' = \tau'_o \rightarrow \tau'_1\) then \\
        \texttt{unify}(C \cup \{\tau_0\equiv\tau'_0, \tau_1\equiv\tau'_1\})
    \item else \texttt{fail}
\end{itemize}