Type Inference
CS 152 (Spring 2020)

Harvard University

Tuesday, March 31, 2020
Announcements

▶ HW2: grading in process...
▶ HW3: due Thu (Apr 2)
▶ HW4: released today, due in 2 weeks (Apr 14)
▶ HWs 5 and 6: topics/scope/number may change, stay posted...
▶ Google feedback form: not currently being monitored
▶ All feedback welcome
  ▶ Suggestions for improvement, any difficulties you are facing, ...
  ▶ (Can post anonymously to Piazza, even to instructors)
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 $X$, $Y$, $Z$, ...: 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

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

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

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

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

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

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

\[ \Gamma \vdash e_1 : \tau_1 \triangleright C_1 \]
\[ \Gamma \vdash e_2 : \tau_2 \triangleright C_2 \]

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

**CT-APP** \[ \frac{X \text{ is fresh}}{\Gamma \vdash e_1 \ e_2 : X \triangleright C'} \]
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
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