Type Inference

CS 152 (Spring 2024)

Harvard University

Thursday, March 21, 2024

Today, we will learn about

- Type inference
- Constraint-based typing
- Unification

Type annotations

$$\textsc{T-Abs}\ \frac{\Gamma, x{:}\tau \vdash e{:}\tau'}{\Gamma \vdash \lambda x{:}\tau.\, e{:}\tau \to \tau'}$$

Type inference

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

Example

$$\lambda a.\,\lambda b.\,\lambda c.\,\text{if } a\ (b+1) \text{ then } b \text{ else } c$$

Example

$$\lambda a.\, \lambda b.\, \lambda c.\, \text{if } a\ (b+1) \text{ then } b \text{ else } c$$

$$\lambda a{:}\,\mathbf{int} \to \mathbf{bool}.\, \lambda b{:}\,\mathbf{int}.\, \lambda c{:}\,\mathbf{int}.\, \text{if } a\ (b+1) \text{ then } b \text{ else } c$$

Example

$$\lambda x.\, \lambda y.\, x$$

Example

$$\lambda x.\, \lambda y.\, x$$

$$\lambda x : X.\, \lambda y : Y.\, x$$

Example

$$\lambda x.\, \lambda y.\, x$$

$$\lambda x : X.\, \lambda y : Y.\, x$$

$$\lambda x : \textbf{int}.\, \lambda y : \textbf{int}.\, x$$

Example

$$\lambda x.\, \lambda y.\, x$$

$$\lambda x : X.\, \lambda y : Y.\, x$$

$$\lambda x : \mathbf{int}.\, \lambda y : \mathbf{int}.\, x$$

$$\lambda x : \mathbf{int}.\, \lambda y : \mathbf{int} \to \mathbf{int}.\, x$$

Example

$$\lambda x.\, \lambda y.\, x$$

$$\lambda x : X.\, \lambda y : Y.\, x$$

$$\lambda x : \mathbf{int}.\, \lambda y : \mathbf{int}.\, x$$

$$\lambda x : \mathbf{int}.\, \lambda y : \mathbf{int} \to \mathbf{int}.\, x$$

$$\ldots$$

Example

$$\lambda x.\, \lambda f.\, f\ x$$

Example

$$\lambda x.\, \lambda f.\, f\ x$$

$$\lambda x : X.\, \lambda f : X \to Y.\, f\ x$$

Constraint-based Type Inference

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

Language

$$e ::= x \mid \lambda x\!:\!\tau.\, e \mid e_1\ e_2 \mid n \mid e_1 + e_2$$
$$\tau ::= \textbf{int} \mid X \mid \tau_1 \to \tau_2$$

Inference rules

$$\textsc{CT-Var}\ \frac{}{\Gamma \vdash x : \tau \triangleright \emptyset}\ x : \tau \in \Gamma$$

$$\textsc{CT-Int}\ \frac{}{\Gamma \vdash n : \textbf{int} \triangleright \emptyset}$$

Inference rules (2)

$$\textsc{CT-Add}\ \frac{\Gamma \vdash e_1 : \tau_1 \triangleright C_1 \qquad \Gamma \vdash e_2 : \tau_2 \triangleright C_2}{\Gamma \vdash e_1 + e_2 : \mathbf{int} \triangleright C_1 \cup C_2 \cup \{\tau_1 \equiv \mathbf{int}, \tau_2 \equiv \mathbf{int}\}}$$

Inference rules (3)

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

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

Example

$$\vdash \lambda a\!:\!X.\,\lambda b\!:\!Y.\,2 + (a\ (b + 3))$$

Example

$$\vdash \lambda a\colon X.\, \lambda b\colon Y.\, 2 + (a\ (b + 3))$$

Example

$$\vdash \lambda a\!:\!X.\,\lambda b\!:\!Y.\,2 + (a\ (b+3))$$

$$X \to Y \to \mathbf{int} \triangleright \{Z \equiv \mathbf{int},\, X \equiv \mathbf{int} \to Z,\, Y \equiv \mathbf{int},\, \mathbf{int} \equiv \mathbf{int}\}$$

Example

$$\vdash \lambda a{:}X.\,\lambda b{:}Y.\,2 + (a\ (b + 3))$$

$$X \rightarrow Y \rightarrow \textbf{int} \triangleright \{Z \equiv \textbf{int}, X \equiv \textbf{int} \rightarrow Z, Y \equiv \textbf{int}, \textbf{int} \equiv \textbf{int}\}$$

$$(\textbf{int} \rightarrow \textbf{int}) \rightarrow \textbf{int} \rightarrow \textbf{int}$$

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 by example

- The substitution $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$ maps
  - type variable X to **int**, and
  - type variable Y to $\textbf{int} \to \textbf{int}$.
- The same variable could occur in both the domain and range of a substitution.
- All substitutions are performed simultaneously.
- The substitution $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to X]$ maps
  - Y to $\textbf{int} \to \mathbf{X}$
  - (not to $\textbf{int} \to \textbf{int}$).

Type subsitutions (aka substitutions)

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

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

$$\sigma(\textbf{int}) = \textbf{int}$$

$$\sigma(\tau \to \tau') = \sigma(\tau) \to \sigma(\tau')$$

Substitution in constraints

- Extended to substitution of constraints, and set of constraints:

$$\sigma(\tau_1 \equiv \tau_2) = \sigma(\tau_1) \equiv \sigma(\tau_2)$$
$$\sigma(C) = \{\sigma(c) \mid c \in C\}$$

Example

The substitution $\sigma = [X \mapsto \textbf{int}, Y \mapsto \textbf{int} \rightarrow \textbf{int}]$ unifies the constraint $X \rightarrow (X \rightarrow \textbf{int}) \equiv \textbf{int} \rightarrow Y$, since

$$\begin{aligned} & \sigma(X \rightarrow (X \rightarrow \textbf{int})) \\ = & \textbf{int} \rightarrow (\textbf{int} \rightarrow \textbf{int}) \\ = & \sigma(\textbf{int} \rightarrow Y) \end{aligned}$$

Unification

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

Unification algorithm

$$unify(C) = \sigma$$

$unify(\emptyset)$

[]

$unify(\{\tau{\equiv}\tau'\} \cup C)$

if $\tau = \tau'$ then

$\quad unify(C)$

else if $\tau = X$ and X not a free variable of τ' then

$\quad$ let $\sigma = [X \mapsto \tau']$ in

$\quad unify(\sigma(C)) \circ \sigma$

else if $\tau' = X$ and X not a free variable of τ then

$\quad$ let $\sigma = [X \mapsto \tau]$ in

$\quad unify(\sigma(C)) \circ \sigma$

else if $\tau = \tau_o \to \tau_1$ and $\tau' = \tau_o' \to \tau_1'$ then

$\quad unify(C \cup \{\tau_0{\equiv}\tau_0', \tau_1{\equiv}\tau_1'\})$

else *fail*

Unification Algorithm (Things to Note)

- Choose a constraint from set C.
- Occurs check – free variable side conditions.

Principal Types

- Recall: given $\vdash e : \tau \triangleright C$, we want a substitution σ that unifies C. Moreover, type of e is $\sigma(\tau)$.
- We want the most general solution.

Principal unifier

- A substitution σ is
  less specific (or *more general*) *than*
  a substition σ',
  written $\sigma \sqsubseteq \sigma'$
  if $\sigma' = \gamma \circ \sigma$ for some substitution γ.
- A *principal unifier* (or *most general unifier*) for a constraint set C is a substitution σ that satisfies C and such that $\sigma \sqsubseteq \sigma'$ for every substitution σ' satisfying C.

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
- $\{\textbf{int} \to \textbf{int} \equiv X \to Y\}$
- $\{X \to Y \equiv Y \to Z, Z \equiv U \to W\}$
- $\{\textbf{int} \equiv \textbf{int} \to Y\}$
- $\{Y \equiv \textbf{int} \to Y\}$
- $\{\}$

Examples

Examples

- $\{X \equiv \mathbf{int}, Y \equiv X \rightarrow X\}$

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$
- $\{\textbf{int} \to \textbf{int} \equiv X \to Y\}$

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \rightarrow X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \rightarrow \textbf{int}]$
- $\{\textbf{int} \rightarrow \textbf{int} \equiv X \rightarrow Y\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int}]$

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$
- $\{\textbf{int} \to \textbf{int} \equiv X \to Y\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int}]$
- $\{X \to Y \equiv Y \to Z, Z \equiv U \to W\}$

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$
- $\{\textbf{int} \to \textbf{int} \equiv X \to Y\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int}]$
- $\{X \to Y \equiv Y \to Z, Z \equiv U \to W\}$
  $[X \mapsto U \to W, Y \mapsto U \to W, X \mapsto U \to W]$

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$
- $\{\textbf{int} \to \textbf{int} \equiv X \to Y\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int}]$
- $\{X \to Y \equiv Y \to Z, Z \equiv U \to W\}$
  $[X \mapsto U \to W, Y \mapsto U \to W, X \mapsto U \to W]$
- $\{\textbf{int} \equiv \textbf{int} \to Y\}$

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$
- $\{\textbf{int} \to \textbf{int} \equiv X \to Y\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int}]$
- $\{X \to Y \equiv Y \to Z, Z \equiv U \to W\}$
  $[X \mapsto U \to W, Y \mapsto U \to W, X \mapsto U \to W]$
- $\{\textbf{int} \equiv \textbf{int} \to Y\}$
  Not unifiable

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \rightarrow X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \rightarrow \textbf{int}]$
- $\{\textbf{int} \rightarrow \textbf{int} \equiv X \rightarrow Y\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int}]$
- $\{X \rightarrow Y \equiv Y \rightarrow Z, Z \equiv U \rightarrow W\}$
  $[X \mapsto U \rightarrow W, Y \mapsto U \rightarrow W, X \mapsto U \rightarrow W]$
- $\{\textbf{int} \equiv \textbf{int} \rightarrow Y\}$
  Not unifiable
- $\{Y \equiv \textbf{int} \rightarrow Y\}$

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$
- $\{\textbf{int} \to \textbf{int} \equiv X \to Y\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int}]$
- $\{X \to Y \equiv Y \to Z, Z \equiv U \to W\}$
  $[X \mapsto U \to W, Y \mapsto U \to W, X \mapsto U \to W]$
- $\{\textbf{int} \equiv \textbf{int} \to Y\}$
  Not unifiable
- $\{Y \equiv \textbf{int} \to Y\}$
  Not unifiable

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$
- $\{\textbf{int} \to \textbf{int} \equiv X \to Y\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int}]$
- $\{X \to Y \equiv Y \to Z, Z \equiv U \to W\}$
  $[X \mapsto U \to W, Y \mapsto U \to W, X \mapsto U \to W]$
- $\{\textbf{int} \equiv \textbf{int} \to Y\}$
  Not unifiable
- $\{Y \equiv \textbf{int} \to Y\}$
  Not unifiable
- $\{\}$

Examples

- $\{X \equiv \textbf{int}, Y \equiv X \to X\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int} \to \textbf{int}]$
- $\{\textbf{int} \to \textbf{int} \equiv X \to Y\}$
  $[X \mapsto \textbf{int}, Y \mapsto \textbf{int}]$
- $\{X \to Y \equiv Y \to Z, Z \equiv U \to W\}$
  $[X \mapsto U \to W, Y \mapsto U \to W, X \mapsto U \to W]$
- $\{\textbf{int} \equiv \textbf{int} \to Y\}$
  Not unifiable
- $\{Y \equiv \textbf{int} \to Y\}$
  Not unifiable
- $\{\}$
  $[]$

Properties of the algorithm *unify*

1. *unify*(C) halts, either by failing or by returning a substitution for all C;
2. If *unify*$(C) = \sigma$, then σ is a unifier for C;
3. if δ is a unifier for C, then *unify*$(C) = \sigma$ with $\sigma \sqsubseteq \delta$.

Proving 1. termination

Define a lexicographic measure on
a constraint set C, called *degree*:
pair (m, n) where
m is the number of distinct type variables in C and
n is the total size of the types in C.

See that each recursive call has a smaller degree.

Proving 2. unifier

Induction on the number of recursive calls in the computation of *unify*(C).

Variable cases depend on observation:
If σ unifies $[X \mapsto \tau]D$, then $\sigma \circ [X \mapsto \tau]$ unifies $\{X \equiv \tau\} \cup D$ for any constraint set D.

Proving 3. principal

Again, induction on the number of recursive calls in the computation of *unify*(C).

- If C is empty, then *unify*(C) immediately returns the trivial substitution []; since $\delta = \delta \circ []$, we have $[] \sqsubseteq \delta$ as required.
- If C is non-empty, then *unify*(C) chooses some constraint $\tau \equiv \tau'$ and continues on the shape. (See TAPL.)

Let-Polymorphism