CS 476 – Programming Language Design

William Mansky
Constraint-Based Type Inference

• Now we have to solve the constraints
  
  \[
  \text{let unify } (c : \text{constraints}) : (\text{ident } \to \text{typ option}) = \ldots
  \]

• Unification produces a substitution of types for type variables

  \[
  \text{unify } \{ \tau_3 = \text{int}, \tau_4 = \text{int}, \tau_1 = \tau_2 \to \tau_3, \tau_1 = \text{int } \to \text{int } \to \tau_4 \} = \{
  \tau_3 = \text{int}, \tau_4 = \text{int}, \tau_1 = \text{int } \to \text{int}, \tau_2 = \text{int}\}
  \]

let type_of (gamma : context) (e : exp) =
  let (t, c) = get_constraints gamma e in
  let s = unify c in apply_subst s t
Unification

• Input: a set of *constraints* of the form $S = T$, where $S$ and $T$ are types with type variables in them

• Output: a *substitution*, a map from type variables to types (which still may have variables in them)

• The output substitution $\sigma$ should solve all the constraints: for each $S = T$ in the input, $[\sigma]S$ is exactly the same as $[\sigma]T$
Unification

• Now we have to solve the constraints
  \[
  \text{let unify (c : constraints) : (ident -> typ option) = ...}
  \]

• Unification produces a substitution of types for type variables
  \[
  \text{unify } \{\tau_3 = \text{int}, \tau_4 = \text{int}, \tau_1 = \tau_2 \rightarrow \tau_3, \tau_1 = \text{int} \rightarrow \tau_4\} = ...
  \]

• Exercise: How would you solve this unification problem? How would you figure out the values of all the type variables?
The Unification Algorithm

• Pick a constraint $S = T$ from the current set $C$
• Apply one of the following rules, as appropriate:
  1. Discard
  2. Substitute left
  3. Substitute right
  4. Decompose
• Update the constraint set $C$ and the substitution $\sigma$ accordingly
• Repeat on the remaining constraints
The Unification Algorithm: Discard

• Applies when the constraint is of the form $T = T$
• Action: remove the constraint from $C$, while leaving $\sigma$ and the rest of $C$ unchanged

$C$: \{int = int, \tau_1 = \tau_2, ... \}$
$\sigma$: \{\tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \mapsto \tau_6, ... \}$
The Unification Algorithm: Discard

• Applies when the constraint is of the form $T = T$

• Action: remove the constraint from $C$, while leaving $\sigma$ and the rest of $C$ unchanged

$C: \{\text{int} = \text{int}, \tau_1 = \tau_2, \ldots \} \Rightarrow \{\tau_1 = \tau_2, \ldots \}$

$\sigma: \{\tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \mapsto \tau_6, \ldots \}$
The Unification Algorithm: Substitute (L)

• Applies when the constraint is of the form $x = T$
• Action: add \( \{x \mapsto T\} \) to \( \sigma \), and apply it to the rest of \( \sigma \) and \( C \)

\[
C: \{\tau_5 \equiv \text{bool}, \tau_1 \equiv \text{int} \rightarrow \tau_5, \ldots\}
\]
\[
\sigma: \{\tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \rightarrow \tau_6, \ldots\}
\]
The Unification Algorithm: Substitute (L)

• Applies when the constraint is of the form $x = T$
• Action: add $\{x \mapsto T\}$ to $\sigma$, and apply it to the rest of $\sigma$ and $C$

$C: \{\tau_5 = \text{bool}, \tau_1 = \text{int} \rightarrow \tau_5, \ldots \} \Rightarrow \{\tau_1 = \text{int} \rightarrow \text{bool}, \ldots \}$

$\sigma: \{\tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \rightarrow \tau_6, \ldots \} \Rightarrow \{\tau_5 \mapsto \text{bool}, \tau_4 \mapsto \text{bool} \rightarrow \tau_6, \ldots \}$
The Unification Algorithm: Substitute (L)

• Applies when the constraint is of the form $x = T$
• Action: add $\{x \mapsto T\}$ to $\sigma$, and apply it to the rest of $\sigma$ and $C$

$C: \{\tau_5 = \tau \rightarrow \tau_5, \tau_1 = \text{int} \rightarrow \tau_5, \ldots \}$

$\sigma: \{\tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \rightarrow \tau_6, \ldots \}$

• “Occurs check”: $x$ must not be free in $T$
The Unification Algorithm: Substitute (L)

• Applies when the constraint is of the form $x = T$
• Action: add $\{x \mapsto T\}$ to $\sigma$, and apply it to the rest of $\sigma$ and $C$

\[ C: \{\tau_5 = \tau \rightarrow \tau_5, \tau_1 = \text{int} \rightarrow \tau_5, \ldots \} \Rightarrow \text{fail} \]
\[ \sigma: \{\tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \rightarrow \tau_6, \ldots \} \]

• “Occurs check”: $x$ must not be free in $T$
The Unification Algorithm: Substitute (R)

• Applies when the constraint is of the form $T = x$
• Action: add \( \{x \mapsto T\} \) to \( \sigma \), and apply it to the rest of \( \sigma \) and \( C \)

\begin{align*}
C & : \{ \text{bool} = \tau_5, \tau_1 = \text{int} \to \tau_5, \ldots \} \Rightarrow \{ \tau_1 = \text{int} \to \text{bool}, \ldots \} \\
\sigma & : \{ \tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \to \tau_6, \ldots \} \Rightarrow \{ \tau_5 \mapsto \text{bool}, \tau_4 \mapsto \text{bool} \to \tau_6, \ldots \} \\
\end{align*}

• “Occurs check”: \( x \) must not be free in \( T \)
The Unification Algorithm: Decompose

• Applies when the constraint is of the form
  \[ T(\tau_1, ... \tau_n) = T(\nu_1, ..., \nu_n) \]
• Action: add \( \tau_1 = \nu_1, ..., \tau_n = \nu_n \) to \( C \)

\[ C: \{ \tau_6 \rightarrow \tau_2 = \tau_5 \rightarrow \text{int}, \tau_1 = \text{int} \rightarrow \tau_5, ... \} \]
\[ \sigma: \{ \tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \rightarrow \tau_6, ... \} \]
The Unification Algorithm: Decompose

- Applies when the constraint is of the form
  \[ T(\tau_1, \ldots, \tau_n) = T(\nu_1, \ldots, \nu_n) \]
- Action: add \( \tau_1 = \nu_1, \ldots, \tau_n = \nu_n \) to \( C \)

\[ C: \{ \tau_6 \rightarrow \tau_2 = \tau_5 \rightarrow \text{int}, \tau_1 = \text{int} \rightarrow \tau_5, \ldots \} \Rightarrow \]
\[ \{ \tau_6 = \tau_5, \tau_2 = \text{int}, \tau_1 = \text{int} \rightarrow \tau_5, \ldots \} \]

\[ \sigma: \{ \tau_3 \leftrightarrow \text{int}, \tau_4 \leftrightarrow \tau_5 \rightarrow \tau_6, \ldots \} \]
The Unification Algorithm: Decompose

• Applies when the constraint is of the form
  \[ T(\tau_1, \ldots, \tau_n) = T(\nu_1, \ldots, \nu_n) \]

• Action: add \( \tau_1 = \nu_1, \ldots, \tau_n = \nu_n \) to \( C \)

\[ C: \{ \tau_6 \rightarrow \tau_2 = (\tau_5, \text{int}), \tau_1 = \text{int} \rightarrow \tau_5, \ldots \} \]

\[ \sigma: \{ \tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \rightarrow \tau_6, \ldots \} \]
The Unification Algorithm: Decompose

• Applies when the constraint is of the form
  \[ T(\tau_1, \ldots, \tau_n) = T(\nu_1, \ldots, \nu_n) \]

• Action: add \( \tau_1 = \nu_1, \ldots, \tau_n = \nu_n \) to \( C \)

\[ C: \{ \tau_6 \rightarrow \tau_2 = (\tau_5, \text{int}), \tau_1 = \text{int} \rightarrow \tau_5, \ldots \} \Rightarrow \text{fail} \]

\[ \sigma: \{ \tau_3 \mapsto \text{int}, \tau_4 \mapsto \tau_5 \rightarrow \tau_6, \ldots \} \]

• If the constructors or number of arguments are different, no solution exists
The Unification Algorithm

• Pick a constraint $S = T$ from the current set $C$
• Apply one of the following rules, as appropriate:
  1. Discard
  2. Substitute left
  3. Substitute right
  4. Decompose
• Update the constraint set $C$ and the substitution $\sigma$ accordingly
• Repeat on the remaining constraints
• When finished, $\sigma$ will unify all the original constraints
Constraint-Based Type Inference

• Step 1: gather constraints, outputs pair \((\tau, C)\) such that if \(C\) can be solved, \(\tau\) is the type of the expression

• Step 2: unify constraints \(C\), obtain solving substitution \(\sigma\)

• Step 3: apply \(\sigma\) to \(\tau\) to get the type of the expression

let type_of (gamma : context) (e : exp) =
    let (t, c) = get_constraints gamma e in
    let s = unify c in apply_subst s t
Constraint-Based Type Inference: Rules

\[(n \text{ is a number})\]
\[\Gamma \vdash n : \text{int} | \{\}\]
\[\Gamma(x) = \tau\]
\[\Gamma \vdash x : \tau | \{\}\]

\[\Gamma(l_1 : \tau_1 | C_1) \quad \Gamma(l_2 : \tau_2 | C_2)\]
\[\Gamma \vdash l_1 + l_2 : \text{int} | \{\tau_1 = \text{int}, \tau_2 = \text{int}\} \cup C_1 \cup C_2\]

\[\Gamma[x \mapsto \tau_1] \vdash l : \tau_2 | C \quad \tau_1 \text{ fresh}\]
\[\Gamma \vdash \text{fun } x \rightarrow l : \tau_1 \rightarrow \tau_2 | C\]
The Unification Algorithm

• Pick a constraint \( S = T \) from the current set \( C \)
• Apply one of the following rules, as appropriate:
  1. Discard
  2. Substitute left
  3. Substitute right
  4. Decompose
• Update the constraint set \( C \) and the substitution \( \sigma \) accordingly
• Repeat on the remaining constraints
• When finished, \( \sigma \) will unify all the original constraints
Constraint-Based Type Inference: Example

\{
\} ⊢ (fun f → fun x → f x + f 3) : \tau_1 \to \tau_2 \to \text{int} \mid C

C = \{\tau_3 = \text{int}, \tau_4 = \text{int}, \tau_1 = \tau_2 \to \tau_3, \tau_1 = \text{int} \to \tau_4\}
Constraint-Based Type Inference: Example

\[
\{\} \vdash (\text{fun } f \to \text{fun } x \to f \, x + f \, 3) : \tau_1 \to \tau_2 \to \text{int} \mid C
\]

\[
C = \{\tau_4 = \text{int}, \tau_1 = \tau_2 \to \tau_3, \tau_1 = \text{int} \to \tau_4\}
\]

\[
\sigma = \{\tau_3 \mapsto \text{int}\}
\]
Constraint-Based Type Inference: Example

\[ \{} \vdash (\text{fun } f \to \text{fun } x \to f \ x + f \ 3) : \tau_1 \to \tau_2 \to \text{int} \mid C \]

\[ C = \{ \tau_4 = \text{int}, \tau_1 = \tau_2 \to \text{int}, \tau_1 = \text{int} \to \tau_4 \} \]

\[ \sigma = \{ \tau_3 \mapsto \text{int} \} \]
Constraint-Based Type Inference: Example

\[
\begin{align*}
\{\} \vdash (\text{fun } f \rightarrow \text{fun } x \rightarrow f\ x + f\ 3) : \tau_1 \rightarrow \tau_2 \rightarrow \text{int} & \mid C \\
C &= \{\tau_1 = \tau_2 \rightarrow \text{int}, \tau_1 = \text{int} \rightarrow \text{int}\} \\
\sigma &= \{\tau_3 \leftrightarrow \text{int}, \tau_4 \leftrightarrow \text{int}\}
\end{align*}
\]
Constraint-Based Type Inference: Example

\[
\begin{align*}
\{ \} \vdash (\text{fun } f \rightarrow \text{fun } x \rightarrow f \; x + f \; 3) : \tau_1 \rightarrow \tau_2 \rightarrow \text{int} | C \\
C &= \{\tau_2 \rightarrow \text{int} = \text{int} \rightarrow \text{int}\} \\
\sigma &= \{\tau_3 \leftrightarrow \text{int}, \tau_4 \leftrightarrow \text{int}, \tau_1 \leftrightarrow \tau_2 \rightarrow \text{int}\}
\end{align*}
\]
Constraint-Based Type Inference: Example

\[ \{ \} \vdash (\text{fun } f \to \text{fun } x \to f \ x + f \ 3) : \tau_1 \to \tau_2 \to \text{int} \mid C \]

\[ C = \{ \tau_2 = \text{int}, \text{int} = \text{int} \} \]

\[ \sigma = \{ \tau_3 \mapsto \text{int}, \tau_4 \mapsto \text{int}, \tau_1 \mapsto \tau_2 \to \text{int} \} \]
Constraint-Based Type Inference: Example

\[
\{ \} \vdash ( \text{fun } f \rightarrow \text{fun } x \rightarrow f \, x + f \, 3 ) : \tau_1 \rightarrow \tau_2 \rightarrow \text{int} \mid C
\]

\[C = \{ \text{int} = \text{int} \}\]

\[\sigma = \{ \tau_3 \leftrightarrow \text{int}, \tau_4 \leftrightarrow \text{int}, \tau_1 \leftrightarrow \text{int} \rightarrow \text{int}, \tau_2 \leftrightarrow \text{int} \}\]
Constraint-Based Type Inference: Example

\[
\{\} \vdash (\text{fun } f \rightarrow \text{fun } x \rightarrow f \ x + f \ 3) : \tau_1 \rightarrow \tau_2 \rightarrow \text{int} \mid C
\]

\[
C = \{
\}
\]

\[
\sigma = \{\tau_3 \leftrightarrow \text{int}, \tau_4 \leftrightarrow \text{int}, \tau_1 \leftrightarrow \text{int} \rightarrow \text{int}, \tau_2 \leftrightarrow \text{int}\}
\]

\[
[\sigma](\tau_1 \rightarrow \tau_2 \rightarrow \text{int}) = (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \rightarrow \text{int}
\]

\[
\{\} \vdash (\text{fun } f \rightarrow \text{fun } x \rightarrow f \ x + f \ 3) : (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \rightarrow \text{int}
\]
Questions?

Top