Type Inference

Lecture 3a
Review of last class ...

- Programming language
  \[ e = x \mid \lambda x : \tau . e \mid e \ e \mid i \mid e + e \mid \text{if } e \ e \ e \]

- Semantics
  \[ (\lambda x : \tau . e) \ e' , \ \beta \ e[x \leftarrow e'] \]

- Type language
  \[ \tau = \alpha \mid \tau \rightarrow \tau \mid \text{int} \]

- Some programs have types
  - (according to type rules)
    \[ A \vdash e : \tau \]

- Soundness Theorem: Evaluation preserves types
  - If \[ A \vdash e : \tau \] and \[ e \ , \ \beta \ d \], then \[ A \vdash d : \tau \]

- Untypable program: \[ 3 + (\lambda x : \tau . e) \]

13 Jan 2004

290G - Lecture 3a
Challenge Problem

• Consider the program
  
  - if 1 (3 + 2) (4 3)

• What is its execution behavior?

• What is its type?
Type Inference

• The *type erasure* of $e$ is $e$ with all type information removed
  - the untyped term

• Is an untyped term the erasure of some simply typed term?
  - And what are the types?

• This is a *type inference* problem.
  - We must infer, rather than check, the types.
Outline

• We develop the inference algorithm in steps:
  - recast the type rules in an equivalent form
  - show typing in the new rules reduces to a constraint satisfaction problem
  - show the constraint problem is solvable
    • In this case, via term unification.

• We will use this outline again.
### The Problems

- There are three problems in developing an algorithm
  - How do we construct the right type assumptions?
  - How do we ensure types match in applications?
  - How do we ensure types match in if-then-else?

\[
\begin{align*}
A(x) &= \tau & A \vdash x : \tau \quad & A, x : \tau \vdash e : \tau' & A \vdash e_1 : \tau \\
A \vdash x : \tau & A \vdash \lambda x : \tau . e : \tau \rightarrow \tau' & A \vdash e_2 : \tau & A \vdash e_1 e_2 : \tau' \\
A \vdash e_1 : \text{int} & A \vdash e_2 : \text{int} & A \vdash e_3 : \tau \\
A \vdash i : \text{int} & A \vdash e_1 + e_2 : \text{int} & A \vdash \text{if } e_1 e_2 e_3 : \tau
\end{align*}
\]
New Rules

\[ A \vdash e_1 : \tau_1 \]
\[ A \vdash e_2 : \tau_2 \]
\[ A \vdash x : \alpha_x \]
\[ A, x : \alpha_x \vdash e : \tau \]
\[ A \vdash \lambda x.e : \alpha_x \rightarrow \tau \]
\[ A \vdash e_1 \cdot e_2 : \beta \]

\[ A \vdash i : \text{int} \]
\[ A \vdash e_1 + e_2 : \text{int} \]
\[ A \vdash \text{if } e_1 \ e_2 \ e_3 : \tau_2 \]

- Sidestep the problems by introducing explicit unknowns and constraints
New Rules

\[
\begin{align*}
A(x) &= \alpha_x \\
\frac{A \vdash x : \alpha_x}{A \vdash \lambda x. e : \alpha_x \rightarrow \tau} & \quad \frac{A, x : \alpha_x \vdash e : \tau}{A \vdash \lambda x. e : \alpha_x \rightarrow \tau}
\end{align*}
\]

\[
\begin{align*}
\tau_1 &= \tau_2 \rightarrow \beta \\
\frac{A \vdash e_1 : \tau_1 \quad A \vdash e_2 : \tau_2}{A \vdash e_1 e_2 : \beta}
\end{align*}
\]

- Type assumption for variable \( x \) is a fresh variable \( \alpha_x \)
New Rules

\[
\begin{align*}
A(x) &= \alpha_x \\
\vdash x : \alpha_x & \quad A, x : \alpha_x \vdash e : \tau \\
\end{align*}
\]

\[
\begin{align*}
\vdash \lambda x.e : \alpha_x \rightarrow \tau & \quad A \vdash e_1 : \tau_1 \\
& \quad A \vdash e_2 : \tau_2 \\
\Rightarrow \tau_1 = \tau_2 \rightarrow \beta & \quad A \vdash e_1 \ e_2 : \beta \\
\end{align*}
\]

\[
\begin{align*}
A \vdash e_1 : \tau_1 & \quad A \vdash e_2 : \tau_2 \\
A \vdash e_3 : \tau_3 & \quad A \vdash e_4 : \tau_4 \\
\Rightarrow \tau_1 = \tau_2 = \text{int} & \quad A \vdash e_1 + e_2 : \text{int} \\
& \quad A \vdash \text{if } e_1 \ e_2 \ e_3 : \tau_2 \\
\end{align*}
\]

- Equality conditions represented as side constraints
New Rules

- Hypotheses are all arbitrary
  - Can always complete a derivation, pending constraint resolution
Notes

• The introduction of unknowns and constraints works only because the shape of the proof is already known.
  - This tells us where to put the constraints and unknowns.

• The revised rules are trivial to implement, except for handling the constraints.
Solutions of Constraints

• The new rules generate a system of type equations.

• Intuitively, a solution of these equations gives a derivation.

• A solution is a substitution $\text{Vars}, \text{Types}$ such that the equations are satisfied.
Example

\[ \alpha = \beta \rightarrow \gamma \]
\[ \alpha = \gamma \rightarrow \beta \]
\[ \beta = \text{int} \]

- A solution is

\[ \alpha = \text{int} \rightarrow \text{int}, \quad \beta = \text{int}, \quad \gamma = \text{int} \]
Solving Type Equations

• Term equations are a unification problem.
  - Solvable in near-linear time using a union-find based algorithm.

• No solutions $\alpha = T[\alpha]$ are permitted
  - The occurs check.
  - The check is omitted if we allow infinite types.
Unification

- Close constraints under four rules.
- If no inconsistency occurs check violation found, system has a solution.
  - \( \text{int} = x, \ y \)

\[
\begin{align*}
S \cup \{\alpha = \alpha\} & \quad \Rightarrow \quad S \\
S \cup \{\alpha = \tau\} & \quad \Rightarrow \quad S[\tau/\alpha] \cup \{\alpha \equiv \tau\} \\
S \cup \{\tau_1 \rightarrow \tau_2 = \tau_3 \rightarrow \tau_4\} & \quad \Rightarrow \quad S \cup \{\tau_1 = \tau_3, \tau_2 = \tau_4\} \\
S \cup \{\text{int} = \text{int}\} & \quad \Rightarrow \quad S
\end{align*}
\]
Syntax

• We distinguish solved equations $\alpha \cong \tau$
• Each rule manipulates only unsolved equations.

\[
S \cup \{\alpha = \alpha\} \quad \Rightarrow \quad S
\]
\[
S \cup \{\alpha = \tau\} \quad \Rightarrow \quad S[\tau/\alpha] \cup \{\alpha \cong \tau\}
\]
\[
S \cup \{\tau_1 \rightarrow \tau_2 = \tau_3 \rightarrow \tau_4\} \quad \Rightarrow \quad S \cup \{\tau_1 = \tau_3, \tau_2 = \tau_4\}
\]
\[
S \cup \{\text{int} = \text{int}\} \quad \Rightarrow \quad S
\]
Rules 1 and 4

- Rules 1 and 4 eliminate trivial constraints.
- Rule 1 is applied in preference to rule 2
  - the only such possible conflict

\[
S \cup \{\alpha = \alpha\} \quad \Rightarrow \quad S \\
S \cup \{\alpha = \tau\} \quad \Rightarrow \quad S[\tau / \alpha] \cup \{\alpha \equiv \tau\} \\
S \cup \{\tau_1 \rightarrow \tau_2 = \tau_3 \rightarrow \tau_4\} \quad \Rightarrow \quad S \cup \{\tau_1 = \tau_3, \tau_2 = \tau_4\} \\
S \cup \{\text{int} = \text{int}\} \quad \Rightarrow \quad S
\]
Rule 2

• Rule 2 eliminates a variable from all equations but one (which is marked as solved).
  - Note the variable is eliminated from all unsolved as well as solved equations

\[
S \cup \{\alpha = \alpha\} \quad \Rightarrow \quad S \\
S \cup \{\alpha = \tau\} \quad \Rightarrow \quad S[\tau/\alpha] \cup \{\alpha \equiv \tau\} \\
S \cup \{\tau_1 \rightarrow \tau_2 = \tau_3 \rightarrow \tau_4\} \quad \Rightarrow \quad S \cup \{\tau_1 = \tau_3, \tau_2 = \tau_4\} \\
S \cup \{\text{int} = \text{int}\} \quad \Rightarrow \quad S
\]
Rule 3

• Rule 3 applies structural equality to non-trivial terms.

\[
S \cup \{\alpha = \alpha\} \quad \Rightarrow \quad S
\]
\[
S \cup \{\alpha = \tau\} \quad \Rightarrow \quad S[\tau / \alpha] \cup \{\alpha \cong \tau\}
\]
\[
S \cup \{\tau_1 \to \tau_2 = \tau_3 \to \tau_4\} \quad \Rightarrow \quad S \cup \{\tau_1 = \tau_3, \tau_2 = \tau_4\}
\]
\[
S \cup \{\text{int} = \text{int}\} \quad \Rightarrow \quad S
\]
Correctness

- Each rule preserves the set of solutions.
  - Rules 1 and 4 eliminate trivial constraints.
  - Rule 2 substitutes equals for equals.
  - Rule 3 is the definition of equality on function types.

\[
\begin{align*}
S \cup \{\alpha = \alpha\} &\quad \Rightarrow \quad S \\
S \cup \{\alpha = \tau\} &\quad \Rightarrow \quad S[\tau/\alpha] \cup \{\alpha \equiv \tau\} \\
S \cup \{\tau_1 \rightarrow \tau_2 = \tau_3 \rightarrow \tau_4\} &\quad \Rightarrow \quad S \cup \{\tau_1 = \tau_3, \tau_2 = \tau_4\} \\
S \cup \{\text{int} = \text{int}\} &\quad \Rightarrow \quad S
\end{align*}
\]
Termination

• Rules 1 and 4 reduce the number of equations.
• Rule 2 reduces the number of variables in unsolved equations.
• Rule 3 decreases the height of terms.

\[
\begin{align*}
S \cup \{\alpha = \alpha\} & \implies S \\
S \cup \{\alpha = \tau\} & \implies S[\tau/\alpha] \cup \{\alpha \equiv \tau\} \\
S \cup \{\tau_1 \rightarrow \tau_2 = \tau_3 \rightarrow \tau_4\} & \implies S \cup \{\tau_1 = \tau_3, \tau_2 = \tau_4\} \\
S \cup \{\text{int} = \text{int}\} & \implies S
\end{align*}
\]
Termination (Cont.)

- Rules 1, 3, and 4 always terminate
  - because terms must eventually be reduced to height 0.
- Eventually rule 2 is applied, reducing the number of variables.

\[
\begin{align*}
S \cup \{\alpha = \alpha\} & \quad \Rightarrow \quad S \\
S \cup \{\alpha = \tau\} & \quad \Rightarrow \quad S[\tau/\alpha] \cup \{\alpha \equiv \tau\} \\
S \cup \{\tau_1 \rightarrow \tau_2 = \tau_3 \rightarrow \tau_4\} & \quad \Rightarrow \quad S \cup \{\tau_1 = \tau_3, \tau_2 = \tau_4\} \\
S \cup \{\text{int} = \text{int}\} & \quad \Rightarrow \quad S
\end{align*}
\]
A Nitpick

• We really need one more operation

• $\tau = \alpha$ should be flipped to $\alpha = \tau$ if $\tau$ is not a variable
  - Needed to ensure rule 2 applies whenever possible.
  - We just assume equations are maintained in this “normal form”.

Solutions

- The final system is a solution.
  - There is one equation $\alpha \equiv \tau$ for each variable.
  - This is a substitution with all the solutions of the original system

- Must also perform occurs check to guarantee there are no recursive constraints.
Example

rewrites  $\alpha = \beta \rightarrow \gamma$, $\alpha = \gamma \rightarrow \beta$, $\beta = \text{int}$
An Example of Failure

\[ \alpha = \beta \rightarrow \gamma, \ \alpha = \gamma \rightarrow (\beta \rightarrow \beta), \ \beta = \text{int} \]
Notes

• The algorithm produces the most general unifier of the equations.
  - All solutions are preserved.

• Less general solutions are always substitution instances of the most general solution.
Next Time …

- Lackwit: A Program Understanding Tool Based on Type Inference
- Type-Based Race Detection for Java