Chapter 9.
Type Inference

Prof. Jaeseung Choi
Dept. of Computer Science and Engineering
Sogang University

Review: Typing Rules for F-

$$\frac{}{\Gamma \vdash n : \text{int}} \qquad \frac{}{\Gamma \vdash \textbf{true} : \text{bool}} \qquad \frac{}{\Gamma \vdash \textbf{false} : \text{bool}} \qquad \frac{}{\Gamma \vdash x : \Gamma(x)}$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \qquad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 < e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : t \quad \Gamma \vdash e_3 : t}{\Gamma \vdash \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 : t} \qquad \frac{\Gamma \vdash e_1 : t_1 \quad \Gamma[x \mapsto \mathrm{t}_1] \vdash e_2 : t_2}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : t_2}$$

$$\frac{\Gamma[x \mapsto t_a] \vdash e_1 : t_r \quad \Gamma[f \mapsto (t_a \to t_r)] \vdash e_2 : t_2}{\Gamma \vdash \textbf{let}\ f\ x = e_1\ \textbf{in}\ e_2 : t_2} \qquad \frac{\Gamma \vdash e_1 : t_a \to t_r \quad \Gamma \vdash e_2 : t_a}{\Gamma \vdash e_1\ \ e_2 : t_r}$$

Review: Domains for Type

- **The set of types (T) is defined as follow**
  - We will use $\boldsymbol{t}$ to denote it ($\boldsymbol{t} \in \boldsymbol{T}$)
  - And $\boldsymbol{\tau}$ below denotes a type variable ($\boldsymbol{\tau} \in \boldsymbol{TyVar} = \boldsymbol{String}$)

$$\frac{}{\text{bool} \in T} \qquad \frac{}{\text{int} \in T} \qquad \frac{}{\tau \ \in T}$$

$$\frac{t_1 \in T \quad t_2 \in T}{t_1 \to t_2 \in T}$$

- **Type environment is a mapping from variable to type**
  - We will use $\boldsymbol{\Gamma}$ to denote it ($\boldsymbol{\Gamma} \in \boldsymbol{TyEnv} = \boldsymbol{Var} \to \boldsymbol{T}$)

Type Inference Outline

- **Step #1: Generate *type equations* (also known as *type constraints*) from the given program**
  - Introduce a **type variable** for each sub-expression that forms the program, and construct equations with them
- **Step #2: Solve the generated equations**
  - If a solution is found, we can decide the type of the program
  - If such solution does not exist (i.e., some contradiction occurs), it means that the program cannot be typed in our type system

Brief Overview

- **Let's first use this example to sketch the whole process**
  - First, introduce type variables (τ_*) and construct type equations
  - Then, find the solution that satisfies the equations

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

Type Equations

$$\begin{aligned} \tau_r &= \text{int} \wedge \\ \text{int} &= \text{int} \wedge \\ \tau_g &= \tau_a \rightarrow \text{int} \wedge \\ \tau_a &= \text{int} \wedge \\ \tau_p &= \tau_g \rightarrow \tau_r \end{aligned}$$

Substitution

$$\begin{aligned} \tau_r &= \text{int} \\ \tau_g &= \text{int} \rightarrow \text{int} \\ \tau_a &= \text{int} \\ \tau_p &= (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \end{aligned}$$

Generating Type Equations

- **Let's define function $Gen(\Gamma, e, t)$ in *pattern match* style**
  - Generate the constraint to make e have type $t \in T$ under Γ
  - Definition of $Gen(\ )$ must be in accordance with the typing rules
  - Generated type equations are connected with conjunction
    - Have the form of $(t_1 = t_1') \land (t_2 = t_2') \land \cdots \land (t_n = t_n')$

$$\frac{}{\Gamma \vdash n : \text{int}} \qquad Gen(\Gamma, n, t) = (t = \text{int})$$

$$\frac{}{\Gamma \vdash \textbf{true} : \text{bool}} \qquad Gen(\Gamma, \textbf{true}, t) = (t = \text{bool})$$

$$\frac{}{\Gamma \vdash \textbf{false} : \text{bool}} \qquad Gen(\Gamma, \textbf{false}, t) = (t = \text{bool})$$

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \qquad Gen(\Gamma, x, t) = (t = \Gamma(x))$$

Generating Type Equations

- **Let's define function $\boldsymbol{Gen}(\boldsymbol{\Gamma}, \boldsymbol{e}, \boldsymbol{t})$ in *pattern match* style**
  - Generate the constraint to make $\boldsymbol{e}$ have type $\boldsymbol{t} \in \boldsymbol{T}$ under $\boldsymbol{\Gamma}$
  - Definition of $\boldsymbol{Gen}(\,)$ must be in accordance with the typing rules
  - Generated type equations are connected with conjunction
    - Have the form of $(\boldsymbol{t_1} = \boldsymbol{t'_1}) \wedge (\boldsymbol{t_2} = \boldsymbol{t'_2}) \wedge \cdots \wedge (\boldsymbol{t_n} = \boldsymbol{t'_n})$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}}$$

$$\begin{aligned}&\boldsymbol{Gen}(\Gamma, e_1 + e_2, t) = \\ &\quad (t = \text{int}) \wedge \boldsymbol{Gen}(\Gamma, e_1, \text{int}) \wedge \boldsymbol{Gen}(\Gamma, e_2, \text{int})\end{aligned}$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 < e_2 : \text{bool}}$$

$$\begin{aligned}&\boldsymbol{Gen}(\Gamma, e_1 < e_2, t) = \\ &\quad (t = \text{bool}) \wedge \boldsymbol{Gen}(\Gamma, e_1, \text{int}) \wedge \boldsymbol{Gen}(\Gamma, e_2, \text{int})\end{aligned}$$

Generating Type Equations

- **Let's define function $\boldsymbol{Gen(\Gamma, e, t)}$ in *pattern match* style**
  - Generate the constraint to make $\boldsymbol{e}$ have type $\boldsymbol{t} \in \boldsymbol{T}$ under $\boldsymbol{\Gamma}$
  - Definition of $\boldsymbol{Gen}(\,)$ must be in accordance with the typing rules
  - Generated type equations are connected with conjunction
    - Have the form of $(\boldsymbol{t_1} = \boldsymbol{t'_1}) \wedge (\boldsymbol{t_2} = \boldsymbol{t'_2}) \wedge \cdots \wedge (\boldsymbol{t_n} = \boldsymbol{t'_n})$

$$\frac{\Gamma \vdash e_1 : \mathrm{bool} \quad \Gamma \vdash e_2 : t \quad \Gamma \vdash e_3 : t}{\Gamma \vdash \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 : t}$$

$$\begin{aligned} &\boldsymbol{Gen}(\Gamma, \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3, t) = \\ &\quad \boldsymbol{Gen}(\Gamma, e_1, \mathrm{bool}) \wedge \\ &\quad \boldsymbol{Gen}(\Gamma, e_2, t) \wedge \\ &\quad \boldsymbol{Gen}(\Gamma, e_3, t) \end{aligned}$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma[x \mapsto \mathrm{t}_1] \vdash e_2 : t_2}{\Gamma \vdash \textbf{let } x = e_1 \textbf{ in } e_2 : t_2}$$

$$\begin{aligned} &\boldsymbol{Gen}(\Gamma, \textbf{let } x = e_1 \textbf{ in } e_2, t) = \\ &\quad \boldsymbol{Gen}(\Gamma, e_1, \tau) \wedge \boldsymbol{Gen}(\Gamma[x \mapsto \tau], e_2, t) \end{aligned}$$

(τ is a newly generated type variable)

Generating Type Equations

- **Let's define function $\boldsymbol{Gen(\Gamma, e, t)}$ in *pattern match* style**
  - Generate the constraint to make $\boldsymbol{e}$ have type $\boldsymbol{t} \in \boldsymbol{T}$ under $\boldsymbol{\Gamma}$
  - Definition of $\boldsymbol{Gen}(\,)$ must be in accordance with the typing rules
  - Generated type equations are connected with conjunction
    - Have the form of $(\boldsymbol{t_1} = \boldsymbol{t'_1}) \wedge (\boldsymbol{t_2} = \boldsymbol{t'_2}) \wedge \cdots \wedge (\boldsymbol{t_n} = \boldsymbol{t'_n})$

$$\frac{\Gamma[x \mapsto t_a] \vdash e_1 : t_r \quad \Gamma[f \mapsto (t_a \to t_r)] \vdash e_2 : t_2}{\Gamma \vdash \textbf{let}\ f\ x = e_1\ \textbf{in}\ e_2 : t_2}$$

$$\begin{aligned} &\boldsymbol{Gen}(\Gamma, \textbf{let}\ f\ x = e_1\ \textbf{in}\ e_2, t) = \\ &\quad \boldsymbol{Gen}(\Gamma[x \mapsto \tau_a], e_1, \tau_r) \wedge \\ &\quad \boldsymbol{Gen}(\Gamma[f \mapsto (\tau_\mathrm{a} \to \tau_r)], e_2, t) \end{aligned}$$

($\boldsymbol{\tau_a}$ and $\boldsymbol{\tau_r}$ are new type variables)

$$\frac{\Gamma \vdash e_1 : t_a \to t_r \quad \Gamma \vdash e_2 : t_a}{\Gamma \vdash e_1\ \ e_2 : t_r}$$

$$\begin{aligned} &\boldsymbol{Gen}(\Gamma, e_1\ e_2, t) = \\ &\quad \boldsymbol{Gen}(\Gamma, e_1, \tau_a \to t) \wedge \boldsymbol{Gen}(\Gamma, e_2, \tau_a) \end{aligned}$$

($\boldsymbol{\tau_a}$ is a new type variable)

Exercise #1

- **Let's run $\boldsymbol{Gen(\Gamma, e, t)}$ and see which type equations are generated from the previous example**
  - Start by introducing type variable τ_p for the whole program

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

$$Gen\left(\phi, \mathbf{let}\, f\ g = 1 + g\ 2\ \mathbf{in}\, f, \tau_p\right) = ?$$

Exercise #1

- **Let's run $\boldsymbol{\color{red}{Gen(\Gamma, e, t)}}$ and see which type equations are generated from the previous example**
  - Start by introducing type variable τ_p for the whole program

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

$$Gen\big(\phi, \mathbf{let}\, f\ g = 1 + g\ 2\ \mathbf{in}\, f, \tau_p\big) =$$

$$Gen(\{g \mapsto \tau_g\}, 1 + g\ 2, \tau_r) \wedge Gen(\{f \mapsto (\tau_g \to \tau_r)\}, f, \tau_p) =$$

Exercise #1

- **Let's run $\boldsymbol{Gen(\Gamma, e, t)}$ and see which type equations are generated from the previous example**
  - Start by introducing type variable τ_p for the whole program

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

$$Gen\big(\phi, \textbf{let}\, f\ g = 1 + g\ 2\ \textbf{in}\, f, \tau_p\big) =$$
$$Gen(\{g \mapsto \tau_g\}, 1 + g\ 2, \tau_r) \wedge Gen(\{f \mapsto (\tau_g \to \tau_r)\}, f, \tau_p) =$$
$$(\tau_r = \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, 1, \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, g\ 2, \text{int}) \wedge (\tau_p = \tau_g \to \tau_r) =$$

Exercise #1

- **Let's run $Gen(\Gamma, e, t)$ and see which type equations are generated from the previous example**
  - Start by introducing type variable τ_p for the whole program

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

$$
\begin{aligned}
&Gen\big(\phi, \textbf{let}\ f\ g = 1 + g\ 2\ \textbf{in}\ f, \tau_p\big) = \\
&\quad Gen(\{g \mapsto \tau_g\}, 1 + g\ 2, \tau_r) \land Gen(\{f \mapsto (\tau_g \to \tau_r)\}, f, \tau_p) = \\
&\quad (\tau_r = \text{int}) \land Gen(\{g \mapsto \tau_g\}, 1, \text{int}) \land Gen(\{g \mapsto \tau_g\}, g\ 2, \text{int}) \land (\tau_p = \tau_g \to \tau_r) = \\
&\quad (\tau_r = \text{int}) \land (\text{int} = \text{int}) \land Gen(\{g \mapsto \tau_g\}, g, \tau_a \to \text{int}) \land Gen(\{g \mapsto \tau_g\}, 2, \tau_a) \land \\
&\quad (\tau_p = \tau_g \to \tau_r) =
\end{aligned}
$$

Exercise #1

- **Let's run $\boldsymbol{Gen(\Gamma, e, t)}$ and see which type equations are generated from the previous example**
  - Start by introducing type variable τ_p for the whole program

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

$$Gen\big(\phi, \textbf{let}\, f\ g = 1 + g\ 2\ \textbf{in}\ f, \tau_p\big) =$$

$$Gen(\{g \mapsto \tau_g\}, 1 + g\ 2, \tau_r) \land Gen(\{f \mapsto (\tau_g \to \tau_r)\}, f, \tau_p) =$$

$$(\tau_r = \text{int}) \land Gen(\{g \mapsto \tau_g\}, 1, \text{int}) \land Gen(\{g \mapsto \tau_g\}, g\ 2, \text{int}) \land (\tau_p = \tau_g \to \tau_r) =$$

$$(\tau_r = \text{int}) \land (\text{int} = \text{int}) \land Gen(\{g \mapsto \tau_g\}, g, \tau_a \to \text{int}) \land Gen(\{g \mapsto \tau_g\}, 2, \tau_a) \land (\tau_p = \tau_g \to \tau_r) =$$

$$(\tau_r = \text{int}) \land (\text{int} = \text{int}) \land (\tau_g = \tau_a \to \text{int}) \land (\tau_a = \text{int}) \land (\tau_p = \tau_g \to \tau_r)$$

Exercise #1

- **Let's run $\boldsymbol{Gen(\Gamma, e, t)}$ and see which type equations are generated from the previous example**
  - Start by introducing type variable τ_p for the whole program

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

$$Gen\big(\phi, \mathbf{let}\ f\ g = 1 + g\ 2\ \mathbf{in}\ f, \tau_p\big) =$$
$$Gen(\{g \mapsto \tau_g\}, 1 + g\ 2, \tau_r) \wedge Gen(\{f \mapsto (\tau_g \to \tau_r)\}, f, \tau_p) =$$
$$(\tau_r = \mathrm{int}) \wedge Gen(\{g \mapsto \tau_g\}, 1, \mathrm{int}) \wedge Gen(\{g \mapsto \tau_g\}, g\ 2, \mathrm{int}) \wedge (\tau_p = \tau_g \to \tau_r) =$$
$$(\tau_r = \mathrm{int}) \wedge (\mathrm{int} = \mathrm{int}) \wedge Gen(\{g \mapsto \tau_g\}, g, \tau_a \to \mathrm{int}) \wedge Gen(\{g \mapsto \tau_g\}, 2, \tau_a) \wedge$$
$$(\tau_p = \tau_g \to \tau_r) =$$
$$\boldsymbol{(\tau_r = \mathrm{int}) \wedge (\mathrm{int} = \mathrm{int}) \wedge (\tau_g = \tau_a \to \mathrm{int}) \wedge (\tau_a = \mathrm{int}) \wedge (\tau_p = \tau_g \to \tau_r)}$$

Type Inference Outline

- **Step #1: Generate *type equations* (also known as *type constraints*) from the given program**
  - Introduce a **type variable** for each sub-expression that forms the program, and construct equations with them
- **Step #2: Solve the generated equations**
  - If a solution is found, we can decide the type of the program
  - If such solution does not exist (i.e., some contradiction occurs), it means that the program cannot be typed in our type system

This time, we will discuss with a running example first and then describe the actual algorithm

Running Example

- **We will construct a substitution (mapping from τ to T)**
  - Let's move the equations from the left to right side, one by one
  - The final substitution we obtain is the solution of the equations

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

Type Equations

$$\begin{aligned}
&\tau_r = \text{int} \wedge \\
&\text{int} = \text{int} \wedge \\
&\tau_g = \tau_a \rightarrow \text{int} \wedge \\
&\tau_a = \text{int} \wedge \\
&\tau_p = \tau_g \rightarrow \tau_r
\end{aligned}$$

Substitution

(starts from empty map)

Running Example

- **We will construct a substitution (mapping from τ to T)**
  - Let's move the equations from the left to right side, one by one
  - The final substitution we obtain is the solution of the equations

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

Type Equations

$$\begin{aligned}
\tau_r &= \text{int} \land \\
\text{int} &= \text{int} \land \\
\tau_g &= \tau_a \rightarrow \text{int} \land \\
\tau_a &= \text{int} \land \\
\tau_p &= \tau_g \rightarrow \tau_r
\end{aligned}$$

Substitution

$$\tau_r \mapsto \text{int}$$

Running Example

- **We will construct a substitution (mapping from τ to T)**
  - Let's move the equations from the left to right side, one by one
  - The final substitution we obtain is the solution of the equations

$$\overbrace{\texttt{let f }\underbrace{\texttt{g}}_{\tau_g}\texttt{ = }\underbrace{\texttt{1 + g }\underbrace{\texttt{2}}_{\tau_a}}_{\tau_r}\texttt{ in f}}^{\tau_p}$$

Type Equations

$$\begin{aligned}
&\tau_r = \text{int} \wedge \\
&\text{int} = \text{int} \wedge \\
&\tau_g = \tau_a \rightarrow \text{int} \wedge \\
&\tau_a = \text{int} \wedge \\
&\tau_p = \tau_g \rightarrow \tau_r
\end{aligned}$$

Substitution

$\tau_r \mapsto \text{int}$

$\textbf{int} = \textbf{int}$ is just ignored

Running Example

- **We will construct a substitution (mapping from τ to T)**
  - Let's move the equations from the left to right side, one by one
  - The final substitution we obtain is the solution of the equations

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

Type Equations

$$\tau_r = \text{int} \land$$
$$\text{int} = \text{int} \land$$
$$\tau_g = \tau_a \rightarrow \text{int} \land$$
$$\tau_a = \text{int} \land$$
$$\tau_p = \tau_g \rightarrow \tau_r$$

Substitution

$$\tau_r \mapsto \text{int}$$
$$\tau_g \mapsto \tau_a \rightarrow \text{int}$$

Running Example

- **We will construct a substitution (mapping from τ to T)**
  - Let's move the equations from the left to right side, one by one
  - The final substitution we obtain is the solution of the equations

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

Type Equations

$$\begin{aligned} &\tau_r = \text{int} \wedge \\ &\text{int} = \text{int} \wedge \\ &\tau_g = \tau_a \rightarrow \text{int} \wedge \\ &\tau_a = \text{int} \wedge \\ &\tau_p = \tau_g \rightarrow \tau_r \end{aligned}$$

Substitution

$$\begin{aligned} &\tau_r \mapsto \text{int} \\ &\tau_g \mapsto \text{int} \rightarrow \text{int} \\ &\tau_a \mapsto \text{int} \end{aligned}$$

τ_a replaced with int

Running Example

- **We will construct a substitution (mapping from τ to T)**
  - Let's move the equations from the left to right side, one by one
  - The final substitution we obtain is the solution of the equations

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

Type Equations

$$\begin{aligned} \tau_r &= \mathrm{int} \ \wedge \\ \mathrm{int} &= \mathrm{int} \ \wedge \\ \tau_g &= \tau_a \to \mathrm{int} \ \wedge \\ \tau_a &= \mathrm{int} \ \wedge \\ \tau_p &= \tau_g \to \tau_r \end{aligned}$$

Substitution

$$\begin{aligned} \tau_r &\mapsto \mathrm{int} \\ \tau_g &\mapsto \mathrm{int} \to \mathrm{int} \\ \tau_a &\mapsto \mathrm{int} \\ \tau_p &\mapsto (\mathrm{int} \to \mathrm{int}) \to \mathrm{int} \end{aligned}$$

τ_g replaced with $\mathrm{int} \to \mathrm{int}$

Running Example

- **We will construct a substitution (mapping from τ to T)**
  - Let's move the equations from the left to right side, one by one
  - The final substitution we obtain is the solution of the equations

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

Type Equations

$$\begin{aligned} \tau_r &= \text{int} \wedge \\ \text{int} &= \text{int} \wedge \\ \tau_g &= \tau_a \to \text{int} \wedge \\ \tau_a &= \text{int} \wedge \\ \tau_p &= \tau_g \to \tau_r \end{aligned}$$

Substitution

$$\begin{aligned} \tau_r &\mapsto \text{int} \\ \tau_g &\mapsto \text{int} \to \text{int} \\ \tau_a &\mapsto \text{int} \\ \tau_p &\mapsto (\text{int} \to \text{int}) \to \text{int} \end{aligned}$$

τ_r replaced with int

Running Example

- **We will construct a substitution (mapping from τ to T)**
  - Let's move the equations from the left to right side, one by one
  - The final substitution we obtain is the solution of the equations

$$\overbrace{\texttt{let f } \underbrace{\texttt{g}}_{\tau_g} \texttt{ = } \underbrace{\texttt{1 + g } \underbrace{\texttt{2}}_{\tau_a}}_{\tau_r} \texttt{ in f}}^{\tau_p}$$

Type Equations

$$\begin{aligned}
\tau_r &= \text{int} \wedge \\
\text{int} &= \text{int} \wedge \\
\tau_g &= \tau_a \to \text{int} \wedge \\
\tau_a &= \text{int} \wedge \\
\tau_p &= \tau_g \to \tau_r
\end{aligned}$$

Substitution

$$\begin{aligned}
\tau_r &\mapsto \text{int} \\
\tau_g &\mapsto \text{int} \to \text{int} \\
\tau_a &\mapsto \text{int} \\
\tau_p &\mapsto \underline{(\text{int} \to \text{int}) \to \text{int}}
\end{aligned}$$

This is the inferred type of the program

Application of Substitution

- **Before describing the actual algorithm, let's define a function that applies a substitution to a type ($\boldsymbol{t} \in \boldsymbol{T}$)**
  - Recall that substitution is a mapping from type variable to type ($\boldsymbol{s} \in \boldsymbol{Subst} = \boldsymbol{TyVar} \rightarrow \boldsymbol{T}$)
  - $\boldsymbol{App}(\boldsymbol{s}, \boldsymbol{t})$ returns a new type $\boldsymbol{t}'$ with the substitution applied
    - $\boldsymbol{App}(s, \mathrm{bool}) = \mathrm{bool}$
    - $\boldsymbol{App}(s, \mathrm{int}) = \mathrm{int}$
    - $\boldsymbol{App}(s, \tau) = \begin{cases} \text{If } \tau \in Dom(s) : & s[\tau] \\ \text{Otherwise} : & \tau \end{cases}$
    - $\boldsymbol{App}(s, t_1 \rightarrow t_2) = \boldsymbol{App}(s, t_1) \rightarrow \boldsymbol{App}(s, t_2)$

Application of Substitution

- **Let's consider some examples**
  - Ex) $\boldsymbol{App}(\{\tau_1 \mapsto \text{bool}\}, \tau_2)) = ?$
  - Ex) $\boldsymbol{App}(\{\tau_1 \mapsto \text{int}\}, \text{bool})) = ?$
  - Ex) $\boldsymbol{App}(\{\tau_1 \mapsto \text{bool}, \tau_2 \mapsto \text{int}\}, \tau_2)) = ?$
  - Ex) $\boldsymbol{App}(\{\tau_1 \mapsto \text{int}, \tau_2 \mapsto \text{bool}\}, \text{bool} \to (\tau_1 \to \tau_3)) = ?$

Application of Substitution

- **Let's consider some examples**
  - Ex) $\boldsymbol{App}(\{\tau_1 \mapsto \text{bool}\}, \tau_2)) = \tau_2$
  - Ex) $\boldsymbol{App}(\{\tau_1 \mapsto \text{int}\}, \text{bool})) = \text{bool}$
  - Ex) $\boldsymbol{App}(\{\tau_1 \mapsto \text{bool}, \tau_2 \mapsto \text{int}\}, \tau_2)) = \text{int}$
  - Ex) $\boldsymbol{App}(\{\tau_1 \mapsto \text{int}, \tau_2 \mapsto \text{bool}\}, \text{bool} \to (\tau_1 \to \tau_3)) =$
    $\text{bool} \to (\text{int} \to \tau_3)$

Solving Equations

- **Now, let's define the algorithm to solve type equations**
  - It describes the previous process of constructing substitution
- **We will define two functions: $\boldsymbol{Unify}(\,)$ and $\boldsymbol{Extend}(\,)$**
- **$\boldsymbol{Unify}(\boldsymbol{t_1} = \boldsymbol{t_2}, \boldsymbol{s})$ must merge a new equation $\boldsymbol{t_1} = \boldsymbol{t_2}$ into the current substitution $\boldsymbol{s}$ and return a new substitution**
  - It first applies the current substitution $\boldsymbol{s}$ to $\boldsymbol{t_1}$ and $\boldsymbol{t}_2$, and then calls $\boldsymbol{Extend}(\,)$ with them
  - $\boldsymbol{Unify}(t_1 = t_2, s) = \boldsymbol{Extend}(\boldsymbol{App}(s, t_1) = \boldsymbol{App}(s, t_2), s)$
- **$\boldsymbol{Extend}(\boldsymbol{t_1} = \boldsymbol{t_2}, \boldsymbol{s})$ is the actual function that moves the equation to the substitution**

Solving Equations

- **Let's define $\boldsymbol{Extend}(\boldsymbol{t_1} = \boldsymbol{t_2}, \boldsymbol{s})$ in *pattern match* style**
  - $\boldsymbol{Extend}(\mathrm{int} = \mathrm{int}, s) = s$
  - $\boldsymbol{Extend}(\mathrm{bool} = \mathrm{bool}, s) = s$
  - $\boldsymbol{Extend}\big((t_x \to t_y) = (t'_x \to t'_y), s\big) =$
    $\boldsymbol{Unify}(t_y = t'_y, \boldsymbol{Extend}(t_x = t'_x, s))$
  - $\boldsymbol{Extend}(\tau = t, s) = \boldsymbol{Extend}(t = \tau, s) =$
    $$\begin{cases} \text{If } \boldsymbol{t} \text{ is } \boldsymbol{\tau} : s \\ \text{Else if } \boldsymbol{\tau} \text{ occurs in } \boldsymbol{t} : \text{error} \\ \text{Otherwise} : \boldsymbol{s}'[\tau \mapsto t], \text{ where } \boldsymbol{s}' \text{ is } \boldsymbol{s} \text{ with } \boldsymbol{\tau} \text{ replaced into } \boldsymbol{t} \end{cases}$$
  - $\boldsymbol{Extend}(*, s) = \mathrm{error}$

If both sides are functions

If the left or right side is type variable

Other cases like $\mathrm{int} = \mathrm{bool}$, $\mathrm{int} = (t_x \to t_y)$, …

Solving Equations

- **Let's examine some tricky cases in $\boldsymbol{Extend}(\boldsymbol{t_1} = \boldsymbol{t_2}, \boldsymbol{s})$**
  - $\boldsymbol{Extend}(\tau = t, s) =$

    $$\begin{cases} \text{If } \boldsymbol{t} \text{ is } \boldsymbol{\tau} : s \\ \text{Else if } \boldsymbol{\tau} \text{ occurs in } \boldsymbol{t} : \text{error} \\ \text{Otherwise} : \boldsymbol{s}'[\tau \mapsto t], \ \text{where } \boldsymbol{s}' \text{ is } \boldsymbol{s} \text{ with } \boldsymbol{\tau} \text{ replaced into } \boldsymbol{t} \end{cases}$$

  - Ex) $\boldsymbol{Extend}\left(\tau_1 = \tau_1, \{\tau_f \mapsto (\tau_1 \to \text{int})\}\right) = \{\tau_f \mapsto (\tau_1 \to \text{int})\}$

    (nothing to update)
  - Ex) $\boldsymbol{Extend}\left(\tau_1 = (\text{int} \to \tau_1), \{\tau_f \mapsto (\tau_1 \to \text{int})\}\right) = \textit{type error}$

    (Such form is not supported in current type system)
  - Ex) $\boldsymbol{Extend}\left(\tau_1 = \text{bool}, \{\tau_f \mapsto (\tau_1 \to \text{int})\}\right) =$

    $\{\tau_f \mapsto (\text{bool} \to \text{int}), \tau_1 \mapsto \text{bool}\}$

Solving Equations

- **Let's examine some tricky cases in $\boldsymbol{Extend}(\boldsymbol{t_1} = \boldsymbol{t_2}, \boldsymbol{s})$**
  - $\boldsymbol{Extend}\big((t_x \to t_y) = (t'_x \to t'_y), s\big) =$ $\boldsymbol{Unify}(t_y = t'_y, \boldsymbol{Extend}(t_x = t'_x, s))$
  - Note that it's not $\boldsymbol{Extend}(t_y = t'_y, \boldsymbol{Extend}(t_x = t'_x, s))$
  - Recall the definition of $\boldsymbol{Unify}(\,)$ in the previous page
    - It will apply the substitution obtained from $\boldsymbol{Extend}(t_x = t'_x, s)$
  - Ex) $\boldsymbol{Extend}((\tau_1 \to \tau_2) = (\text{int} \to \tau_1), \{\,\}) = ?$

Solving Equations

- **Let's examine some tricky cases in $\boldsymbol{Extend}(\boldsymbol{t_1} = \boldsymbol{t_2}, \boldsymbol{s})$**
  - $\boldsymbol{Extend}\big((t_x \to t_y) = (t'_x \to t'_y), s\big) =$
    $\boldsymbol{Unify}(t_y = t'_y, \boldsymbol{Extend}(t_x = t'_x, s))$
  - Note that it's not $\boldsymbol{Extend}(t_y = t'_y, \boldsymbol{Extend}(t_x = t'_x, s))$
  - Recall the definition of $\boldsymbol{Unify}(\,)$ in the previous page
    - It will apply the substitution obtained from $\boldsymbol{Extend}(t_x = t'_x, s)$
  - Ex) $\boldsymbol{Extend}((\tau_1 \to \tau_2) = (\mathrm{int} \to \tau_1), \{\,\}) =$
    $\boldsymbol{Unify}(\tau_2 = \tau_1, \boldsymbol{Extend}(\tau_1 = \mathrm{int},\ \{\,\})) =$
    $\boldsymbol{Unify}(\tau_2 = \tau_1, \{\tau_1 \mapsto \mathrm{int}\}) =$
    $\boldsymbol{Extend}(\tau_2 = \mathrm{int}, \{\tau_1 \mapsto \mathrm{int}\}) =$
    $\{\tau_1 \mapsto \mathrm{int}, \tau_2 \mapsto \mathrm{int}\}$

Overall Algorithm

- **Now we can put all the things together and define the overall type inference algorithm as follow**
- **Review the previous running example (Exercise #1) and confirm how this algorithm is applied**
  - In other words, run $\boldsymbol{TypeInfer}$(`let f g = 1 + g 2 in f`)

$\boldsymbol{TypeInfer}(\boldsymbol{E})$ =
  let $(t_1 = t_1') \wedge (t_2 = t_2') \ldots \wedge (t_n = t_n')$ = $\boldsymbol{Gen}(\phi, \boldsymbol{E}, \tau_p)$
  let s_1 = $\boldsymbol{Unify}(t_1 = t_1', \phi)$
  let s_2 = $\boldsymbol{Unify}(t_2 = t_2', s_1)$
  ...
  let s_n = $\boldsymbol{Unify}(t_n = t_n', s_{n-1})$
  $s_n[\tau_p]$ // This is the final type of program $\boldsymbol{E}$

Exercise #2

- $TypeInfer($**`let f g = g 1 in f`**$) = ?$

$$\overbrace{\texttt{let f g = g 1 in f}}^{\tau_p}$$

Exercise #2

- $\boldsymbol{TypeInfer}$(**`let f g = g 1 in f`**)$= ?$

$$\overbrace{\texttt{let f }\underbrace{\texttt{g}}_{\tau_g}\texttt{ = }\underbrace{\texttt{g }\underbrace{\texttt{1}}_{\tau_a}}_{\tau_r}\texttt{ in f}}^{\tau_p}$$

$$Gen\left(\phi, \mathbf{let}\, f\ g = g\ 1\ \mathbf{in}\, f, \tau_p\right) =$$
$$Gen(\{g \mapsto \tau_g\}, g\ 1, \tau_r) \wedge Gen(\{f \mapsto (\tau_g \to \tau_r)\}, f, \tau_p) =$$
$$Gen(\{g \mapsto \tau_g\}, g, \tau_a \to \tau_r) \wedge Gen(\{g \mapsto \tau_g\}, 1, \tau_a) \wedge (\tau_p = \tau_g \to \tau_r) =$$
$$\boldsymbol{(\tau_g = \tau_a \to \tau_r) \wedge (\tau_a = \mathrm{int}) \wedge (\tau_p = \tau_g \to \tau_r)}$$

Exercise #2

- $TypeInfer(\texttt{let f g = g 1 in f}) = ?$

$$\overbrace{\texttt{let f } \underset{\tau_g}{\underline{\texttt{g}}} \texttt{ = } \underset{\tau_r}{\underline{\texttt{g } \underset{\tau_a}{\underline{\texttt{1}}}}} \texttt{ in f}}^{\tau_p}$$

Type Equations

$$\tau_g = (\tau_a \to \tau_r) \ \wedge$$
$$\tau_a = \text{int} \ \wedge$$
$$\tau_p = (\tau_g \to \tau_r)$$

Substitution

$$\tau_g = \text{int} \to \tau_r$$
$$\tau_a = \text{int}$$
$$\tau_p = (\text{int} \to \tau_r) \to \tau_r$$

The solution may contain unresolved type variables, as shown in this case

More Exercises

- **Consider the extended version of `F-` language that supports recursive function and anonymous function**
  - How should we define $\boldsymbol{Gen}()$ for the following cases?

$$\boldsymbol{Gen}(\Gamma, \textbf{let rec } f\ x = e_1 \textbf{ in } e_2, t) = ?$$

$$\boldsymbol{Gen}(\Gamma, \textbf{fun } x \rightarrow e, t) = ?$$

- **Run type inference for various examples**
  - $\boldsymbol{TypeInfer}$`(let f x = x + 1 in f true)`
  - $\boldsymbol{TypeInfer}$`(fun g -> g g < 1)`
  - `...`

Solutions

- $\boldsymbol{TypeInfer}$(**`let f x = x + 1 in f true`**)$= ?$

$$\overbrace{\texttt{let f } \underbrace{\texttt{x}}_{\tau_x} \texttt{ = } \underbrace{\texttt{x + 1}}_{\tau_r} \texttt{ in f } \underbrace{\texttt{true}}_{\tau_a}}^{\tau_p}$$

Type Equations

$\tau_r = \text{int} \land$
$\tau_x = \text{int} \land$
$\text{int} = \text{int} \land$
$(\tau_x \to \tau_r) = (\tau_a \to \tau_p) \land$
$\tau_a = \text{bool}$

Solutions

- $\boldsymbol{TypeInfer}$(`let f x = x + 1 in f true`)$= ?$

$$\overbrace{\texttt{let f } \underbrace{\texttt{x}}_{\tau_x} \texttt{ = } \underbrace{\texttt{x + 1}}_{\tau_r} \texttt{ in f } \underbrace{\texttt{true}}_{\tau_a}}^{\tau_p}$$

Type Equations

$\tau_r = \mathrm{int}\ \wedge$
$\tau_x = \mathrm{int}\ \wedge$
$\mathrm{int} = \mathrm{int}\ \wedge$
$(\tau_x \rightarrow \tau_r) = (\tau_a \rightarrow \tau_p)\ \wedge$
$\tau_a = \mathrm{bool}$

Substitution

$\tau_r = \mathrm{int}$
$\tau_x = \mathrm{int}$
$\tau_a = \mathrm{int}$
$\tau_p = \mathrm{int}$

Encounter $\mathrm{int} = \mathrm{bool}$

Type error!

Solutions

- $TypeInfer($`fun g -> g g < 1`$) = ?$

$$\overbrace{\texttt{fun}\ \ \underbrace{\texttt{g}}_{\tau_g}\ \ \texttt{->}\ \ \underbrace{\texttt{g}\ \ \underbrace{\texttt{g}}_{\tau_a}\ \ \texttt{<}\ \ \texttt{1}}_{\tau_r}}^{\tau_p}$$

Type Equations

$\tau_p = \tau_g \to \tau_r\ \wedge$
$\tau_r = \text{bool}\ \wedge$
$\tau_g = \tau_a \to \text{int}\ \wedge$
$\tau_g = \tau_a\ \wedge$
$\text{int} = \text{int}$

Solutions

- $\boldsymbol{TypeInfer}($**`fun g -> g g < 1`**$) = ?$

$$\overbrace{\texttt{fun}\ \ \underbrace{\texttt{g}}_{\tau_g}\ \ \texttt{->}\ \ \underbrace{\texttt{g}\ \ \underbrace{\texttt{g}}_{\tau_a}\ \ \texttt{<}\ \ \texttt{1}}_{\tau_r}}^{\tau_p}$$

Type Equations

$\tau_p = \tau_g \to \tau_r \ \wedge$

$\tau_r = \text{bool} \ \wedge$

$\tau_g = \tau_a \to \text{int} \ \wedge$

$\tau_g = \tau_a \ \wedge$

$\text{int} = \text{int}$

Substitution

$\tau_p = (\tau_a \to \text{int}) \to \text{bool}$

$\tau_r = \text{bool}$

$\tau_g = (\tau_a \to \text{int})$

Type error!

Encounter$(\tau_a \to \text{int}) = \tau_a$