

# **Chapter 9.**

# **Type Inference**

**Prof. Jaeseung Choi**

**Dept. of Computer Science and Engineering**

**Sogang University**

# Review: Typing Rules for F-

$$\overline{\Gamma \vdash n : \text{int}}$$

$$\overline{\Gamma \vdash \text{true} : \text{bool}}$$

$$\overline{\Gamma \vdash \text{false} : \text{bool}}$$

$$\overline{\Gamma \vdash x : \Gamma(x)}$$

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

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

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

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

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

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

# Review: Domains for Type

## ■ The set of types ( $T$ ) is defined as follow

- We will use  $t$  to denote it ( $t \in T$ )
- And  $\tau$  below denotes a type variable ( $\tau \in TyVar = String$ )

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

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

## ■ Type environment is a mapping from variable to type

- We will use  $\Gamma$  to denote it ( $\Gamma \in TyEnv = Var \rightarrow T$ )

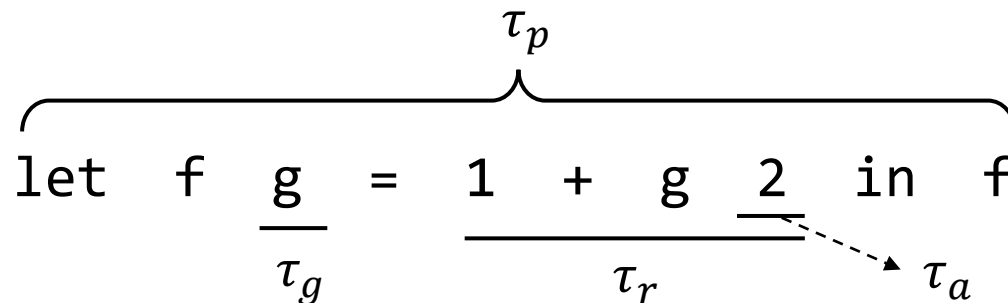
# Type Inference Outline

- **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 ( $\tau_*$ ) and construct type equations
- Then, find the solution that satisfies the equations



### 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  $\Gamma$
  - Definition of ***Gen***( ) must be in accordance with the typing rules
  - Generated type equations are connected with conjunction
    - Have the form of  $(t_1 = t'_1) \wedge (t_2 = t'_2) \wedge \dots \wedge (t_n = t'_n)$

$$\overline{\Gamma \vdash n : \text{int}}$$

$$\mathbf{Gen}(\Gamma, n, t) = (t = \text{int})$$

$$\overline{\Gamma \vdash \text{true} : \text{bool}}$$

$$\mathbf{Gen}(\Gamma, \text{true}, t) = (t = \text{bool})$$

$$\overline{\Gamma \vdash \text{false} : \text{bool}}$$

$$\mathbf{Gen}(\Gamma, \text{false}, t) = (t = \text{bool})$$

$$\overline{\Gamma \vdash x : \Gamma(x)}$$

$$\mathbf{Gen}(\Gamma, x, t) = (t = \Gamma(x))$$

# 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  $\Gamma$
- Definition of ***Gen***( ) must be in accordance with the typing rules
- Generated type equations are connected with conjunction
  - Have the form of  $(t_1 = t'_1) \wedge (t_2 = t'_2) \wedge \cdots \wedge (t_n = t'_n)$

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

$$\mathbf{Gen}(\Gamma, e_1 + e_2, t) = (t = \text{int}) \wedge \mathbf{Gen}(\Gamma, e_1, \text{int}) \wedge \mathbf{Gen}(\Gamma, e_2, \text{int})$$

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

$$\mathbf{Gen}(\Gamma, e_1 < e_2, t) = (t = \text{bool}) \wedge \mathbf{Gen}(\Gamma, e_1, \text{int}) \wedge \mathbf{Gen}(\Gamma, e_2, \text{int})$$

# 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  $\Gamma$
- Definition of ***Gen***( ) must be in accordance with the typing rules
- Generated type equations are connected with conjunction
  - Have the form of  $(t_1 = t'_1) \wedge (t_2 = t'_2) \wedge \cdots \wedge (t_n = t'_n)$

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

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

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

$$\begin{aligned} \text{Gen}(\Gamma, \text{let } x = e_1 \text{ in } e_2, t) = \\ & \text{Gen}(\Gamma, e_1, \tau) \wedge \text{Gen}(\Gamma[x \mapsto \tau], e_2, t) \\ & (\tau \text{ is a newly generated type variable}) \end{aligned}$$



# Generating Type Equations

## ■ Let's define function $\mathbf{Gen}(\Gamma, e, t)$ in *pattern match* style

- Generate the constraint to make  $e$  have type  $t \in T$  under  $\Gamma$
- Definition of  $\mathbf{Gen}()$  must be in accordance with the typing rules
- Generated type equations are connected with conjunction
  - Have the form of  $(t_1 = t'_1) \wedge (t_2 = t'_2) \wedge \dots \wedge (t_n = t'_n)$

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

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

$$\begin{aligned} \mathbf{Gen}(\Gamma, \text{let } f \ x = e_1 \text{ in } e_2, t) = \\ \mathbf{Gen}(\Gamma[x \mapsto \tau_a], e_1, \tau_r) \wedge \\ \mathbf{Gen}(\Gamma[f \mapsto (\tau_a \rightarrow \tau_r)], e_2, t) \end{aligned}$$

( $\tau_a$  and  $\tau_r$  are new type variables)

$$\begin{aligned} \mathbf{Gen}(\Gamma, e_1 \ e_2, t) = \\ \mathbf{Gen}(\Gamma, e_1, \tau_a \rightarrow t) \wedge \mathbf{Gen}(\Gamma, e_2, \tau_a) \end{aligned}$$

( $\tau_a$  is a new type variable)

# 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  $\tau_p$  for the whole program

$$\begin{array}{c}
 \tau_p \\
 \underbrace{\hspace{10em}} \\
 \text{let } f \quad \frac{g}{\tau_g} = 1 + g \quad \frac{2}{\tau_r} \text{ in } f \\
 \hspace{15em} \nearrow \tau_a
 \end{array}$$

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

# Exercise #1

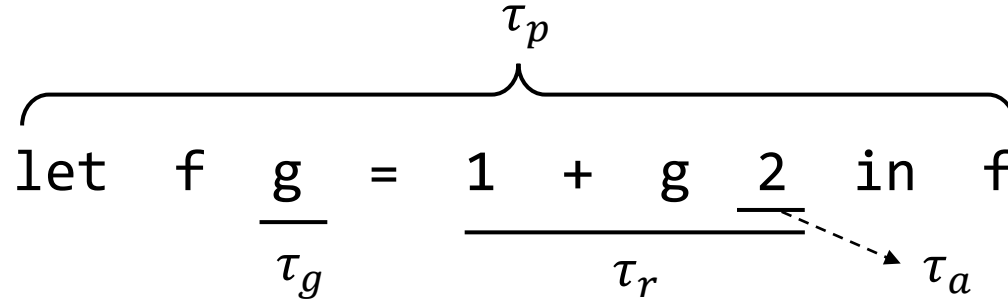
- Let's run ***Gen***( $\Gamma, e, t$ ) and see which type equations are generated from the previous example
  - Start by introducing type variable  $\tau_p$  for the whole program

$$\begin{array}{c}
 \tau_p \\
 \underbrace{\hspace{10em}} \\
 \text{let } f \quad \frac{g}{\tau_g} = 1 + g \quad \frac{2}{\tau_r} \text{ in } f
 \end{array}
 \quad \begin{array}{c}
 \text{---} \\
 \nearrow \tau_a
 \end{array}$$

$$\begin{aligned}
 &Gen(\phi, \text{let } f \ g = 1 + g \ 2 \text{ in } f, \tau_p) = \\
 &Gen(\{g \mapsto \tau_g\}, 1 + g \ 2, \tau_r) \wedge Gen(\{f \mapsto (\tau_g \rightarrow \tau_r)\}, f, \tau_p) =
 \end{aligned}$$

# Exercise #1

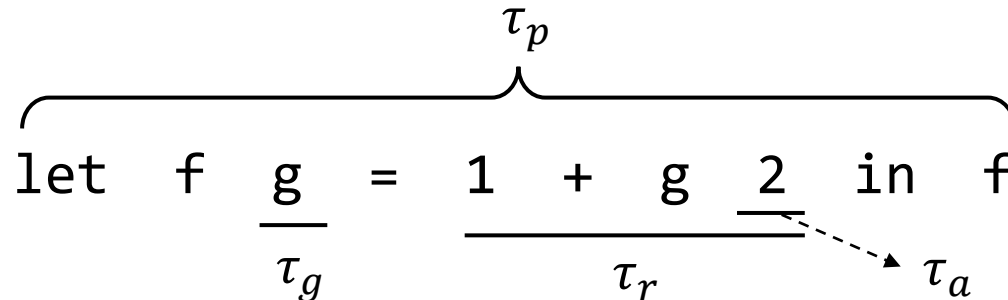
- Let's run *Gen*( $\Gamma, e, t$ ) and see which type equations are generated from the previous example
  - Start by introducing type variable  $\tau_p$  for the whole program



$$\begin{aligned}
 &Gen(\phi, \text{let } f \text{ } \frac{g}{\tau_g} = 1 + g \text{ } \frac{2}{\tau_r} \text{ in } f, \tau_p) = \\
 &Gen(\{g \mapsto \tau_g\}, 1 + g \text{ } 2, \tau_r) \wedge Gen(\{f \mapsto (\tau_g \rightarrow \tau_r)\}, f, \tau_p) = \\
 &(\tau_r = \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, 1, \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, g \text{ } 2, \text{int}) \wedge (\tau_p = \tau_g \rightarrow \tau_r) =
 \end{aligned}$$

# 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  $\tau_p$  for the whole program

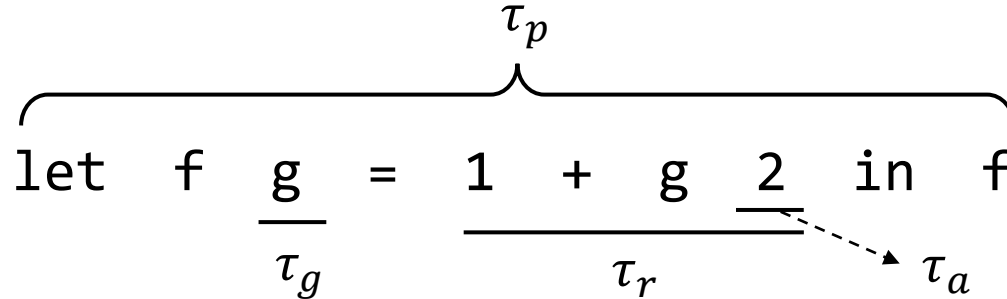


$$\begin{aligned}
 & Gen(\phi, \text{let } f \text{ } g = 1 + g \text{ } 2 \text{ in } f, \tau_p) = \\
 & Gen(\{g \mapsto \tau_g\}, 1 + g \text{ } 2, \tau_r) \wedge Gen(\{f \mapsto (\tau_g \rightarrow \tau_r)\}, f, \tau_p) = \\
 & (\tau_r = \text{int}) \wedge \textcolor{red}{Gen}(\{g \mapsto \tau_g\}, 1, \text{int}) \wedge \textcolor{blue}{Gen}(\{g \mapsto \tau_g\}, g \text{ } 2, \text{int}) \wedge (\tau_p = \tau_g \rightarrow \tau_r) = \\
 & (\tau_r = \text{int}) \wedge (\text{int} = \text{int}) \wedge \textcolor{blue}{Gen}(\{g \mapsto \tau_g\}, g, \tau_a \rightarrow \text{int}) \wedge \textcolor{blue}{Gen}(\{g \mapsto \tau_g\}, 2, \tau_a) \wedge \\
 & (\tau_p = \tau_g \rightarrow \tau_r) =
 \end{aligned}$$

# 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  $\tau_p$  for the whole program

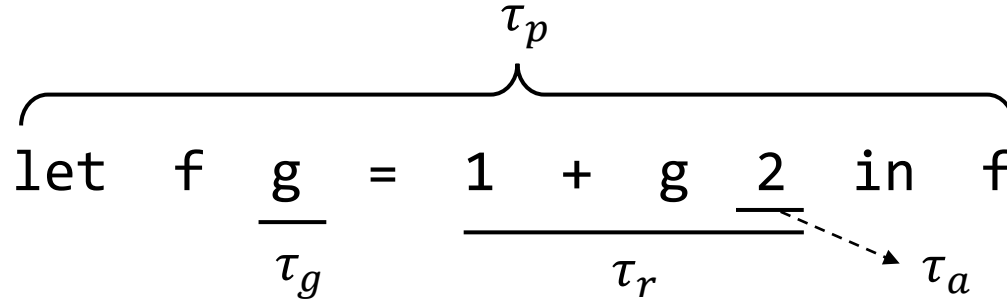


$$\begin{aligned}
 & Gen(\phi, \text{let } f \text{ } g = 1 + g \text{ } 2 \text{ in } f, \tau_p) = \\
 & Gen(\{g \mapsto \tau_g\}, 1 + g \text{ } 2, \tau_r) \wedge Gen(\{f \mapsto (\tau_g \rightarrow \tau_r)\}, f, \tau_p) = \\
 & (\tau_r = \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, 1, \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, g \text{ } 2, \text{int}) \wedge (\tau_p = \tau_g \rightarrow \tau_r) = \\
 & (\tau_r = \text{int}) \wedge (\text{int} = \text{int}) \wedge \textcolor{red}{Gen}(\{g \mapsto \tau_g\}, g, \tau_a \rightarrow \text{int}) \wedge \textcolor{blue}{Gen}(\{g \mapsto \tau_g\}, 2, \tau_a) \wedge \\
 & (\tau_p = \tau_g \rightarrow \tau_r) = \\
 & (\tau_r = \text{int}) \wedge (\text{int} = \text{int}) \wedge \textcolor{red}{(\tau_g = \tau_a \rightarrow \text{int})} \wedge \textcolor{blue}{(\tau_a = \text{int})} \wedge (\tau_p = \tau_g \rightarrow \tau_r)
 \end{aligned}$$

# 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  $\tau_p$  for the whole program



$$\begin{aligned}
 & Gen(\phi, \text{let } f \text{ } \frac{g}{\tau_g} = 1 + g \text{ } \frac{2}{\tau_r} \text{ in } f, \tau_p) = \\
 & Gen(\{g \mapsto \tau_g\}, 1 + g \text{ } 2, \tau_r) \wedge Gen(\{f \mapsto (\tau_g \rightarrow \tau_r)\}, f, \tau_p) = \\
 & (\tau_r = \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, 1, \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, g \text{ } 2, \text{int}) \wedge (\tau_p = \tau_g \rightarrow \tau_r) = \\
 & (\tau_r = \text{int}) \wedge (\text{int} = \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, g, \tau_a \rightarrow \text{int}) \wedge Gen(\{g \mapsto \tau_g\}, 2, \tau_a) \wedge \\
 & (\tau_p = \tau_g \rightarrow \tau_r) = \\
 & (\tau_r = \text{int}) \wedge (\text{int} = \text{int}) \wedge (\tau_g = \tau_a \rightarrow \text{int}) \wedge (\tau_a = \text{int}) \wedge (\tau_p = \tau_g \rightarrow \tau_r)
 \end{aligned}$$

# 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  $\tau$  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

$$\text{let } f \text{ } \underbrace{\frac{g}{\tau_g} = \frac{1 + g \text{ } \frac{2}{\tau_r}}{\tau_r}}_{\tau_p} \text{ in } f$$

$\tau_a$

## Type Equations

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

## Substitution

*(starts from empty map)*

# Running Example

- We will construct a **substitution (mapping from  $\tau$  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

$$\text{let } f \quad \underbrace{\frac{g}{\tau_g} = \frac{1 + g \quad 2}{\tau_r}}_{\tau_p} \text{ in } f$$

$\tau_a$

## Type Equations

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

## Substitution

$\tau_r \mapsto \text{int}$

# Running Example

- We will construct a **substitution** (mapping from  $\tau$  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

$$\text{let } f \text{ } \frac{g}{\tau_g} = \frac{1 + g \text{ } \frac{2}{\tau_a}}{\tau_r} \text{ in } f$$

$\tau_p$  (bracketed over the entire expression)  
 $\tau_r$  (under the right-hand side)  
 $\tau_a$  (pointed to by a dashed arrow from the '2')

## 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

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

int = int is just ignored

# Running Example

- We will construct a **substitution** (mapping from  $\tau$  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

$$\text{let } f \text{ } \frac{g}{\tau_g} = \frac{1 + g \text{ } \frac{2}{\tau_a}}{\tau_r} \text{ in } f$$

$\tau_p$  (bracketed over the entire expression)  
 $\tau_r$  (under the right-hand side)  
 $\tau_g$  (under  $g$ )  
 $\tau_a$  (pointed to by a dashed arrow from the  $2$ )

## 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

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

# Running Example

- We will construct a **substitution** (mapping from  $\tau$  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

$$\text{let } f \text{ } \underbrace{\frac{g}{\tau_g} = \frac{1 + g \text{ } \frac{2}{\tau_r}}{\tau_r}}_{\tau_p} \text{ in } f$$

$\tau_a$

## Type Equations

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

## Substitution

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

$\tau_a$  replaced  
with **int**

# Running Example

- We will construct a **substitution** (mapping from  $\tau$  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

$$\overbrace{\text{let } f \quad \frac{g}{\tau_g} = 1 + g \quad \frac{2}{\tau_r} \text{ in } f}^{\tau_p} \quad \tau_a$$

(A dashed arrow points from the  $2$  in the expression to  $\tau_a$ )

## Type Equations

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

## Substitution

$\tau_r \mapsto \text{int}$   
 $\tau_g \mapsto \text{int} \rightarrow \text{int}$   
 $\tau_a \mapsto \text{int}$   
 $\tau_p \mapsto (\text{int} \rightarrow \text{int}) \rightarrow \text{int}$

$\tau_g$  replaced  
with **int  $\rightarrow$  int**

# Running Example

- We will construct a **substitution** (mapping from  $\tau$  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

$$\text{let } f \text{ } \frac{g}{\tau_g} = \frac{1 + g \text{ } \frac{2}{\tau_a}}{\tau_r} \text{ in } f$$

$\tau_p$  (bracketed over the entire expression)  
 $\tau_r$  (under the right-hand side)  
 $\tau_a$  (pointed to by a dashed arrow from the  $2$ )

## Type Equations

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

## Substitution

$\tau_r \mapsto \text{int}$   
 $\tau_g \mapsto \text{int} \rightarrow \text{int}$   
 $\tau_a \mapsto \text{int}$   
 $\tau_p \mapsto (\text{int} \rightarrow \text{int}) \rightarrow \text{int}$

$\tau_r$  replaced with **int**

# Running Example

- We will construct a **substitution** (mapping from  $\tau$  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

$$\text{let } f \text{ } \frac{g}{\tau_g} = \frac{1 + g \text{ } \frac{2}{\tau_a}}{\tau_r} \text{ in } f$$

$\tau_p$  (bracketed over the entire expression)  
 $\tau_r$  (under the right-hand side)  
 $\tau_a$  (pointed to by a dashed arrow from the  $2$ )

## 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

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

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

# Application of Substitution

## ■ Let's consider some examples

- Ex)  $App(\{\tau_1 \mapsto \text{bool}\}, \tau_2)) = ?$
- Ex)  $App(\{\tau_1 \mapsto \text{int}\}, \text{bool})) = ?$
- Ex)  $App(\{\tau_1 \mapsto \text{bool}, \tau_2 \mapsto \text{int}\}, \tau_2)) = ?$
- Ex)  $App(\{\tau_1 \mapsto \text{int}, \tau_2 \mapsto \text{bool}\}, \text{bool} \rightarrow (\tau_1 \rightarrow \tau_3)) = ?$

# Application of Substitution

## ■ Let's consider some examples

- Ex)  $App(\{\tau_1 \mapsto \text{bool}\}, \tau_2) = \tau_2$
- Ex)  $App(\{\tau_1 \mapsto \text{int}\}, \text{bool}) = \text{bool}$
- Ex)  $App(\{\tau_1 \mapsto \text{bool}, \tau_2 \mapsto \text{int}\}, \tau_2) = \text{int}$
- Ex)  $App(\{\tau_1 \mapsto \text{int}, \tau_2 \mapsto \text{bool}\}, \text{bool} \rightarrow (\tau_1 \rightarrow \tau_3)) = \text{bool} \rightarrow (\text{int} \rightarrow \tau_3)$

# 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: *Unify()* and *Extend()***
- ***Unify*( $t_1 = t_2, s$ ) must merge a new equation  $t_1 = t_2$  into the current substitution  $s$  and return a new substitution**
  - It first applies the current substitution  $s$  to  $t_1$  and  $t_2$ , and then calls *Extend()* with them
  - $Unify(t_1 = t_2, s) = Extend(App(s, t_1) = App(s, t_2), s)$
- ***Extend*( $t_1 = t_2, s$ ) is the actual function that moves the equation to the substitution**

# Solving Equations

## ■ Let's define $Extend(t_1 = t_2, s)$ in *pattern match style*

- $Extend(\text{int} = \text{int}, s) = s$

- $Extend(\text{bool} = \text{bool}, s) = s$

- $Extend((t_x \rightarrow t_y) = (t'_x \rightarrow t'_y), s) =$   
 $Unify(t_y = t'_y, Extend(t_x = t'_x, s))$

If both sides are functions

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

If the left or right side is type variable

- $Extend(*, s) = \text{error}$

Other cases like  $\text{int} = \text{bool}$  ,  $\text{int} = (t_x \rightarrow t_y)$ , ...

# Solving Equations

## ■ Let's examine some tricky cases in $Extend(t_1 = t_2, s)$

- $Extend(\tau = t, s) =$

- If  $t$  is  $\tau : s$
  - Else if  $\tau$  occurs in  $t$  : error
  - Otherwise :  $s'[\tau \mapsto t]$ , where  $s'$  is  $s$  with  $\tau$  replaced into  $t$

- Ex)  $Extend(\tau_1 = \tau_1, \{\tau_f \mapsto (\tau_1 \rightarrow \text{int})\}) = \{\tau_f \mapsto (\tau_1 \rightarrow \text{int})\}$   
(nothing to update)

- Ex)  $Extend(\tau_1 = (\text{int} \rightarrow \tau_1), \{\tau_f \mapsto (\tau_1 \rightarrow \text{int})\}) = \text{type error}$   
(Such form is not supported in current type system)

- Ex)  $Extend(\tau_1 = \text{bool}, \{\tau_f \mapsto (\tau_1 \rightarrow \text{int})\}) =$   
 $\{\tau_f \mapsto (\text{bool} \rightarrow \text{int}), \tau_1 \mapsto \text{bool}\}$

# Solving Equations

## ■ Let's examine some tricky cases in $Extend(t_1 = t_2, s)$

- $Extend((t_x \rightarrow t_y) = (t'_x \rightarrow t'_y), s) =$   
 $Unify(t_y = t'_y, Extend(t_x = t'_x, s))$
- Note that it's not  $Extend(t_y = t'_y, Extend(t_x = t'_x, s))$
- Recall the definition of  $Unify()$  in the previous page
  - It will apply the substitution obtained from  $Extend(t_x = t'_x, s)$
- Ex)  $Extend((\tau_1 \rightarrow \tau_2) = (\text{int} \rightarrow \tau_1), \{ \}) = ?$

# Solving Equations

## ■ Let's examine some tricky cases in $Extend(t_1 = t_2, s)$

- $Extend((t_x \rightarrow t_y) = (t'_x \rightarrow t'_y), s) =$   
 $Unify(t_y = t'_y, Extend(t_x = t'_x, s))$
- Note that it's not  $Extend(t_y = t'_y, Extend(t_x = t'_x, s))$
- Recall the definition of  $Unify()$  in the previous page
  - It will apply the substitution obtained from  $Extend(t_x = t'_x, s)$
- Ex)  $Extend((\tau_1 \rightarrow \tau_2) = (\text{int} \rightarrow \tau_1), \{ \}) =$   
 $Unify(\tau_2 = \tau_1, Extend(\tau_1 = \text{int}, \{ \})) =$   
 $Unify(\tau_2 = \tau_1, \{ \tau_1 \mapsto \text{int} \}) =$   
 $Extend(\tau_2 = \text{int}, \{ \tau_1 \mapsto \text{int} \}) =$   
 $\{ \tau_1 \mapsto \text{int}, \tau_2 \mapsto \text{int} \}$



# 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 *TypeInfer*(let f g = 1 + g 2 in f)

```
TypeInfer(E) =  
  let  $(t_1 = t'_1) \wedge (t_2 = t'_2) \dots \wedge (t_n = t'_n) = \mathbf{Gen}(\phi, E, \tau_p)$   
  let  $s_1 = \mathbf{Unify}(t_1 = t'_1, \phi)$   
  let  $s_2 = \mathbf{Unify}(t_2 = t'_2, s_1)$   
  ...  
  let  $s_n = \mathbf{Unify}(t_n = t'_n, s_{n-1})$   
   $s_n[\tau_p]$  // This is the final type of program E
```

# Exercise #2

■ *TypeInfer*(let f g = g 1 in f) = ?

$\tau_p$

let f g = g 1 in f

# Exercise #2

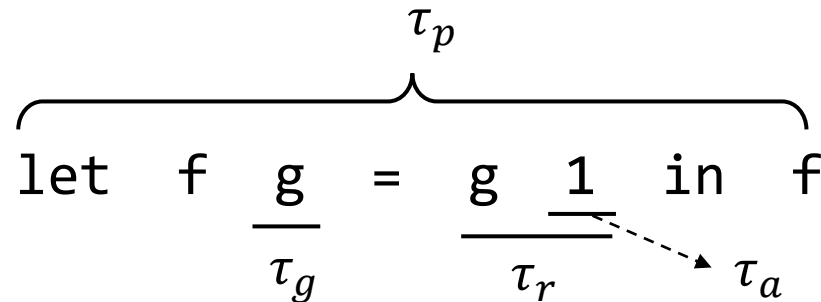
■ *TypeInfer*(let f g = g 1 in f) = ?

$$\overbrace{\text{let } f \frac{g}{\tau_g} = \frac{g \frac{1}{\tau_r}}{\tau_r} \text{ in } f}^{\tau_p} \quad \tau_a$$

$$\begin{aligned} \text{Gen}(\phi, \text{let } f \ g = g \ 1 \text{ in } f, \tau_p) &= \\ \text{Gen}(\{g \mapsto \tau_g\}, g \ 1, \tau_r) \wedge \text{Gen}(\{f \mapsto (\tau_g \rightarrow \tau_r)\}, f, \tau_p) &= \\ \text{Gen}(\{g \mapsto \tau_g\}, g, \tau_a \rightarrow \tau_r) \wedge \text{Gen}(\{g \mapsto \tau_g\}, 1, \tau_a) \wedge (\tau_p = \tau_g \rightarrow \tau_r) &= \\ (\tau_g = \tau_a \rightarrow \tau_r) \wedge (\tau_a = \text{int}) \wedge (\tau_p = \tau_g \rightarrow \tau_r) \end{aligned}$$

# Exercise #2

■  $TypeInfer(\text{let } f \text{ } g = g \text{ } 1 \text{ in } f) = ?$



## Type Equations

$$\tau_g = (\tau_a \rightarrow \tau_r) \wedge$$

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

$$\tau_p = (\tau_g \rightarrow \tau_r)$$

## Substitution

$$\tau_g = \text{int} \rightarrow \tau_r$$

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

$$\tau_p = (\text{int} \rightarrow \tau_r) \rightarrow \tau_r$$

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 *Gen()* for the following cases?

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

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

- Run type inference for various examples

- *TypeInfer*(let f x = x + 1 in f true)
- *TypeInfer*(fun g -> g g < 1)
- ...

# Solutions

■ *TypeInfer*(let f x = x + 1 in f true)=?

$$\overbrace{\text{let } f \frac{x}{\tau_x} = \frac{x + 1}{\tau_r} \text{ in } f \frac{\text{true}}{\tau_a}}^{\tau_p}$$

## Type Equations

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

$$\tau_x = \text{int} \wedge$$

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

$$(\tau_x \rightarrow \tau_r) = (\tau_a \rightarrow \tau_p) \wedge$$

$$\tau_a = \text{bool}$$

# Solutions

■ *TypeInfer*(let f x = x + 1 in f true)=?

$$\frac{\text{let } f \quad \frac{x}{\tau_x} = \frac{x + 1}{\tau_r} \quad \text{in } f \quad \frac{\text{true}}{\tau_a}}{\tau_p}$$

## Type Equations

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

## Substitution

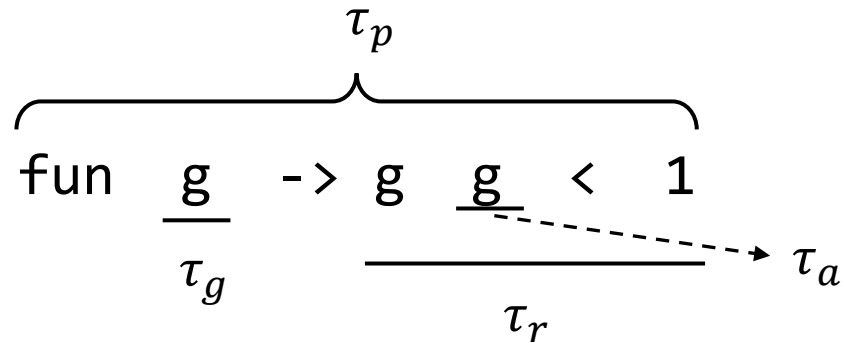
$$\begin{aligned}\tau_r &= \text{int} \\ \tau_x &= \text{int} \\ \tau_a &= \text{int} \\ \tau_p &= \text{int}\end{aligned}$$

Encounter int = bool

Type error!

# Solutions

■ *TypeInfer*(fun g -> g g < 1)=?



## Type Equations

$$\tau_p = \tau_g \rightarrow \tau_r \wedge$$

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

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

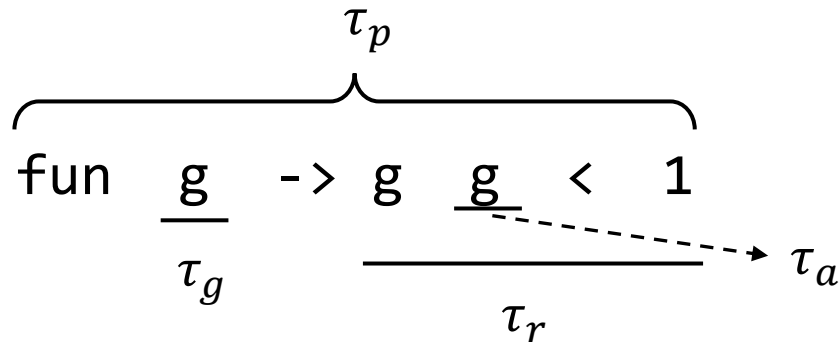
$$\tau_g = \tau_a \wedge$$

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



# Solutions

■ *TypeInfer*(fun g -> g g < 1) = ?



## Type Equations

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

## Substitution

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

Type error!

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