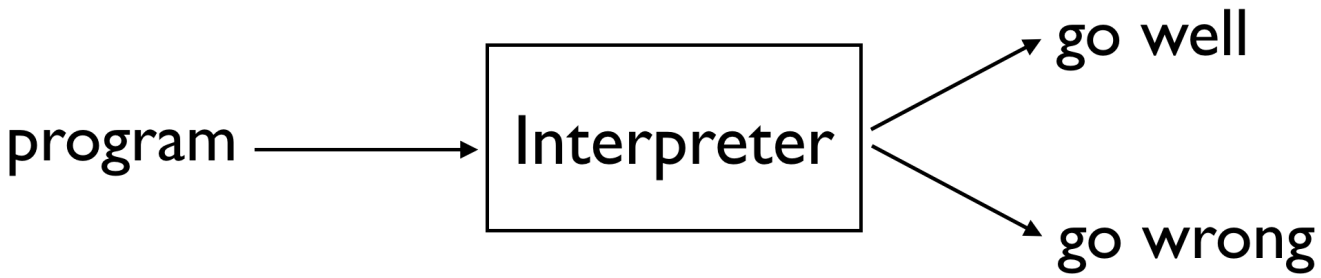
COSE212: Programming Languages

Lecture 9 — Type System (1)

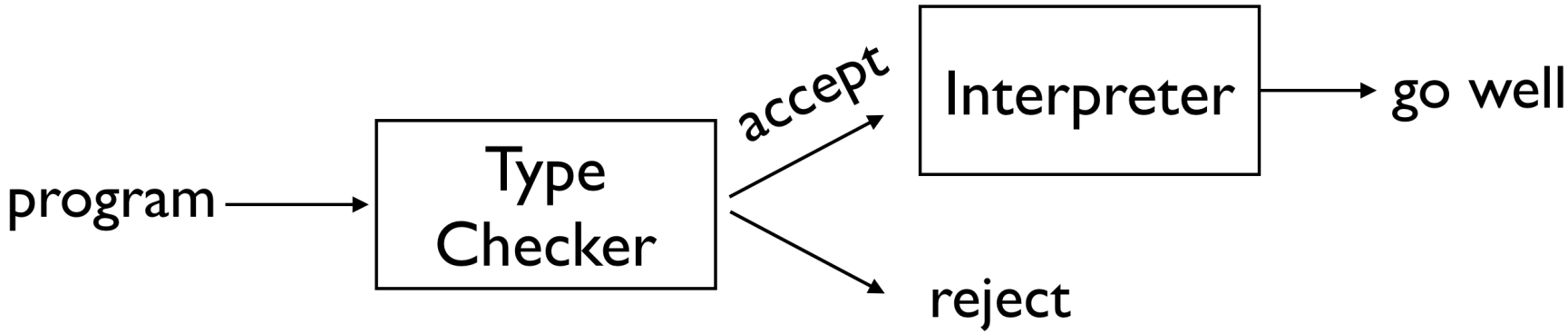
Hakjoo Oh
2015 Fall

Program Execution without Type Checker



- `if true then 88 else 99`
- `if 3 then 88 else 99`
- `(proc (x) (x 3)) (proc (x) x)`
- `(proc (x) (x 3)) 4`
- `let x = 4 in (x 3)`
- `(proc (x) (3 x))` e
- `let x = zero? 0 in (3-x)`

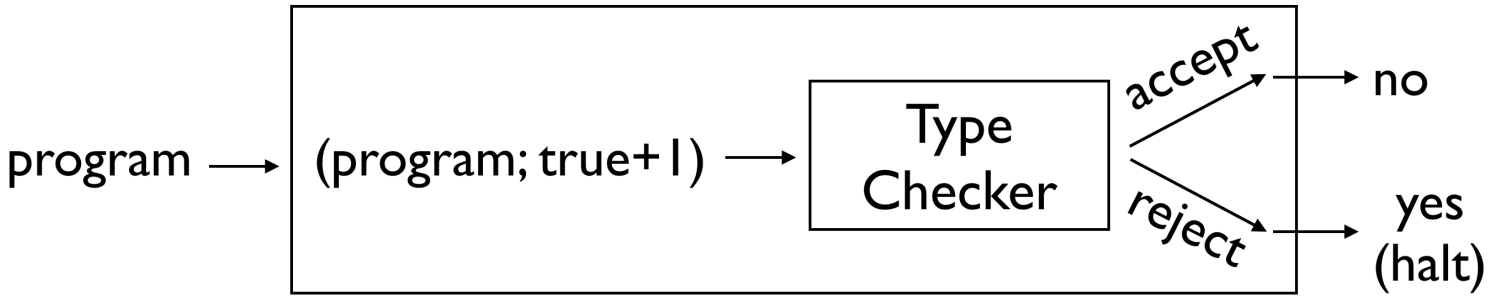
Program Execution with Type Checker



- `if true then 88 else 99`
- `if 3 then 88 else 99`
- `(proc (x) (x 3)) (proc (x) x)`
- `(proc (x) (x 3)) 4`
- `let x = 4 in (x 3)`
- `(proc (x) (3 x))` e
- `let x = zero? 0 in (3-x)`

Type Checking is Undecidable

If possible, we can solve the halting problem using the type checker:



Only Approximate Type Checking is Possible

All static type systems in use give approximated answers, which can be divided into *sound* and *unsound* type checkers.

- Sound type checker guarantees that type checked programs do not go wrong. Used in ML, Haskell, etc.
- Unsound type checker does not give such guarantee. Used in C, C++, etc.

Plan: Sound Type Checker for the PROC Language

$$
\begin{array}{rcl}
E & \rightarrow & n \\
 & | & x \\
 & | & E + E \\
 & | & E - E \\
 & | & \texttt{zero?}\ E \\
 & | & \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E \\
 & | & \texttt{let}\ x = E\ \texttt{in}\ E \\
 & | & \texttt{proc}\ x\ E \\
 & | & E\ E
\end{array}
$$

Plan: Sound Type Checker for the PROC Language

$$\frac{}{\rho \vdash n \Rightarrow n} \qquad \frac{}{\rho \vdash x \Rightarrow \rho(x)} \qquad \frac{\rho \vdash E_1 \Rightarrow n_1 \qquad \rho \vdash E_2 \Rightarrow n_2}{\rho \vdash E_1 + E_2 \Rightarrow n_1 + n_2}$$

$$\frac{\rho \vdash E \Rightarrow 0}{\rho \vdash \texttt{zero?}\ E \Rightarrow \mathit{true}} \qquad \frac{\rho \vdash E \Rightarrow n}{\rho \vdash \texttt{zero?}\ E \Rightarrow \mathit{false}}\ n \neq 0$$

$$\frac{\rho \vdash E_1 \Rightarrow \mathit{true} \qquad \rho \vdash E_2 \Rightarrow v}{\rho \vdash \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 \Rightarrow v} \qquad \frac{\rho \vdash E_1 \Rightarrow \mathit{false} \qquad \rho \vdash E_3 \Rightarrow v}{\rho \vdash \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 \Rightarrow v}$$

$$\frac{\rho \vdash E_1 \Rightarrow v_1 \qquad [x \mapsto v_1]\rho \vdash E_2 \Rightarrow v}{\rho \vdash \texttt{let}\ x = E_1\ \texttt{in}\ E_2 \Rightarrow v}$$

$$\frac{}{\rho \vdash \texttt{proc}\ x\ E \Rightarrow (x, E, \rho)}$$

$$\frac{\rho \vdash E_1 \vdash (x, E, \rho') \qquad \rho \vdash E_2 \Rightarrow v \qquad [x \mapsto v]\rho' \vdash E \Rightarrow v'}{\rho \vdash E_1\ E_2 \Rightarrow v'}$$

Types

Types are defined inductively:

$$\begin{array}{rcl} \boldsymbol{T} & \rightarrow & \text{int} \\ & | & \text{bool} \\ & | & \boldsymbol{T} \rightarrow \boldsymbol{T} \end{array}$$

Examples:

- int
- bool
- int $\rightarrow$ int
- bool $\rightarrow$ int
- int $\rightarrow$ (int $\rightarrow$ bool)
- (int $\rightarrow$ int) $\rightarrow$ (bool $\rightarrow$ bool)
- (int $\rightarrow$ int) $\rightarrow$ (bool $\rightarrow$ (bool $\rightarrow$ int))

Types of Expressions

We need *type environment*:

$$\Gamma : Var \rightarrow T$$

Notation:

$\Gamma \vdash e : t \Leftrightarrow$ Under type environment Γ, expression e has type t.

Examples

- $[] \vdash \mathbf{3} : \text{int}$
- $[x \mapsto \text{int}] \vdash x : \text{int}$
- $[] \vdash \mathbf{4} - \mathbf{3} :$
- $[x \mapsto \text{int}] \vdash x - \mathbf{3} :$
- $[] \vdash \texttt{zero?}\ \mathbf{11} :$
- $[] \vdash \texttt{proc}\ (x)\ (x - \mathbf{11}) :$
- $[] \vdash \texttt{proc}\ (x)\ (\texttt{let}\ y = x - \mathbf{11}\ \texttt{in}\ (x - y)) :$
- $[] \vdash \texttt{proc}\ (x)\ (\texttt{if}\ x\ \texttt{then}\ \mathbf{11}\ \texttt{else}\ \mathbf{22}) :$
- $[] \vdash \texttt{proc}\ (x)\ (\texttt{proc}\ (y)\ \texttt{if}\ y\ \texttt{then}\ x\ \texttt{else}\ \mathbf{11}) :$
- $\texttt{proc}\ (f)\ (\texttt{if}\ (f\ \mathbf{3})\ \texttt{then}\ \mathbf{11}\ \texttt{else}\ \mathbf{22}) :$
- $[] \vdash (\texttt{proc}\ (x)\ x)\ \mathbf{1} :$
- $[f \mapsto \text{int} \to \text{int}] \vdash (f\ (f\ \mathbf{1})) :$

Typing Rules

Inductive rules for assigning types to expressions:

$$\frac{}{\Gamma \vdash n : \text{int}} \qquad \frac{}{\Gamma \vdash x : \Gamma(x)}$$

$$\frac{\Gamma \vdash E_1 : \text{int} \qquad \Gamma \vdash E_2 : \text{int}}{\Gamma \vdash E_1 + E_2 : \text{int}} \qquad \frac{\Gamma \vdash E_1 : \text{int} \qquad \Gamma \vdash E_2 : \text{int}}{\Gamma \vdash E_1 - E_2 : \text{int}}$$

$$\frac{\Gamma \vdash E : \text{int}}{\Gamma \vdash \texttt{zero?}\ E : \text{bool}} \qquad \frac{\Gamma \vdash E_1 : \text{bool} \qquad \Gamma \vdash E_2 : t \qquad \Gamma \vdash E_3 : t}{\texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 : t}$$

$$\frac{\Gamma \vdash E_1 : t_1 \qquad [x \mapsto t_1]\Gamma \vdash E_2 : t_2}{\Gamma \vdash \texttt{let}\ x = E_1\ \texttt{in}\ E_2 : t_2} \qquad \frac{\Gamma \vdash E_1 : t_1 \to t_2 \qquad \Gamma \vdash E_2 : t_1}{\Gamma \vdash E_1\ E_2 : t_2}$$

$$\frac{[x \mapsto t_1]\Gamma \vdash E : t_2}{\Gamma \vdash \texttt{proc}\ x\ E : t_1 \to t_2}$$

We say that a closed expression E has type t iff we can derive $[] \vdash E : t$.

Examples

- `zero?` $(1 + 2)$
- `proc` (x) $(x - 11)$
- `proc` (x) (`if` x `then` 11 `else` 22)
- (`proc` (x) x) 1
- `proc` (x) (`proc` (y) `if` y `then` x `else` 11)

Expressions May Have Multiple Types

- $\texttt{proc}\ x\ x$:

$$\frac{[x \mapsto \text{int}] \vdash x : \text{int}}{[] \vdash \texttt{proc}\ x\ x : \text{int} \to \text{int}}$$

$$\frac{[x \mapsto \text{bool}] \vdash x : \text{bool}}{[] \vdash \texttt{proc}\ x\ x : \text{bool} \to \text{bool}}$$

$$\frac{[x \mapsto (\text{int} \to \text{int})] \vdash x : \text{int} \to \text{int}}{[] \vdash \texttt{proc}\ x\ x : (\text{int} \to \text{int}) \to (\text{int} \to \text{int})}$$

- $\texttt{proc}\ (f)\ (f\ \mathbf{3})$ has type $(\text{int} \to t) \to t$ for any t.
- $\texttt{proc}\ (f)\ \texttt{proc}\ (x)\ (f\ (f\ x))$ has type $((t \to t) \to (t \to t))$ for any t.

Soundness of Typing Rules

Theorem (Soundness)

If a program E has a type error, we cannot find t such that $[] \vdash E : t$. In other words, If E is typed, i.e., $[] \vdash E : t$ for some t, then E is guaranteed to run well.

Examples:

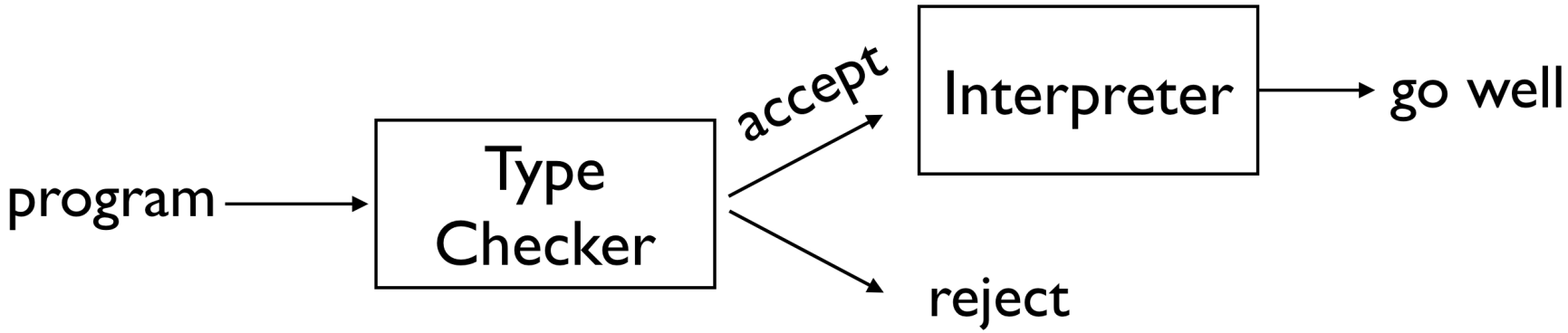
- `if 3 then 88 else 99`
- `(proc (x) (x 3)) 4`
- `let x = 4 in (x 3)`
- `(proc (x) (3 x))` e
- `let x = zero? 0 in (3-x)`
- `(proc` (x) $(x+1))$ `((proc` y $y)$ `(proc` z $z))$

Incompleteness

Even though some programs do not have type errors, they do not have types:

- `if zero? 1 then 11 else (zero? 22))`
- `(proc (f) (f f)) (proc x x)`

Type Checker



- The type checker accepts a program E only if $[] \vdash E : t$ for some t.
- Otherwise, E is rejected.

Challenge

Given a program $\boldsymbol{E}$, how to check $[] \vdash \boldsymbol{E} : \boldsymbol{t}$? Nontrivial, because of the following type rule:

$$\frac{[\boldsymbol{x} \mapsto \boldsymbol{t_1}]\boldsymbol{\Gamma} \vdash \boldsymbol{E} : \boldsymbol{t_2}}{\boldsymbol{\Gamma} \vdash \texttt{proc}\ \boldsymbol{x}\ \boldsymbol{E} : \boldsymbol{t_1} \rightarrow \boldsymbol{t_2}}$$

Two approaches:

- *Type Annotation*: Programmers are required to supply the type of the function argument. Used in C, C++, Java, etc.
- *Type Inference*: Type checker attempts to automatically infer types. Only possible if the language is carefully designed. Used in ML, Haskell, etc.