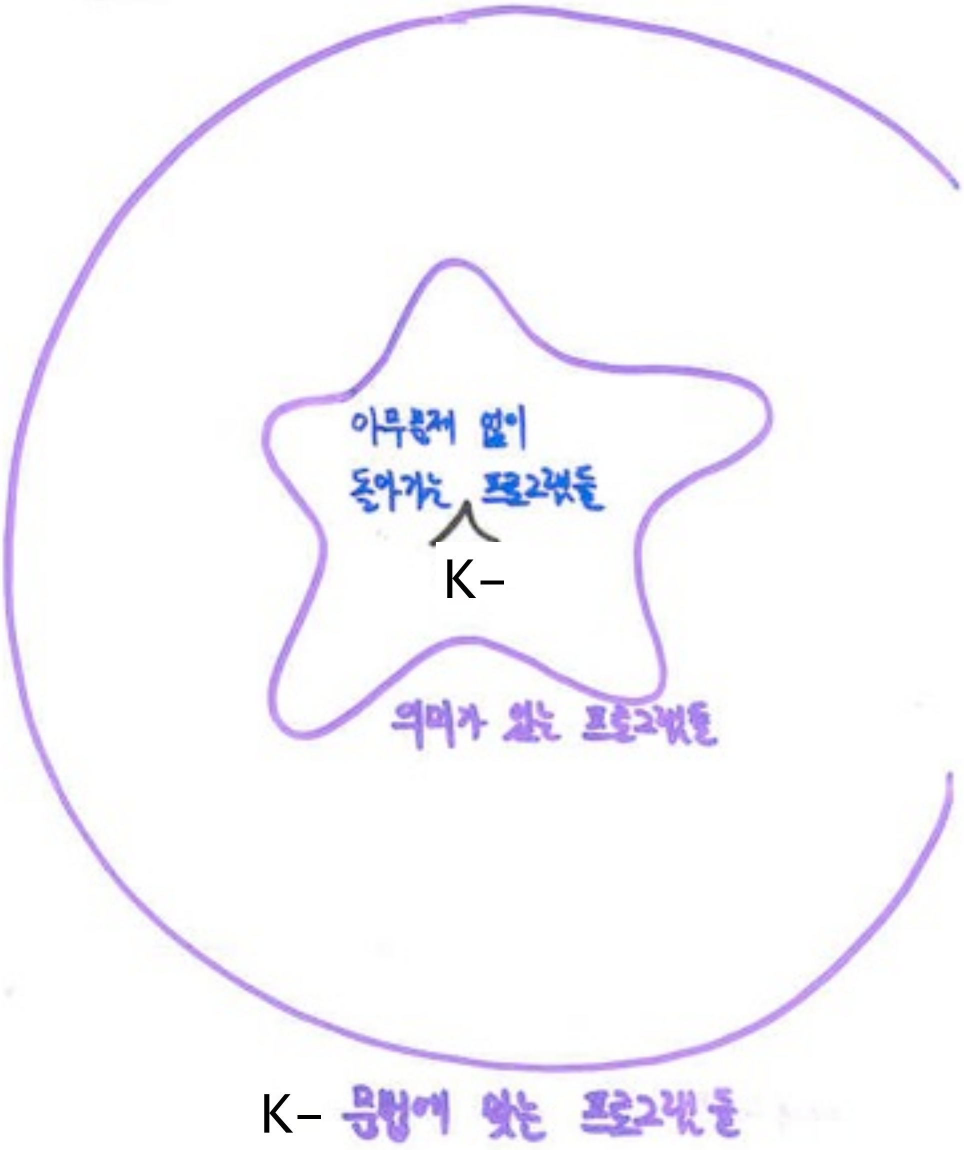
의미 없는 프로그램들

- let x := 1
  in call x (2)
- let x := 1
  in let y := true
     in x+y
- let procedure f(x) = ⋯
  in f + 3
- while 1 do E
- if {name := 1} then 2
- for x := 1 to false do E
- not (E + E)
- if (read x) then 1
- let x := {id := 83031147, age := 19}
  in x.id.age
- let procedure f(x) = ⋯ in call f (f)

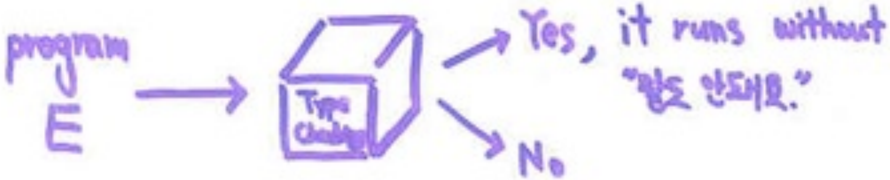
의미있는 프로그램만을 돌리고 싶다!



Type Checking

* 주어진 프로그램이 의미있다고,
문제없이 잘 돌아가는 프로그램이라고,
결정하는 한 방법.

특히, 그 방법이 안전하다면야...



* static type checking : before execution
프로그램을 돌리기 전에 미리.

* dynamic type checking : during execution
프로그램을 돌리는 중에.

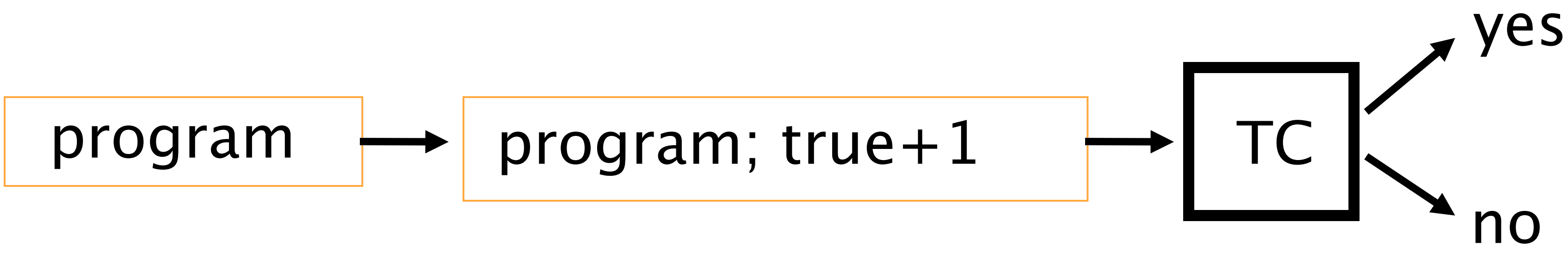
Sound and complete type-checking
is usually impossible.

Only sound type-checking is an alternative.

And we try to make it as complete as possible.

Why is sound & complete type-checking impossible
in general?

If such type-checking procedure TC exists, then
we can solve the Halting problem as follows:



program

program; true+1

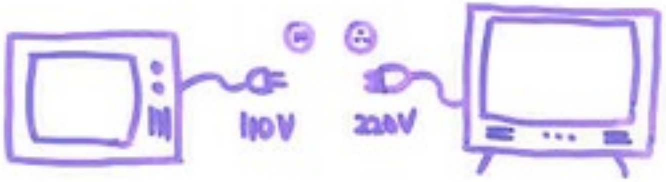
TC

yes

no

Types

품새, 꼴, 격



E : int

E : bool

E : {x: int, y: bool}

f : int → {age: int, id: int}

E : unit

Type Hierarchies

Logics	Computation
syntax definition	parser
semantic definition	interpreter
type checking rules	type checking procedure

다른 이 두개의 세계를 자유자제로 운용할 수 있는 실력을 익히도록 하세요.

$$\Gamma, \varphi \vdash \varphi$$

$$\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi} \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}$$

$$\frac{\Gamma, \varphi \vdash \rho \quad \Gamma, \psi \vdash \rho \quad \Gamma \vdash \varphi \vee \psi}{\Gamma \vdash \rho}$$

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi} \qquad \frac{\Gamma \vdash \varphi \to \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$$

$$\dfrac{\dfrac{\varphi, \psi \vdash \varphi}{\varphi \vdash \psi \to \varphi}}{\vdash \varphi \to (\psi \to \varphi)}$$

$$E \;\rightarrow\; n \mid E + E \mid E \times E$$

$$\boxed{E : \mathtt{even}} \quad \boxed{E : \mathtt{odd}}$$

$$\frac{n\ mod\ 2 = 0}{n : \mathtt{even}} \qquad\qquad \frac{n\ mod\ 2 = 1}{n : \mathtt{odd}}$$

$$\frac{E_1 : \mathtt{even} \qquad E_2 : \mathtt{even}}{E_1 + E_2 : \mathtt{even}}$$

$$\frac{E_1 : \mathtt{odd} \qquad E_2 : \mathtt{odd}}{E_1 + E_2 : \mathtt{even}}$$

$$\begin{array}{lcl} E & \rightarrow & n \mid E + E \mid E \times E \\ & \mid & x \mid \texttt{let } x \texttt{ = } E \texttt{ in } E \end{array}$$

$x + 1 : \texttt{even?}$ $\quad$ $x + 1 : \texttt{odd?}$

All depends on whether x is odd or even

All depends on assumption about x

$$\boxed{\Gamma \vdash E : t}$$

$$\frac{n\ mod\ 2 = 0}{\Gamma \vdash n : \texttt{even}}$$

$$\frac{\Gamma \vdash E_1 : \texttt{even} \qquad \Gamma \vdash E_2 : \texttt{even}}{\Gamma \vdash E_1 + E_2 : \texttt{even}}$$

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

$$\frac{\Gamma \vdash E_1 : t_1 \qquad \Gamma[t/x] \vdash E_2 : t}{\Gamma \vdash \texttt{let}\ x\texttt{=}\ E_1\ \texttt{in}\ E_2 : t}$$

$$\dfrac{\dfrac{\dfrac{x:\mathsf{o}\vdash x:\mathsf{o}\qquad x:\mathsf{o}\vdash 2:\mathsf{e}}{x:\mathsf{o}\vdash x+2:\mathsf{o}}}{x:\mathsf{e}\vdash \mathtt{let}\ x = 3\ \mathtt{in}\ x+2:\mathsf{o}}\qquad \dfrac{x:\mathsf{e}\vdash x:\mathsf{e}}{}}{x:\mathsf{e}\vdash (\mathtt{let}\ x = 3\ \mathtt{in}\ x+2)+x:\mathsf{o}}$$

$$\boxed{\Gamma \vdash E : \tau}$$

K- 의 Types

$$\left.\begin{array}{rcl} \tau & \rightarrow & \text{int} \\ & | & \text{bool} \\ & | & \text{unit} \\ & | & \{x_1 \mapsto \tau, \cdots, x_n \mapsto \tau\} \\ & | & \tau \rightarrow \tau \\ & | & \tau \text{ var} \end{array}\right\} \text{monomorphic types}$$

* 다음 프로그램식들의 type을 찾아보자 *

```
• let x := 1          • let procedure f(x) = x := 1 ; x
  in x                  in let y := 0
                           in call f(y) + y

• let x := {name := 1, age := 2}
  in x.name := x.age

• if E then 1 else 2       • if E then 1
                                  else true
```

$$\Gamma \vdash n : \texttt{int} \qquad\qquad \Gamma \vdash \mathit{true} : \texttt{bool}$$

$$\Gamma \vdash () : \texttt{unit}$$

$$\frac{\Gamma \vdash E_1 : \texttt{int} \quad \Gamma \vdash E_2 : \texttt{int}}{\Gamma \vdash E_1 + E_2 : \texttt{int}}$$

$$\frac{\begin{array}{c}\Gamma \vdash E_1 : \mathtt{bool} \\ \Gamma \vdash E_2 : \tau \qquad\qquad \Gamma \vdash E_3 : \tau\end{array}}{\Gamma \vdash \mathtt{if}\ E_1\ \mathtt{then}\ E_2\ \mathtt{else}\ E_3 : \tau}$$

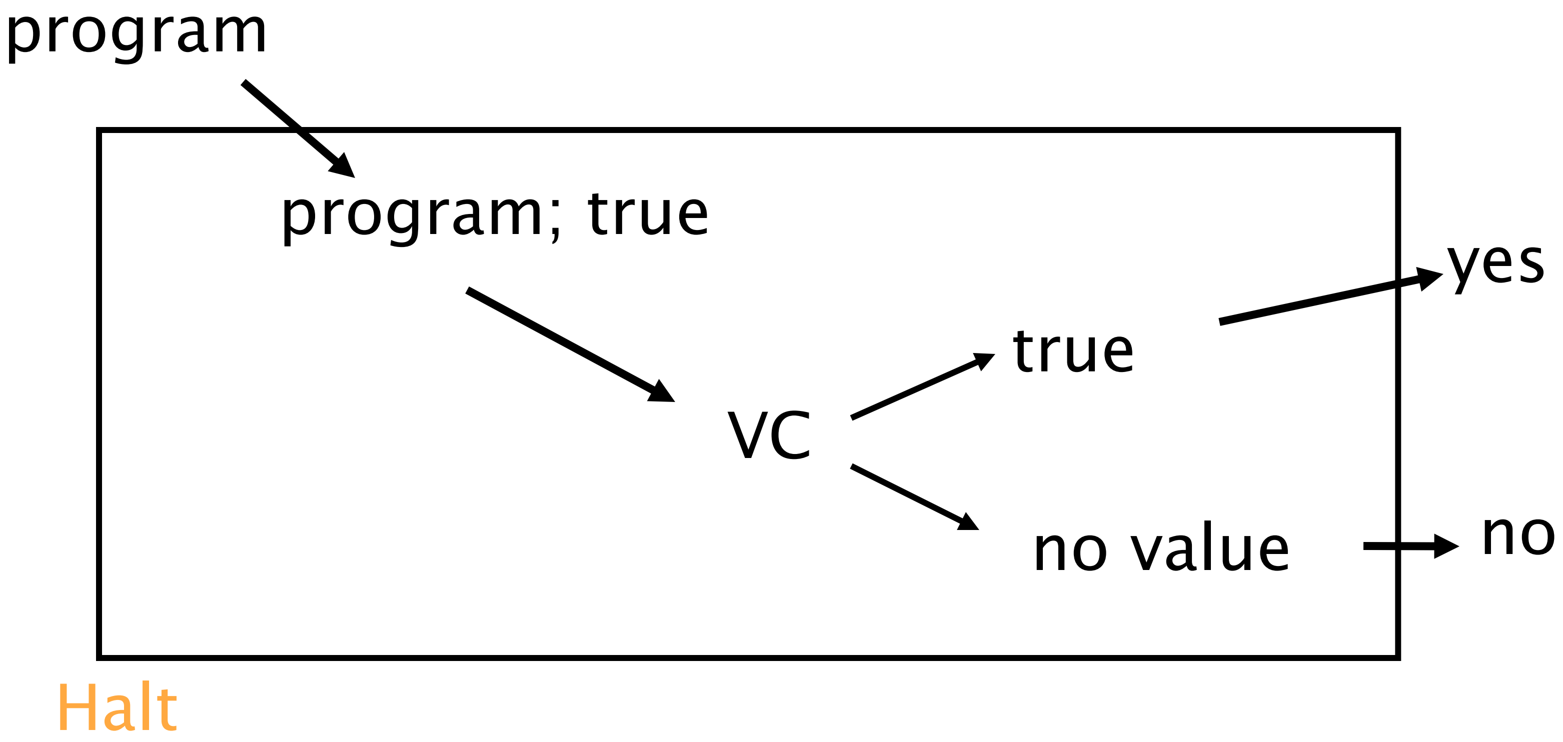
Why two branches should have the same type?

(if E then 1 else true) + 1

혹시 E의 값을 미리 알 수 있지 않을까?
항상 true이므로, 위의 프로그램은 OK!
혹은, 항상 false이므로, no OK!
혹은, true/false모두 가능하므로, no OK!
혹은, 값이 없으므로, OK!

그와같이 E의 값을 프로그램 실행시키지 않고 정확히 알 수 있는 방법은 없습니다.

그와 같은 방법(VC)이 있다면, 제가 Halting problem을 풀어드릴 수 있어요:



$$\frac{\Gamma \vdash E_1 : \texttt{bool} \qquad \Gamma \vdash E_2 : \texttt{unit}}{\Gamma \vdash \texttt{while}\ E_1\ E_2 : \texttt{unit}}$$

Type Checking K- Programs

$$\boxed{\Gamma \vdash E : \tau}$$

$$\Gamma \vdash n : \text{int} \qquad \Gamma \vdash \text{true} : \text{bool} \qquad \Gamma \vdash () : \text{unit}$$

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

$$\frac{\Gamma \vdash E_1 : \tau_1 \quad \Gamma \vdash E_2 : \tau_2}{\Gamma \vdash E_1 ; E_2 : \tau_2}$$

$$\frac{\Gamma \vdash E_1 : \text{bool} \quad \Gamma \vdash E_2 : \tau \quad \Gamma \vdash E_3 : \tau}{\Gamma \vdash \text{if } E_1 \text{ then } E_2 \text{ else } E_3 : \tau}$$

$$\frac{\Gamma \vdash E_1 : \text{bool} \quad \Gamma \vdash E_2 : \text{unit}}{\Gamma \vdash \text{if } E_1 \text{ then } E_2 : \text{unit}}$$

$$\frac{\Gamma \vdash E_1 : \text{bool} \quad \Gamma \vdash E_2 : \text{unit}}{\Gamma \vdash \text{while } E_1 \text{ do } E_2 : \text{unit}}$$

$$\frac{\Gamma(x) = \text{int var} \quad \Gamma \vdash E_1 : \text{int} \quad \Gamma \vdash E_2 : \text{int} \quad \Gamma \vdash E_3 : \text{unit}}{\Gamma \vdash \text{for } x := E_1 \text{ to } E_2 \text{ do } E_3 : \text{unit}}$$

$$\frac{\Gamma \vdash E_1 : \tau_1 \quad \Gamma[\tau_1 \text{ var}/x] \vdash E_2 : \tau_2}{\Gamma \vdash \text{let } x := E_1 \text{ in } E_2 : \tau_2}$$

$$\frac{\Gamma \vdash E_1 : \tau \quad \Gamma(x) = \tau \text{ var}}{\Gamma \vdash x := E_1 : \text{unit}}$$

$$\frac{\Gamma(x) = \tau \text{ var}}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma[\tau \text{ var}/y][\tau \to \tau_1/x] \vdash E_1 : \tau_1 \quad \Gamma[\tau \to \tau_1/x] \vdash E_2 : \tau_2}{\Gamma \vdash \text{let procedure } x(y) = E_1 \text{ in } E_2 : \tau_2}$$

$$\frac{\Gamma \vdash E : \tau_1 \quad \Gamma(x) = \tau_1 \to \tau_2}{\Gamma \vdash \text{call } x(E) : \tau_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash E_1 : \tau_1 \\ \vdots \\ \Gamma \vdash E_n : \tau_n\end{array}}{\Gamma \vdash \{x_1 := E_1; \cdots; x_n := E_n\} : \left\{\begin{array}{l} x_1 \mapsto \tau_1 \text{ var}, \\ \quad\vdots \\ x_n \mapsto \tau_n \text{ var}\end{array}\right\}}$$

$$\frac{\begin{array}{l}\Gamma \vdash E : \tau \\ \tau(x) = \tau_1 \text{ var}\end{array}}{\Gamma \vdash E.x : \tau_1}$$

$$\frac{\begin{array}{ll}\Gamma \vdash E_1 : \tau & \tau(x) = \tau' \text{var} \\ \Gamma \vdash E_2 : \tau' & \end{array}}{\Gamma \vdash E_1.x := E_2 : \text{unit}}$$

$$\frac{\begin{array}{l}\Gamma \vdash E_1 : \tau \\ \Gamma \vdash E_2 : \tau\end{array}}{\Gamma \vdash E_1 = E_2 : \text{bool}}$$

어디 보자
Observations

* 메모리 주소에는 할당될 때의 타입을 유지 시킨다.

```
let x := 1
in x := true
```

"reject!"

* 타입은 하나만 가능하다.

```
let procedure f(x) = x
in f(1); f(true)
```

"reject!"

```
let procedure f(x) = x.age := 19
in f ({age:= 1, id := 001});
   f ({age := 2, height := 170})
```

"reject!"

* 함수 타입을 "유추"해야 할텐데...

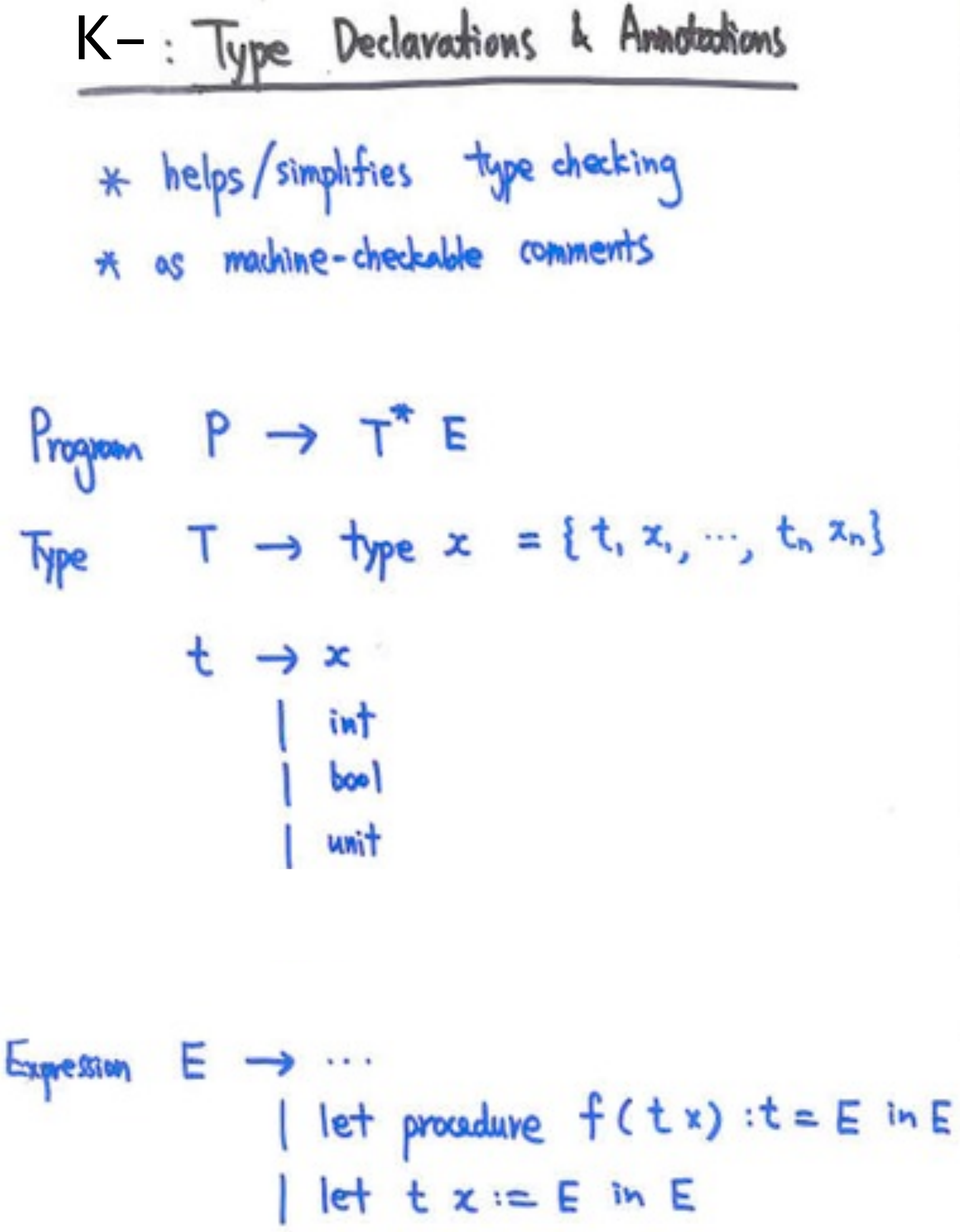
$$\frac{\Gamma[\tau\mathrm{var}/y][\tau \to \tau_1/x] \vdash E_1 : \tau_1 \qquad \Gamma[\tau \to \tau_1/x] \vdash E_2 : \tau}{\Gamma \vdash \texttt{let procedure } x(y) = E_1 \texttt{ in } E_2 : \tau}$$

* 레코드 타입이 너무 깐깐한 건 아닐까?

```
let
  node := { x:=0, next :={}}
in
  node.next := { x:=1, next :={}}
```

"reject!"

프로그래머에게 기대자.



K- : Type Declarations & Annotations

* helps/simplifies type checking

* as machine-checkable comments

Program $P \rightarrow T^* E$

Type $T \rightarrow \text{type } x = \{t_1\ x_1, \cdots, t_n\ x_n\}$

$t \rightarrow x$
$\quad |\ \text{int}$
$\quad |\ \text{bool}$
$\quad |\ \text{unit}$

Expression $E \rightarrow \cdots$
$\quad |\ \text{let procedure } f(t\ x) : t = E \text{ in } E$
$\quad |\ \text{let } t\ x := E \text{ in } E$

e.g.)

```
type  bbs  = { int name,  bool zap }
let  bbs  x := { name := 0,  zap := true }
in  if x.zap  then  1
                else  x.name
```

e.g.)

```
type  intlist  = { int x,  intlist next }
let  intlist  l  := { x := 0,  next := {} }
in
    l.next := { x := 1,  next := {} } ;
    l.next.next  :=  { x := 2,  next := {} }
```

e.g.)

```
type  intree  = { intree l ,  int x,  intree r }
let  procedure  shake ( intree t ) : intree
     = if  t = {}  then  t
         else  let  intree  t' := {}
                   in   t' := call shake (t.l);
                        t.l := call shake(t.r);
                        t.r := t'
in
   call shake ( { l := {},  x := 1,  r := { l := {}, x := 2, r := {} }
                })
```

$$
\begin{array}{llcl}
\Gamma & \in \ \mathit{VarEnv} & = & \mathit{Var} \to \mathit{Type} \\
\Delta & \in \ \mathit{TypeEnv} & = & \mathit{Tname} \to \mathit{Type} \\
\tau & \in \ \mathit{Type} \ \ \tau & \to & \texttt{unit} \,|\, \texttt{bool} \,|\, \texttt{int} \\
 & & | & \tau\texttt{->}\tau \ | \ \tau \ \texttt{var} \\
 & & | & x
\end{array}
$$

$$\boxed{\Delta \vdash T^* : \Delta'} \qquad \boxed{\Gamma, \Delta \vdash E : \tau}$$

$$\frac{\emptyset \vdash T^* : \Delta \qquad \emptyset, \Delta \vdash E : \tau}{\emptyset, \emptyset \vdash T^* \ E : \tau}$$

$$\frac{\Delta \vdash T_1 : \Delta_1 \qquad \Delta_1 \vdash T_2 : \Delta_2}{\Delta \vdash T_1\; T_2 : \Delta_2}$$

$$\frac{x_1 \neq x_2}{\begin{array}{c}\Delta \vdash \texttt{type x =}\{t_1\, x_1,\ t_2\, x_2\} : \\ \Delta[\{x_1 \mapsto t_1\,\texttt{var}, x_2 \mapsto t_2\,\texttt{var}\}/\texttt{x}]\end{array}}$$

$$\frac{\Gamma, \Delta \vdash E : \tau \qquad \Delta(y) = \{x \mapsto \tau\ \texttt{var}\}}{\Gamma, \Delta \vdash \texttt{\{x:=E\}} : y}$$

$$\frac{}{\Gamma, \Delta \vdash \texttt{\{\}} : x}$$

$$\frac{\Gamma[t \to t'/\texttt{f}][t\ \texttt{var}/\texttt{x}], \Delta \vdash E : t'}{\Gamma, \Delta \vdash\ \ \texttt{proc f(}t\ \texttt{x):}t'\ \texttt{=}\ E : \textsf{ok}}$$

어디보자 II
Observations

```
* let  x := { a:=1, b:=2 }  in  x  end

* type 리스트 = { int x, 리스트 next }
  type 트리스 = { int x, 트리스 prev }
  let
        ...  { x:=1, next:={} }
             { x:=1, prev:={} }
  ...

* type t1 = { int x, bool y }
  type t2 = { bool y, int x }
  let ...  { x:=1, y:=true }
  ...

* type 장군 = { int x, 멍군 y }
  type 멍군 = { bool x, 장군 z }
  let 장군 x:={ x:=1, y:={} }
     멍군 x':={ x:=true, z:={} }
  in
     x.y := x';
     x'.z := x
  end
```

```
* type x = {int x, bool y}
  type y = {int id, x employ}
  let
     x x := {x := 1, y := true}
     y y := {id := 2, employ := x}
  in
   . . .
```

name spaces : type names
variable names
field names

* every record type has to be named ?

What would you do to remove this restriction ?

- can you change the language's syntax & type system for it ?
- go ahead

어떻게 흘러와 봤는데,
우리가 정의한 타입 체킹
규칙들이 그럴듯 한 데,
그런데,

어딘가 삼류인 듯 한.
무책임한.
후손들이 웃을. 미개한.

우리의 바램

Theorem [Type Safety]

Let E be a program.
If E is type-checked OK, then it does not go wrong.

Theorem [Type Safety]

Let E be a program.
If {} |- E: t then E does not go wrong.

Theorem [Type Safety]

Let E be a program.
If {} |- E : t then 돌렷({},{},E) runs OK.

```
*   type 리스트 = {int x, 리스트 next}
    let
      리스트 node = {x := 1, next := {}}
    in
      node.next.x
    end
```

type-checked! but cannot run ☹

What would you do to make the type system <u>safe</u>?

- what did we miss?