L30: Type Systems (Continued)

Software Analysis

IIT Guwahati · October 15, 2026

Back to the Original Example

T.java

```
1: class T {
2:   int f(float a, int b,
3:         int[] c) {
4:     if (a)
5:       return b;
6:     else
7:       return c;
8:   }
9: }
```

```
$ javac T.java
T.java:4: error: incompatible types
    if (a)
        ^
  required: boolean
  found:    float
T.java:7: error: incompatible types
    return c;
           ^
  required: int
  found:    int[]
2 errors
```

A More Complex Rule (1/21)

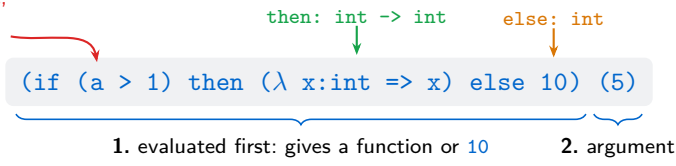
$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

The `if` gets the type `t1` of its then branch.

A More Complex Rule (2/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

a: a **free** variable, an input;
its value is known only at run time;
the checker knows only `a : int`



Question. With `a : int`, does this rule accept this program?
Can it go wrong when it runs?

A More Complex Rule (3/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2}{A \vdash \texttt{if } e0 \texttt{ then } e1 \texttt{ else } e2 : t1} \text{ [If-Then-Else]}$$

$$[a \mapsto int] \vdash \texttt{(if (a > 1) then (}\lambda\texttt{ x:int => x) else 10) (5) : int}$$

A More Complex Rule (4/21)

$$\frac{\texttt{A} \vdash \texttt{e0} : \texttt{bool} \quad \texttt{A} \vdash \texttt{e1} : \texttt{t1} \quad \texttt{A} \vdash \texttt{e2} : \texttt{t2}}{\texttt{A} \vdash \texttt{if e0 then e1 else e2} : \texttt{t1}}\ [\text{If-Then-Else}]$$

$$\frac{[a \mapsto \texttt{int}] \vdash \texttt{if (a > 1) then } (\lambda\ \texttt{x:int => x}) \texttt{ else 10} : \texttt{int -> int} \quad [a \mapsto \texttt{int}] \vdash \texttt{5} : \texttt{int}}{[a \mapsto \texttt{int}] \vdash \texttt{(if (a > 1) then } (\lambda\ \texttt{x:int => x}) \texttt{ else 10) (5)} : \texttt{int}}\ [\texttt{Call}]$$

`[Call]`: the if must be a function `int -> int`, and the argument `5` an `int`.

A More Complex Rule (5/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

$$\frac{[a{\mapsto}int] \vdash \texttt{if (a > 1) then (}\lambda\ \texttt{x:int => x) else 10 : int -> int} \quad \frac{}{[a{\mapsto}int] \vdash \texttt{5 : int}}\ [\text{Int}]}{[a{\mapsto}int] \vdash \texttt{(if (a > 1) then (}\lambda\ \texttt{x:int => x) else 10) (5) : int}}\ [\text{Call}]$$

[Int]: 5 is an integer literal. An axiom, so this goal is closed.

A More Complex Rule (6/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ \texttt{[If-Then-Else]}$$

$$\frac{\dfrac{\dfrac{[a{\mapsto}int] \vdash a > 1 : bool \quad [a{\mapsto}int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int}{[a{\mapsto}int] \vdash \texttt{if (a > 1) then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else 10} : int \to int}\ \texttt{[If-Then-Else]} \quad \dfrac{[a{\mapsto}int] \vdash 5 : int}{[a{\mapsto}int] \vdash 10 : int}\ \texttt{[Int]}}{[a{\mapsto}int] \vdash \texttt{(if (a > 1) then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else 10) (5)} : int}\ \texttt{[Call]}$$

`[If-Then-Else]`: the if gets the type of its then branch, `int -> int`.
`a > 1 : bool` we take as given (the rules for booleans are not shown).

A More Complex Rule (7/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

$$\dfrac{\dfrac{\dfrac{[a\mapsto int] \vdash a > 1 : bool}{\dfrac{[a\mapsto int][x\mapsto int] \vdash x : int}{[a\mapsto int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int}\ [\text{Def}]} \quad \dfrac{}{[a\mapsto int] \vdash 10 : int}}{[a\mapsto int] \vdash \texttt{if (a > 1) then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else 10} : int \to int}\ [\text{If-Then-Else}] \quad \dfrac{}{[a\mapsto int] \vdash 5 : int}\ [\text{Int}]}{[a\mapsto int] \vdash \texttt{(if (a > 1) then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else 10) (5)} : int}\ [\text{Call}]$$

[Def]: type the body x with x↦int added to the environment.

A More Complex Rule (8/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

$$\dfrac{\dfrac{\dfrac{}{[a\mapsto int] \vdash a > 1 : bool}\ [Var]}{[a\mapsto int][x\mapsto int] \vdash x : int}}{[a\mapsto int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int}\ [Def] \quad \dfrac{}{[a\mapsto int] \vdash 10 : int}\ [Int]}{[a\mapsto int] \vdash \texttt{if (a > 1) then (}\lambda\ x{:}int \Rightarrow x\texttt{) else 10} : int \to int}\ [If\text{-}Then\text{-}Else] \quad \dfrac{}{[a\mapsto int] \vdash 5 : int}\ [Int]}{[a\mapsto int] \vdash \texttt{(if (a > 1) then (}\lambda\ x{:}int \Rightarrow x\texttt{) else 10) (5)} : int}\ [Call]$$

`[Var]` gives `x : int`; `[Int]` gives `10 : int`. No open goal is left: the tree is a complete derivation, so the program has type `int`: **accepted** ✓

A More Complex Rule (9/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

$$\dfrac{\dfrac{\dfrac{}{[a\mapsto int] \vdash a > 1 : bool} \quad \dfrac{\dfrac{}{[a\mapsto int][x\mapsto int] \vdash x : int}[\text{Var}]}{[a\mapsto int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int}[\text{Def}] \quad \dfrac{}{[a\mapsto int] \vdash 10 : int}[\text{Int}]}{[a\mapsto int] \vdash \text{if } (a > 1) \text{ then } (\lambda\ x{:}int \Rightarrow x) \text{ else } 10 : int \to int}[\text{If-Then-Else}] \quad \dfrac{}{[a\mapsto int] \vdash 5 : int}[\text{Int}]}{[a\mapsto int] \vdash (\text{if } (a > 1) \text{ then } (\lambda\ x{:}int \Rightarrow x) \text{ else } 10)\ (5) : int}[\text{Call}]$$

Run with `a = 3`: $(\lambda\ x{:}int \Rightarrow x)\ (5) \to 5$ ✓

A More Complex Rule (10/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

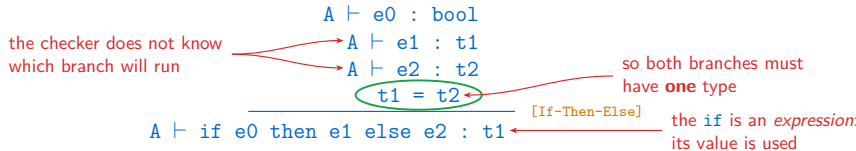
$$\dfrac{\dfrac{\dfrac{}{[a\mapsto int] \vdash a > 1 : bool}\ [\text{Var}]}{[a\mapsto int][x\mapsto int] \vdash x : int}\ [\text{Def}]}{[a\mapsto int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int} \quad \dfrac{}{[a\mapsto int] \vdash 10 : int}\ [\text{Int}]$$

$$\dfrac{[a\mapsto int] \vdash \text{if } (a > 1) \text{ then } (\lambda\ x{:}int \Rightarrow x) \text{ else } 10 : int \to int \quad [a\mapsto int] \vdash 5 : int}{[a\mapsto int] \vdash (\text{if } (a > 1) \text{ then } (\lambda\ x{:}int \Rightarrow x) \text{ else } 10)\ (5) : int}\ [\text{Call}]$$

$$[\text{If-Then-Else}] \quad [\text{Int}]$$

Run with `a = 0`: `10 (5)` $\rightarrow$ calls an integer: **goes wrong** $\times$. This rule is **unsound**.

A More Complex Rule (11/21)



Solution: add `t1 = t2`, a **side condition**: a constraint between two types, not a judgment.

A More Complex Rule (12/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2 \quad t1 = t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ \text{[If-Then-Else]}$$

$$[a \mapsto int] \vdash \texttt{(if (a > 1) then (}\lambda\ \texttt{x:int => x) else 10) (5)} : int$$

Check again with the **fixed rule**, bottom up from the same goal.

A More Complex Rule (13/21)

$$\frac{\texttt{A} \vdash \texttt{e0 : bool} \quad \texttt{A} \vdash \texttt{e1 : t1} \quad \texttt{A} \vdash \texttt{e2 : t2} \quad \texttt{t1 = t2}}{\texttt{A} \vdash \texttt{if e0 then e1 else e2 : t1}}\ \texttt{[If-Then-Else]}$$

$$\frac{\texttt{[a↦int]} \vdash \texttt{if (a > 1) then (λ x:int => x) else 10 : int -> int} \quad \texttt{[a↦int]} \vdash \texttt{5 : int}}{\texttt{[a↦int]} \vdash \texttt{(if (a > 1) then (λ x:int => x) else 10) (5) : int}}\ \texttt{[Call]}$$

`[Call]`, as before: the `if` must be `int -> int`, and `5` an `int`.

A More Complex Rule (14/21)

$$\frac{\texttt{A} \vdash \texttt{e0} : \texttt{bool} \quad \texttt{A} \vdash \texttt{e1} : \texttt{t1} \quad \texttt{A} \vdash \texttt{e2} : \texttt{t2} \quad \texttt{t1} = \texttt{t2}}{\texttt{A} \vdash \texttt{if e0 then e1 else e2} : \texttt{t1}}\ [\text{If-Then-Else}]$$

$$\frac{[a{\mapsto}\texttt{int}] \vdash \texttt{if (a > 1) then } (\lambda\ \texttt{x:int => x}) \texttt{ else 10} : \texttt{int -> int} \qquad \frac{}{[a{\mapsto}\texttt{int}] \vdash \texttt{5} : \texttt{int}}\ [\texttt{Int}]}{[a{\mapsto}\texttt{int}] \vdash \texttt{(if (a > 1) then } (\lambda\ \texttt{x:int => x}) \texttt{ else 10) (5)} : \texttt{int}}\ [\texttt{Call}]$$

[Int] closes `5 : int`.

A More Complex Rule (15/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2 \quad t1 = t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ \texttt{[If-Then-Else]}$$

$$\frac{\dfrac{\dfrac{[a\mapsto int] \vdash a > 1 : bool \quad [a\mapsto int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int}{[a\mapsto int] \vdash (a > 1)\ then\ (\lambda\ x{:}int \Rightarrow x)\ else\ 10 : int \to int}\ \texttt{[If-Then-Else]} \quad \dfrac{[a\mapsto int] \vdash 10 : int \quad t1 = t2}{[a\mapsto int] \vdash 5 : int}\ \texttt{[Int]}}{[a\mapsto int] \vdash (if\ (a > 1)\ then\ (\lambda\ x{:}int \Rightarrow x)\ else\ 10)\ (5) : int}\ \texttt{[Call]}$$

[If-Then-Else], fixed: the same three hypotheses, plus the side condition `t1 = t2`.

A More Complex Rule (16/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2 \quad t1 = t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ \texttt{[If-Then-Else]}$$

$$\frac{\dfrac{[a \mapsto int] \vdash a > 1 : bool}{[a \mapsto int][x \mapsto int] \vdash x : int}\ \texttt{[Def]}}{[a \mapsto int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int} \qquad \frac{[a \mapsto int] \vdash 10 : int}{t1 = t2}\ \texttt{[If-Then-Else]} \qquad \frac{}{[a \mapsto int] \vdash 5 : int}\ \texttt{[Int]}}{[a \mapsto int] \vdash (\text{if } (a > 1) \text{ then } (\lambda\ x{:}int \Rightarrow x) \text{ else } 10)\ (5) : int}\ \texttt{[Call]}$$

[Def]: type the body x with x↦int added to the environment.

A More Complex Rule (17/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2 \quad t1 = t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

$$\dfrac{\dfrac{\dfrac{}{[a\mapsto int] \vdash a > 1 : bool}}{[a\mapsto int][x\mapsto int] \vdash x : int}[\text{Var}]}{[a\mapsto int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int}[\text{Def}] \quad \dfrac{}{[a\mapsto int] \vdash 10 : int}[\text{Int}]}{[a\mapsto int] \vdash \texttt{if (a > 1) then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else 10} : int \to int}[\text{If-Then-Else}] \quad \dfrac{}{[a\mapsto int] \vdash 5 : int}[\text{Int}]$$

$$[a\mapsto int] \vdash \texttt{(if (a > 1) then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else 10) (5)} : int \quad [\text{Call}]$$

t1 = t2

`[Var]` closes `x : int`; `[Int]` closes `10 : int`. Only the side condition is left.

A More Complex Rule (18/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2 \quad t1 = t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[a\mapsto int] \vdash a > 1 : bool}{[a\mapsto int][x\mapsto int] \vdash x : int}\ [Var]}{[a\mapsto int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int}\ [Def] \quad \dfrac{[a\mapsto int] \vdash 10 : int}{int \to int = t2}\ [Int]}{[a\mapsto int] \vdash \texttt{if (a > 1) then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else 10} : int \to int}\ [\text{If-Then-Else}] \quad \dfrac{}{[a\mapsto int] \vdash 5 : int}\ [Int]}{[a\mapsto int] \vdash \texttt{(if (a > 1) then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else 10) (5)} : int}\ [Call]$$

The then branch matches `A ⊢ e1 : t1`, so `t1 = int -> int`.

A More Complex Rule (19/21)

$$\frac{\texttt{A} \vdash \texttt{e0} : \texttt{bool} \quad \texttt{A} \vdash \texttt{e1} : \texttt{t1} \quad \texttt{A} \vdash \texttt{e2} : \texttt{t2} \quad \texttt{t1} = \texttt{t2}}{\texttt{A} \vdash \texttt{if e0 then e1 else e2} : \texttt{t1}}\ [\text{If-Then-Else}]$$

$$\frac{\dfrac{\dfrac{}{[a{\mapsto}int] \vdash a > 1 : bool}}{[a{\mapsto}int][x{\mapsto}int] \vdash x : int}\ [Var]}{[a{\mapsto}int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int}\ [Def] \quad \frac{}{[a{\mapsto}int] \vdash 10 : int}\ [Int]}{[a{\mapsto}int] \vdash \texttt{if } (a > 1) \texttt{ then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else } 10 : int \to int}\ [\text{If-Then-Else}] \quad \frac{}{[a{\mapsto}int] \vdash 5 : int}\ [Int]$$

$$\frac{[a{\mapsto}int] \vdash \texttt{if } (a > 1) \texttt{ then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else } 10 : int \to int \quad [a{\mapsto}int] \vdash 5 : int}{[a{\mapsto}int] \vdash (\texttt{if } (a > 1) \texttt{ then } (\lambda\ x{:}int \Rightarrow x) \texttt{ else } 10)\ (5) : int}\ [Call]$$

$$\texttt{int} \to \texttt{int} = \texttt{int}$$

The else branch matches $\texttt{A} \vdash \texttt{e2} : \texttt{t2}$, so $\texttt{t2} = \texttt{int}$.

A More Complex Rule (20/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2 \quad t1 = t2}{A \vdash \texttt{if e0 then e1 else e2} : t1}\ [\text{If-Then-Else}]$$

$$\dfrac{\dfrac{\dfrac{}{[a\mapsto int] \vdash a > 1 : bool}\ [\text{Var}]}{[a\mapsto int][x\mapsto int] \vdash x : int}\ [\text{Def}]}{[a\mapsto int] \vdash \lambda\ x{:}int \Rightarrow x : int \to int} \quad \dfrac{}{[a\mapsto int] \vdash 10 : int}\ [\text{Int}]$$

$$\dfrac{\dfrac{}{[a\mapsto int] \vdash \texttt{if (a > 1) then (}\lambda\ x{:}int \Rightarrow x\texttt{) else 10} : int \to int}\ \times \quad \texttt{int -> int = int ? no}}{[a\mapsto int] \vdash \texttt{(if (a > 1) then (}\lambda\ x{:}int \Rightarrow x\texttt{) else 10) (5)} : int}\ \times\ [\text{Call}]$$

$$\dfrac{}{[a\mapsto int] \vdash 5 : int}\ [\text{Int}]$$

Is `int -> int = int`? No. `[If-Then-Else]` cannot be used, so the `if` gets no type and `[Call]` fails too. The program is **rejected** before it runs.

A More Complex Rule (21/21)

$$\frac{A \vdash e0 : bool \quad A \vdash e1 : t1 \quad A \vdash e2 : t2 \quad t1 = t2}{A \vdash \texttt{if } e0 \texttt{ then } e1 \texttt{ else } e2 : t1}\ [\text{If-Then-Else}]$$

Java: `if` is a *statement*; it has no value, so the branches need not agree:

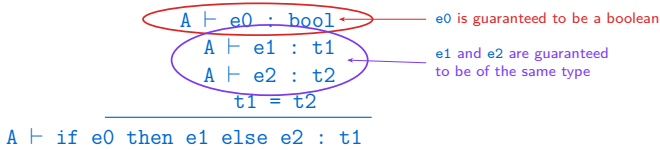
```
if (c) { int a = 1; } else { String b = "a"; }
```

Java's *expression* `c ? e1 : e2` does need a common type:

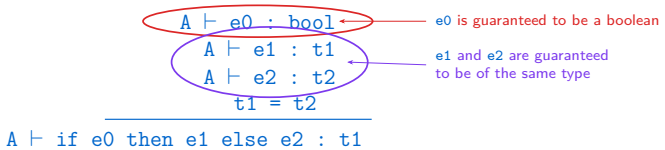
```
int y = c ? 1 : "a";
```

rejected: `"a"` is not an `int`

Type Soundness (1/3)



Type Soundness (2/3)



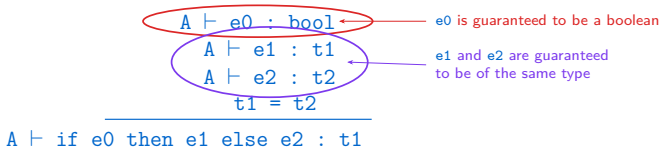
A type system is **sound** iff

whenever 1. `A ⊢ e : t` and

2. if `A(x) = t'`, then `x` has a value `v'` in `t'`

then `e` evaluates to a value `v` in `t`

Type Soundness (3/3)



A type system is **sound** iff

whenever 1. `A ⊢ e : t` and

2. if `A(x) = t'`, then `x` has a value `v'` in `t'`

then `e` evaluates to a value `v` in `t`

Example

1. `[x↦int] ⊢ x + 1 : int` (by `[Add]`, `[Var]`, `[Int]`)
2. `x = 41`, and `41` is in `int`

then `x + 1 → 42`, and `42` is in `int` ✓

Comments on Soundness (1/2)

- Soundness is extremely useful

  Program type-checks $\Rightarrow$ no errors at runtime

  Verifies absence of a class of errors

Comments on Soundness (2/2)

- Soundness is extremely useful

  Program type-checks $\Rightarrow$ no errors at runtime

  Verifies absence of a class of errors

- This is a very strong guarantee

  Verified property holds in all executions

  "Well-typed programs cannot go wrong" (Robin Milner, 1934–2010; Turing Award, 1991)

The Price of Soundness (1/3)

- Soundness comes at a price: **false positives**

  an error is reported, but no execution has it

The Price of Soundness (2/3)

- Soundness comes at a price: **false positives**

  an error is reported, but no execution has it

- Alternative: use unsound analysis

  Reduces **false positives**

  Introduces **false negatives**: a real error is not reported

The Price of Soundness (3/3)

- Soundness comes at a price: **false positives**

  an error is reported, but no execution has it

- Alternative: use unsound analysis

  Reduces **false positives**

  Introduces **false negatives**: a real error is not reported

- Type systems are sound

  But most **bug finding** analyses are not sound

QUIZ: Properties of Type Systems (1/2)

Check the below untypable programs that can "go wrong" (some execution gives an error):

Untypable: no derivation of `A ⊢ e : t` exists, for any type `t`; the checker rejects `e`.
E.g. `42 (5)`: `[Call]` needs `42 : t' -> t`, but the only rule for `42` is `[Int]`, which gives `int`.

`42 (λ x:int => (x + 5))`	☐	
`(λ x:int => x) + 1`	☐	
`if (true) then 1 else ((λ x:int => x) + 1)`	☐	

`(if (c != 0) then (λ x:int => x)`
`else (λ x:int->int => (x 1)))`
`(if (c != 0) then 1`
`else (λ z:int => z))`
	☐	

QUIZ: Properties of Type Systems (2/2)

Check the below untypable programs that can "go wrong" (some execution gives an error):

Untypable: no derivation of `A ⊢ e : t` exists, for any type `t`; the checker rejects `e`.
E.g. `42 (5)`: `[Call]` needs `42 : t' -> t`, but the only rule for `42` is `[Int]`, which gives `int`.

`42 (λ x:int => (x + 5))`	☒	calls 42: every run
`(λ x:int => x) + 1`	☒	adds a function: every run
`if (true) then 1 else ((λ x:int => x) + 1)`	☐	bad branch never runs

`(if (c != 0) then (λ x:int => x)`
`  else (λ x:int->int => (x 1)))`
`(if (c != 0) then 1`
`  else (λ z:int => z))`
	☐	every run well defined

Our type system rejects all four. The last two never go wrong, so rejecting them is a **false positive**: the type system is **incomplete**.