# Lecture 22. Typing Systems

$$\Gamma \vdash e : t$$

- $\Gamma$: Typing Context
- $e$: Expr
- $t$: Type

$\{ x_1 : t_1, x_2 : t_2, \ldots \}$

$e ::= x \mid i \mid b \mid e + e \mid \ldots$

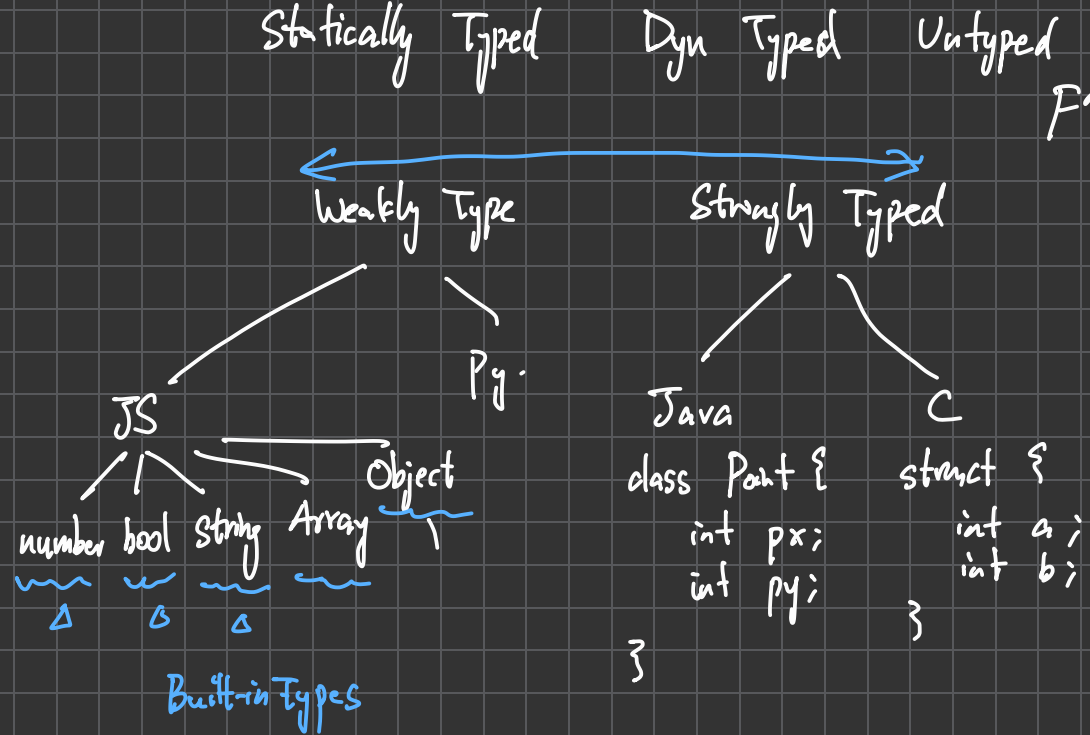
$t ::= \text{Int} \mid \text{Bool} \mid t \to t \mid \text{Ref } t$

TFBS: $\text{Ref}(\text{Ref}(\text{Int})) \to$

C/C++: `int **` (`((int) *) *`)

ptr ptr

# Lecture 22. Type System II




---

$F^b$ $\quad e ::= x \mid i \mid b \mid \text{Let } x = e \text{ In } e$

$TF^b$ ← $\quad e ::= x \mid i \mid b$

$\quad \mid \text{Let } x : t = e \text{ In } e$

$TF^bSRX$ (state, record, exceptions)

Why do we need to add annotation

$\quad \mid \text{Fun } x : t \to e$

$\quad \mid e + e \mid \cdots$

$t ::= \text{Int} \mid \text{Bool} \mid t \to t$

# Type Annotation

Opt-in  Language don't force type anno
but you can add it

Opt-out  Languages normally force type anno
but you can choose to not provide it

C++

```
auto i = 3;
```

↑ Type inference

```
{ add = fun x -> fun y -> x+y                    // int -> int -> int
```

← very minimal opt-in

```
( (fun (x:int) : (int->int) ->
      (fun (y:int) : int ->
          ((x:int) + (y:int)) : int)
  ) : (int -> int)
) : (int -> int -> int)
```

← very excessive

```
{ let add: int -> int -> int =
      fun (x:int) -> fun (y:int) -> x + y
```

← moderate

balancing

Amount of Type Annotation ⟷ Expressiveness / Automation of Type System

$TF^b$ : whenever we introduce variables we add type anno

This Course { Single Directional / Monotonic

OCaml / Haskell :

do not need to specify any type

infer "most-general" type

Hindley Milner Type System

'a → 'a

---

$\Gamma \vdash e : t$

Typing Context — expr — type

under context $\Gamma$, the expr $e$ is typed as $t$

$\{ x_1 : t_1, x_2 : t_2, \dots \}$

$t ::= \underline{Int}$
$\mid \underline{Bool}$
$\mid \underline{t \to t}$
$\mid t \mid t$

$\{\} \vdash e : t \equiv \vdash e : t$

$\Delta \phi$

$TF^b$

$e ::= x \mid i \mid b$

variable, integer, bool

$F^b$ Sem

[int] $\quad i \Rightarrow i$

$TF^b$ Typing rules

[Bool] $\dfrac{}{\Gamma \vdash b : \text{Bool}}$

[Int] $\dfrac{}{\Gamma \vdash i : \text{Int}}$

[Variable] $\dfrac{\Gamma(x) = t}{\Gamma \vdash x : t}$

[Add] $\dfrac{\Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int}}{\Gamma \vdash e_1 + e_2 : \boxed{\text{Int}}}$

$e_2, e_3$ of the same type

[ITE] $\dfrac{\Gamma \vdash e_1 : \text{Bool} \quad \Gamma \vdash e_2 : t \quad \Gamma \vdash e_3 : t}{\Gamma \vdash \text{If } e_1 \text{ Then } e_2 \text{ Else } e_3 : t}$

$v_1 \quad v_3$

Python

return None

Java

```
Point f() {
  if (cond) {
    return new Point(1, 2);
  } else {
    return null;
  }
}
```

Object | null

---

[Let] $$\frac{\Gamma \vdash e_1 : t \qquad \Gamma \cup \{x : t\} \vdash e_2 : t'}{\Gamma \vdash \text{Let } x : t = e_1 \text{ In } e_2 : t'}$$

[Abs] $$\frac{\Gamma \cup \{x : t\} \vdash e : t_2}{\Gamma \vdash \text{Fun } x : t \to e : t \to t_2}$$

[App] $$\frac{\Gamma \vdash e_1 : t_1 \to t_2 \qquad \Gamma \vdash e_2 : t_1}{\Gamma \vdash e_1\ e_2 : t_2}$$

$\Gamma \vdash e : t$

ternary relation

$\Gamma$ $e$ $t$

type_check( context, expr, type) → 𝔹

not practical

---

{ type_check (context, expr) ⇒ type

will terminate

t

---

## Subtyping / Polymorphism

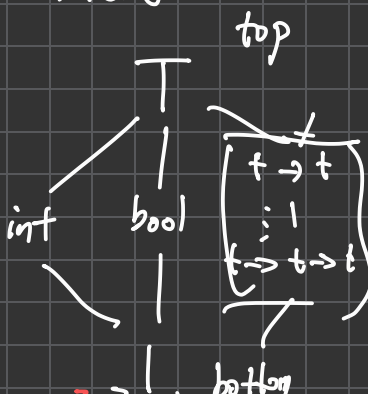
There are cases where you want to have multiple types used in the same context

TF^b R

get_x =
Fun p : {x : Int} → p.x

get_x { x : 1 ; y : 2 }

get_x { x : 1 ; y : 2 ; z : 3 }



TF$^b$R

$t ::= Int \mid Bool \mid t \to t$

$\mid \{ x : t, \ldots \}$ ← Record Type

one type $\{x : t, y : t', z : t'', \ldots\}$ can always be used in the context where $\{x : t\}$ is expected

Subtype

$t_1 <: t_2$ ← subtype

partial order

$t_1$ can be used anywhere $t_2$ is expected

$\bot <: t$

Bottom ← $\bot$

Bottom type is subtype of Any

$t <: \top$

Any type is subtype of Top

Rust

panic!(..) ← return $\bot$

unimplemented!(..) ↙

Java

```
int f() {
  if (..) throw Exception
  else return 1
}
```

$TF^b S R X$

Ref

subtype

---

$TF^b S$ $\quad t ::= (\ldots TF^b \ldots) \mid Ref\ t$

$(((int)*)*)$ C

$\Leftarrow$ Ref (Ref (Int))

$$\dfrac{\dfrac{\dfrac{\{*\}(x) = Int}{\{x:Int, y:Int\} \vdash x : Int}[Var] \quad \dfrac{\{*\}(y) = Int}{\{x:Int, y:Int\} \vdash x : Int}[Var]}{\{x:Int,\ y:Int\} \vdash x + y : Int}[Add]}{\{x:Int\} \vdash Fun\ y \to x + y : Int \to Int}[Abs]}{\Delta \vdash Fun\ x \to Fun\ y \to x + y : Int \to (Int \to Int)}[Abs]$$

$x : Int$ $\quad y : Int$