

Understanding types

$$3 + 5 \Rightarrow^* 8$$

$$(\text{if } 3 \leq 5 \text{ then } 3 \text{ else } 5) + 8 \Rightarrow^* 11$$

$$3 + \text{true} \Rightarrow^* ?$$

$$\text{if } 3 \text{ then } 4 \text{ else } 5 \Rightarrow^* ?$$

Language L_1 (Untyped expressions)

$$e ::= \underbrace{0 \mid 1 \mid 2 \mid \dots}_{c \in \text{Int}} \mid \underbrace{\text{true} \mid \text{false}}_{\text{Booleans}}$$

$$\mid e_1 + e_2 \mid e_1 - e_2$$

$$\mid e_1 \text{ and } e_2 \mid e_1 \text{ or } e_2 \mid \text{not } e_1$$

$$\mid e_1 \leq e_2$$

| if e_1 then e_2 else e_3

Language L_2 (Separating Ints & Bools)

Arithmetic expressions

$a ::= 0 \mid 1 \mid 2 \mid \dots$
 └──────────┘
 $c \in \text{Int}$

| $a_1 \pm a_2$

| if b then a_1 else a_2

Boolean expressions

$b ::= \text{true} \mid \text{false}$
 └──────────┘
 Booleans

| b_1 and b_2 | b_1 or b_2

| not b_1

| if b_1 then b_2 else b_3

|||

$(b_1 \text{ and } b_2) \text{ or}$
 $(\text{not } b_1 \text{ and } b_3)$

Grammar in L_2 already encodes our typing rules

Guarantee: No runtime type errors.

Problem : What if the language has
infinitely many "types"?

Language L_3 (Ints, Bools, Lists)

$e ::= c \in \text{Int} \mid c \in \text{Bool} \mid [e_1; e_2; \dots; e_k]$

$\mid e_1 \pm e_2 \mid e_1 \text{ and/or } e_2 \mid \text{not } e_1$

$\mid e_1 [e_2]$

$\mid \text{if } e_1 \text{ then } e_2 \text{ else } e_3$

Int, Bool, List

$[3; 4; 5][1] + 2 \Rightarrow^* 6$

$[\text{true}; \text{false}; \text{false}][1] + 2 \Rightarrow^* X$

- Need to distinguish

List [Int] from List [Bool]

from List [List [Int]]

List [List [Bool]]

⋮

$a ::= c \in \text{Int} \mid \leftarrow \cancel{c \in \text{Bool}} \mid \cancel{[e_1; e_2; \dots; e_k]}$

$\mid \cancel{a_1 \pm a_2} \mid \cancel{e_1 \text{ and/or } e_2} \mid \cancel{\text{not } e_1}$

$\mid \cancel{a_1 [a_2]} \quad b_1 [a_2]$

$\mid \text{if } \cancel{e_1} \text{ then } a_2 \text{ else } a_3$
b

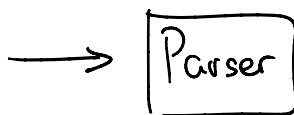
$b ::= \cancel{c \in \text{Int}} \mid c \in \text{Bool} \mid [a_1; a_2; \dots; a_k]$

$\mid \cancel{e_1 \pm e_2} \mid \cancel{e_1 \text{ and/or } e_2} \mid \cancel{\text{not } e_1}$

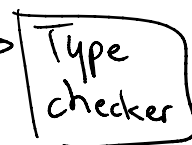
$\mid \cancel{b_1 [a_2]} \quad b_1 [a_2]$

$\mid \text{if } b_1 \text{ then } b_2 \text{ else } b_3$

Program
Source
code



→ AST



Yes, program
is sensible
→ ...

No, the program
...
..

code

1 - ...
No, the program is possibly nonsensical

$(\text{if true then } 3 \text{ else false}) + 3$

How to make sense of expressions in L_3

if c is an integer literal,
then c is of type Int

$c \in \text{Int}$] If everything above the line is true

$c : \text{Int}$] Then everything below the line is true



Typing judgment

$c \in \text{Bool}$	$e_1 : T \quad e_2 : T \quad \dots \quad e_k : T$
<hr/>	<hr/>
$c : \text{Bool}$	$[e_1; e_2; \dots; e_k] : \text{List } T$

$e_1 : \text{Int} \quad e_2 : \text{Int}$] If e_1 is of type Int
& e_2 is of type Int

$e_1 + e_2 : \text{Int}$
 $e_1 - e_2 : \text{Int}$] then (we define)
 $e_1 + e_2$ to have type Int .

$$\frac{e_1 : \text{Int} \quad e_2 : \text{Int}}{e_1 \leq e_2 : \text{Bool}}$$

$$\frac{e_1 : \text{Bool} \quad e_2 : \text{Bool}}{e_1 \text{ and } e_2 : \text{Bool}}$$
$$\text{not } e_1 : \text{Bool}$$

$$\frac{e_1 : \text{List}[T] \quad e_2 : \text{Int}}{e_1[e_2] : T}$$

z : Int

x : Int

x + 3 : Int

x : { Int where $_ < 3$ }

arr [x]

arr : { List [Int] with len > x }

x : { Int > 0 }

$$\frac{e_1 : \text{Bool} \quad e_2 : T \quad e_3 : T}{}$$

If e_1 then e_2 else $e_3 : T$

Claim : Well-typed programs do not get stuck.

Ill-typed programs may get stuck.

	Static	Dynamic
Strong	Agda, Coq, Idris Haskell, Rust OCaml	Python
Weak	Java C++-98 C	Javascript
		Perl, Bash

Origin of types : Late 1800s / early 1900s

Whitehead & Russell: Principia Mathematica

$A = \{ x \notin x \}$: All elements which don't belong to themselves

$A \in A$

$\Rightarrow A \notin A$

$A \notin A$

$\Rightarrow A \in A$

Russell's Paradox

-- n & n

- REA

Paradox

Language L_4 : (Integers & functions)

Expressions $e ::=$

- $c \in \text{Int}$
- $x \in \text{Var}$
- $e_1 \pm e_2$
- $e_1 \ e_2$
- $\text{fun } (x:T) \rightarrow e$
 ↑
 Var

Types, $T ::=$ Int | $T_1 \rightarrow T_2$

Functions which take values of type T_1 as input & produce values of type T_2 .

Infinite many types

Int	Int \rightarrow Int	Int \rightarrow (Int \rightarrow Int)
3	(fun x \rightarrow x+2)	fun x \rightarrow fun y \rightarrow x+y
	(Int \rightarrow Int) \rightarrow Int	...
	fun f \rightarrow f 0	

Typing rules for L_4

$c \in \text{Int}$	$(x:T) \in \text{Ctx}$	\vdash : Turnstile
$\text{Ctx} \vdash c : \text{Int}$	$\text{Ctx} \vdash x : T$	Ctx, Γ

$\text{Ctx} \vdash e_1, e_2 : \text{Int}$	$\text{Ctx} \vdash e_1 : T' \rightarrow T$	$\text{Ctx} \vdash e_2 : T'$
$\text{Ctx} \vdash e_1 + e_2 : \text{Int}$	$\text{Ctx} \vdash e_1 \ e_2 : T$	

$\text{Ctx}, x:T \vdash e : T'$
$\text{Ctx} \vdash \text{fun } (x:T) \rightarrow e : T \rightarrow T'$

let x=3 in X: Int

let x=false in X: Bool

let $x=3$ in $x+x$

let $x=false$ in $x+x$

Example : $(\text{fun } (x: \text{Int} \rightarrow \text{Int}) \rightarrow x \ 5)$ $(\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int}$
 $(\text{fun } (y: \text{Int}) \rightarrow y+1)$: Int

① Empty $\text{Ctx} \vdash (\text{fun } (y: \text{Int}) \rightarrow \underbrace{y+1}) : \text{Int} \rightarrow \text{Int}$

$(y: \text{Int}) \vdash (y+1) : \text{Int}$

② $(\text{fun } (\underbrace{x: \text{Int} \rightarrow \text{Int}}) \rightarrow \underbrace{x \ 5}) : (\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int}$

$(x: \text{Int} \rightarrow \text{Int}) \vdash (\underbrace{x \ 5}) : \text{Int}$
 $\text{Ctx} \vdash 5 : \text{Int}$

$x: \text{Int} \rightarrow \text{Int}$