Lecture 17

Tuesday, October 20, 2020 3:55 PM

Types

Language L_1

Expressions $e ::= \underbrace{0 \mid 1 \mid 2 \mid \cdots}_{c \in Int} \mid \underbrace{true \mid false}_{c \in Bool}$

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

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

$\mid e_1 \leq e_2$

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

$(\text{if } (3 \leq 4) \text{ then } 8 \text{ else } 9) + 5$

$\Rightarrow^* 13$

$true + 5 \Rightarrow^* ?$

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

Arithmetic Expressions

$a ::= c \in Int$

$0 \mid 1 \mid 2 \mid \cdots$

$\mid a_1 \pm a_2$

$\mid \text{if } \underline{b} \text{ then } a_1 \text{ else } a_2$

Boolean expressions

$b ::= c \in Bool$

$true \mid false$

$\mid a_1 \leq a_2$

$\mid b_1 \begin{smallmatrix}\text{and}\\ \text{or}\end{smallmatrix} b_2 \mid \text{not } b_1$

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

We introduced two "types": int & bool.

<u>Guarantee</u>: No runtime type errors.

<u>Language L_2</u>: Integers, Booleans, Lists.

$$
\begin{aligned}
e ::= \; & c \in \mathrm{Int} \mid c \in \mathrm{Bool} \mid [e_1; e_2; e_3; \cdots; e_k] \\
\mid \; & e_1 + e_2 \mid e_1 - e_2 \\
\mid \; & e_1 \le e_2 \mid e_1 \text{ and } e_2 \mid e_1 \text{ or } e_2 \mid \text{not } e_1 \\
\mid \; & e_1 [e_2] \\
\mid \; & \text{if } e_1 \text{ then } e_2 \text{ else } e_3
\end{aligned}
$$

<u>Types of terms in L_2</u>

$$T ::= \mathrm{Int} \mid \mathrm{Bool} \mid \mathrm{List}[T]$$

Since there are infinitely many types

$\mathrm{List}[\mathrm{Int}]$ $\mathrm{List}[\mathrm{List}[\mathrm{Int}]]$ $\mathrm{List}[\mathrm{List}[L[\mathrm{Bool}]]]$ ...

we cannot refine the abstract syntax to prevent nonsensical programs from being written.

Program $\rightarrow$ $\boxed{\text{Parser}}$ $\rightarrow$ AST $\rightarrow$ $\boxed{\text{Type checker}}$ $\rightarrow$ ...

Program $\rightarrow$ Parser $\rightarrow$ AST $\rightarrow$ Type checker $\rightarrow \cdots$

Typing rules for L_2 "$e : T$" e is of type T.

$$\frac{c \in \mathrm{Int}}{c : \mathrm{Int}} \qquad \frac{c \in \mathrm{Bool}}{c : \mathrm{Bool}} \qquad \frac{e_1 : T \quad e_2 : T \quad \cdots \quad e_k : T}{[e_1; e_2; e_3; \cdots; e_k] : \mathrm{List}[T]}$$

$$\boxed{\frac{e_1 : \mathrm{Int} \quad e_2 : \mathrm{Int}}{e_1 + e_2 : \mathrm{Int}}}$$

For all expressions e_1 & e_2
if e_1 is of type Int
& e_2 is of type Int
then $e_1 + e_2$ is of type Int

$$\frac{e_1 : \mathrm{Int} \quad e_2 : \mathrm{Int}}{e_1 - e_2 : \mathrm{Int}}$$

$$\frac{e_1 : \mathrm{Int} \quad e_2 : \mathrm{Int}}{e_1 \le e_2 : \mathrm{Bool}}$$

$$\frac{e_1 : \mathrm{Bool} \qquad e_2 : \mathrm{Bool}}{\begin{array}{ll} e_1 \text{ and } e_2 & : \mathrm{Bool} \\ e_1 \text{ or } e_2 & : \mathrm{Bool} \\ \text{not } e_1 & : \mathrm{Bool} \end{array}}$$

$[3;4;5][0] \Rightarrow^* 3$

$[3;4;5][\mathrm{true}] \Rightarrow^* ?$

$$\frac{e_1 : \mathrm{List}[T] \qquad e_2 : \mathrm{Int}}{e_1[e_2] : \underline{\quad T \quad}}$$

For all expressions e_1 & e_2, for all types T
if e_1 is of type List[T] & e_2 is of type Int
then $e_1[e_2]$ is of type T.

$$\frac{e_1 : \text{Bool} \qquad e_2 : T \qquad e_3 : T}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \underline{\ T\ }}$$

if true then 3
else 4

if 5 then 3
else 4

What would have happened if we admitted 3 + true ?

Evaluation would get "stuck".

runtime type error.

Claim: Well typed programs do not get "stuck".

3 + (if true then 5 else false)

Sibling: Ill typed programs $\frac{\text{always}}{\text{sometimes}}$ get stuck.

3+true

	Static	Dynamic
Strong	Ocaml Java, C	Python JavaScript
Weak	C	Perl, Bash

Types: 1890s/1900s.
Russell & Whitehead
Principia Mathematica.

Lisp: 50s

Fortran: 50s.

Types: 60s ?

Russell's paradox

The set of all sets which don't belong to themselves

If we can define the set A

If we can ~~define the set~~ A

$A = \{x \mid x \notin x\}$

then we can ask:

Does $A \in A$?

If $A \in A$ then $A \notin A$

On the other hand, if $A \notin A$, $A \in A$.

Language L_3 : Integers & functions

fun $x \to x+1$

$(\text{fun } x \to x+1)\ 3 \Rightarrow^* 4$.

Expressions $e ::= c \in \text{Int} \mid x \in \text{Var}$

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

$\mid \text{fun } (x : T) \to e_1$ "Abstraction"

Variable

$\mid e_1 \ e_2$ "Application"

Types, $T ::= \text{Int} \mid$ Functions $T_1 \to T_2$

Infinitely many types?

Int, Int $\to$ Int, Int $\to$ (Int $\to$ Int), Int $\to$ (Int $\to$ (Int $\to$ Int))

$Int, \; Int \to Int, \; Int \to (Int \to Int), \; Int \to (Int \to (Int \to Int))$
...

$$\frac{c \in Int}{Ctx \vdash c : Int} \qquad \frac{Ctx \vdash e_1 : Int \quad Ctx \vdash e_2 : Int}{Ctx \vdash e_1 \pm e_2 : Int} \qquad \frac{Ctx \vdash e_1 : T_1 \to T_2 \quad Ctx \vdash e_2 : T_1}{Ctx \vdash e_1 \; e_2 : T_2}$$

$$\cancel{\frac{}{x : Int}}$$

$$\frac{(x : T) \in Ctx}{Ctx \vdash x : T} \qquad \frac{Ctx, (x : T_1) \vdash e_1 : T_2}{Ctx \vdash fun \; (x : T_1) \cdot e_1 : T_1 \to T_2}$$

$Ctx \vdash x : Int \to Int$ $\qquad$ $Ctx \vdash y : Int$

$$(fun \; (x : Int \to Int) \to x \;\; 5) \; (fun \; (y : Int) \to y + 1) : Int$$

Takes a single Int as input & produces an Int as output
$(Int \to Int)$