Recitation 19: Type Checking

Type System

Decides whether

e
- Cannot be typed "ill typed"
- Can be typed "well typed"
  - If so, what is the type?

HasType(e, t, ctx)

ctx ⊢ e : t

ctx : Var → Types

e.g. initially ctx = {}

let f x = x + 1 in

ctx = {f : int → int}

f y ← using ctx, conclude y : int and expr has type int.

Lexer tokens

Parser exprs

Semantic Analysis { Type Checking

Evaluator values

true + 3 → stuck

env : Var → Val

Type System for SimPL

e ::= x | i | b | e_1 bop e_2 | if e_1 then e_2 else e_3 | let x = e_1 in e_2

bop ::= + | * | <

ctx : e ⊢ t

t ::= int | bool

Base Cases

$ctx \vdash i : int$
$ctx \vdash b : bool$
$\{x : t, \ldots\} \vdash x : t$

Inductive cases

Binop

if $ctx \vdash e_1 : int$
and $ctx \vdash e_2 : int$
$ctx \vdash e_1 + e_2 : int$
$ctx \vdash e_1 * e_2 : int$
$ctx \vdash e_1 \leq e_2 : bool$

Let

if $ctx \vdash e_1 : t_1$
and $ctx[e_1 \to t_1] \vdash e_2 : t$

$ctx \vdash let\ x = e_1\ in\ e_2 : t$

Conditionals

if $ctx \vdash e_1 : bool$
and $ctx \vdash e_2 : t$
and $ctx \vdash e_3 : t$
$ctx \vdash$ if e_1 then e_2 else $e_3 : t$

What's the point? Want to guarantee <u>type safety</u>

- progress
- preservation

Example:

```
e = let b = true in
      let x = if b then 3 else 4 in
        x + x
```