

Type Systems

Prof. Clarkson Fall 2019

Today's music: Check Yo Self by Ice Cube

CLICKER QUESTION 1

Review

Previously in 3110:

- Evaluation, i.e., formal dynamic semantics
- Small- and big-step relations
- Substitution and environment models

Today:

- Type systems, i.e., formal static semantics
- Type safety
- Type inference

Evaluation errors

```
5 + false #>
if 5 then true else 0 #>
```

Goal: prevent evaluation errors

- analyze program before running it
- reject program (and refuse to run) if possible evaluation errors detected

TYPE SYSTEMS

if expressions [from lec 2]

```
Syntax:
   if e1 then e2 else e3

Type checking (static semantics):
   if e1:bool and e2:t and e3:t
   then (if e1 then e2 else e3):t
```

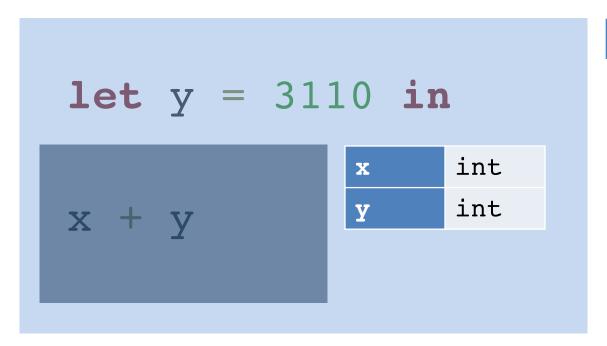
T + e : t

typing relation

the eval function we implemented

Static environment

$$let x = 42 in$$



x int

Environments

Static environment:

map from identifiers to types

x	int
Y	int

Dynamic environment:

map from identifiers to values

x	42
Y	3110

Typing relation examples

```
{x:int} ⊢ x + 2 : int
{x:bool} ⊬ x + 2 : int
{x:int} ⊬ x + 2 : bool
```

Typed SimPL

```
e ::= x | i | b
    | e1 bop e2
    | let x : t = e1 in e2
    l if e1 then e2 else e3
bop ::= + | * | <=
t ::= int | bool
```

Values and variables

```
env ⊢ i : int
```

env ⊢ b : bool

 $env \vdash x : env(x)$

Binary operators

```
env \vdash e1 + e2 : int
  if
  env ⊢ e1 : int
  env ⊢ e2 : int
env \vdash e1 * e2 : int
  if
  env ⊢ e1 : int
  env ⊢ e2 : int
env \vdash e1 \le e2 : bool
  if
  env ⊢ e1 : int
  env ⊢ e2 : int
```

If expressions

```
env \rightarrow if e1 then e2 else e3 : t

if
  env \rightarrow e1 : bool
  env \rightarrow e2 : t
  env \rightarrow e3 : t
```

Let expressions

```
env ⊢ let x = e1 in e2 : t2

if
  env ⊢ e1 : t1
  env[x → t1] ⊢ e2 : t2
```

TYPE SAFETY

Preventing evaluation errors

Evaluation of an expression **e** is **stuck** if:

- e is not a value, and
- e #

Purpose of type system:

guarantee no expression ever gets stuck

Type safety

Type safety means never getting stuck

Type safety = progress + preservation

- Progress: can always step (unless already value)
- Preservation: stepping never changes type

Preservation

```
if env \vdash e : t
and e \rightarrow e'
then env \vdash e' : t
```

$$\{\} \vdash (10 + 1) + (5 + 6) : int$$

 $(10 + 1) + (5 + 6) \rightarrow 11 + (5 + 6)$
 $\{\} \vdash 11 + (5 + 6) : int$

Progress

```
if \{\} \vdash e : t then e is a value or there exists an e' such that e \rightarrow e'
```

$$\{\} \vdash (10 + 1) + (5 + 6) : int$$

{} **⊬** x : int

Type safety proof sketch

Claim: Well-typed programs don't get stuck.

Proof: by induction on number of steps to reach a value.

Base case: value. Zero steps.

Already done, hence not stuck.

Inductive case: not a value.

- By progress: can take one step.
- By preservation: still well-typed.
- IH applies: one step taken.

QED.

TYPE INFERENCE

Typed SimPL, without annotations

```
e ::= x | i | b
    | e1 bop e2
    | let x = e1 in e2
    | if e1 then e2 else e3
bop ::= + | * | <=
t ::= int | bool
```

Guess and check

```
let x = e1 in e2
```

- Infer type t1 of e1
- Put {x:t1} in static environment
- Use that to type check e2

OCaml type inference

Based on Hindley-Milner algorithm

- Never infers the wrong types
- Never fails to infer types
- Usually runs in linear time

(for the curious: see textbook 10.5, but we aren't covering it this semester)

Robin Milner



1934-2010

Awarded 1991 Turing Award for "...ML, the first language to include polymorphic type inference and a type-safe exception handling mechanism..."

Upcoming events

- [today] A6 released
- [tonight] MS1 due: no late submissions

This is true to type.

THIS IS 3110