CS 3110

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` $\not\Rightarrow$

`if 5 then true else 0` $\not\Rightarrow$

Goal: prevent evaluation errors

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

Demo

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`**

$$\mathtt{T} \vdash \mathtt{e} : \mathtt{t}$$

typing relation

the `eval` function we implemented

Static environment

```
let x = 42 in
  let y = 3110 in
    x + y
```

x	int

x	int
y	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}` $\vdash$ `x + 2 : int`

`{x:bool}` $\nvdash$ `x + 2 : int`

`{x:int}` $\nvdash$ `x + 2 : bool`

Typed SimPL

```
e ::= x | i | b
    | e1 bop e2
    | let x : t = e1 in e2
    | if e1 then e2 else e3

bop ::= + | * | <=

t ::= int | bool
```

Values and variables

$$env \vdash i : int$$

$$env \vdash b : bool$$

$$env \vdash x : env(x)$$

Binary operators

```
env ⊢ e1 + e2 : int
```
if
```
env ⊢ e1 : int
env ⊢ e2 : int
```

```
env ⊢ e1 * e2 : int
```
if
```
env ⊢ e1 : int
env ⊢ e2 : int
```

```
env ⊢ e1 <= e2 : bool
```
if
```
env ⊢ e1 : int
env ⊢ e2 : int
```

If expressions

`env` ⊢ `if e1 then e2 else e3 : t`

if

`env` ⊢ `e1 : bool`

`env` ⊢ `e2 : t`

`env` ⊢ `e3 : t`

Let expressions

$\texttt{env} \vdash \texttt{let x = e1 in e2 : t2}$

if

$\texttt{env} \vdash \texttt{e1 : t1}$

$\texttt{env[x} \mapsto \texttt{t1]} \vdash \texttt{e2 : t2}$

Demo

TYPE SAFETY

Preventing evaluation errors

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

- **e** is not a value, and
- **e** $\not\rightarrow$

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 $\texttt{env} \vdash \texttt{e} : \texttt{t}$

and $\texttt{e} \longrightarrow \texttt{e}'$

then $\texttt{env} \vdash \texttt{e}' : \texttt{t}$

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

Progress

if $\texttt{\{\}} \vdash \texttt{e : t}$

then $\texttt{e}$ is a value or

there exists an $\texttt{e}'$ such that $\texttt{e} \longrightarrow \texttt{e}'$

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

$$\{\} \nvdash x : \mathrm{int}$$

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`**

Demo

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