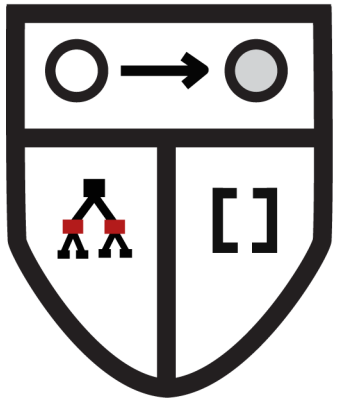

CS 3110

Type Checking

Nate Foster

Spring 2019

Today's music: *Check Yo Self* by Ice Cube

Review

Previously in 3110: formal semantics

- Dynamic semantics: small-step relation
- Substitution operation

Today:

- Formal static semantics

REVIEW: SUBSTITUTION MODEL

Demo

FORMAL STATIC SEMANTICS

Question

What do we get when we evaluate the following?

```
step Add(Bool false, Int 42)
```

A. `Int 42`

B. `Int 43`

C. `Add(Bool 42, Int 42)`

D. It goes into an infinite loop

E. None of the above

Static semantics

We can have nonsensical expressions:

```
5 + false
```

```
if 5 then true else 0
```

Need to rule those out...

`if` expressions [from lec 2]

Syntax:

```
if e1 then e2 else e3
```

Type checking:

if **`e1`** has type **`bool`** and **`e2`** has type **`t`** and **`e3`** has type **`t`**
then **`if e1 then e2 else e3`** has type **`t`**

Static semantics

Defined as a ternary relation:

```
T |- e : t
```

- Read as in typing context `T`, expression `e` has type `t`
- Turnstile `|-` can be read as "proves" or "shows"
- You're already used to `e : t`, because utop uses that notation
- *Typing context* is a dictionary mapping variable names to types

Types

```
type typ =
   | Tint
   | TBool
```

Static Semantics: Constants

```
T |- i : Int
```

```
T |- b : Bool
```

Static Semantics: Integers

```
T |- i : Int
```

Static Semantics: Variables

```
T |- x : t
```

if `T(x) = t`

Static Semantics: Conditionals

```
  T |- if e1 then e2 else e3 : t
  if T |- e1 : bool
and T |- e2 : t
and T |- e3: t
```

Static Semantics: Let Expressions

```
   T |- let x = e1 in e2 : t
   if T |- e1 : t1
and T,x:t1 |- e2 : t
```

Static semantics

e.g.,

```
x:int |- x + 2 : int
x:int,y:int |- x < y : bool
|- 5 + 2 : int
```

Demo

Static semantics

e.g.,

```
x:int |- x + 2 : int
x:int,y:int |- x < y : bool
|- 5 + 2 : int
```

Demo

Purpose of type system

Ensure **type safety:** well-typed programs don't get *stuck:*

- haven't reached a value, and
- unable to evaluate further

Lemmas:

Progress: if **`e : t`**, then either **`e`** is a value or **`e`** can take a step.

Preservation: if **`e : t`**, and if **`e`** takes a step to **`e'`**, then **`e' : t`**.

Type safety = progress + preservation

Proving type safety is a fun part of CS 4110

Upcoming events

- [Wednesday/Thursday] Beta demos
- [Thursday] Guest Lecture by Yaron Minsky

This is not a substitute.

THIS IS 3110