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
FORMAL STATIC SEMANTICS
Question

What do we get when we evaluate the following?

```
step Add(Boolean false, Integer 42)
```

A. Integer 42  
B. Integer 43  
C. Add(Boolean 42, Integer 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:
\[
\text{if } e_1 \text{ then } e_2 \text{ else } e_3
\]

Type checking:
\[
\text{if } e_1 \text{ has type } \text{bool} \text{ and } e_2 \text{ has type } t \text{ and } e_3 \text{ has type } t
\text{ then } \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \text{ has type } t
\]
Static semantics

Defined as a ternary relation:

\[ \vdash e : t \]

- Read as in typing context \( \vdash \), expression \( e \) has type \( t \)
- Turnstile \( \vdash \) 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

\[ T \vdash i : \text{Int} \]

\[ T \vdash b : \text{Bool} \]
Static Semantics: Integers

\[ T |- i : \text{Int} \]
Static Semantics: Variables

\[
T \vdash x : t \\
\text{if } T(x) = t
\]
Static Semantics: Conditionals

\[ T \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : t \]
\[ \text{if } T \vdash e_1 : \text{bool} \]
and \[ T \vdash e_2 : t \]
and \[ T \vdash e_3 : t \]
Static Semantics: Let Expressions

\[
T \mid- \text{let } x = e_1 \text{ in } e_2 : t
\]

if \( T \mid- e_1 : t_1 \)

and \( T, x : t_1 \mid- e_2 : t \)
Static semantics

e.g.,
\[ x : \text{int} \vdash x + 2 : \text{int} \]
\[ x : \text{int}, y : \text{int} \vdash x < y : \text{bool} \]
\[ \vdash 5 + 2 : \text{int} \]
Static semantics

e.g.,

\[ x : \text{int} \vdash x + 2 : \text{int} \]
\[ x : \text{int}, y : \text{int} \vdash x < y : \text{bool} \]
\[ \vdash 5 + 2 : \text{int} \]
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