Program Testing and Analysis: Operational Semantics

Dr. Michael Pradel

Software Lab, TU Darmstadt

Warm-up Quiz

What does the following code print?

```
var e = eval;

(function f() {
  var x = 5;
  e("x=7")
  console.log(x);
})();
```

Options: 5, 7, or something else

Warm-up Quiz

```
var e = eval;

(function f() {
  var x = 5;
  e("x=7")
  console.log(x);
})();
```

Correct answer: 5

Warm-up Quiz

```
var e = eval;

(function f() {
  var x = 5;
  e("x=7")
  console.log(x);
})();
```

`eval()` evaluates JavaScript code given as a string

Correct answer: 5

Warm-up Quiz

Store function into variable (functions are first-class objects)

```
var e = eval;

(function f() {
  var x = 5;
  e("x=7")
  console.log(x);
})();
```

Correct answer: 5

Warm-up Quiz

```
var e = eval;

(function f() {
  var x = 5;
  e("x=7")
  console.log(x);
})();
```

Define a function and call it immediately

Correct answer: 5

Warm-up Quiz

```
var e = eval;

(function f() {
  var x = 5;
  e("x=7")
  console.log(x);
})();
```

Indirect `eval()`: Works at global scope rather than local scope

Correct answer: 5

Big Picture

Last lecture:

- Syntax of languages
- Representations of programs

This lecture:

- Assign meaning (= semantics) to programs
- Focus: Operational semantics of imperative languages
- Formal foundation for specifying languages and for describing dynamic analyses

Plan for Today

- **Motivation & preliminaries**
- **Abstract syntax of SIMP**
- **An abstract machine for SIMP**
- **Structural operation semantics for SIMP**
  - Small-step semantics
  - Big-step semantics

Why Do We Need Operational Semantics?

Example (C code):

```
int i = 5;
f(i++, --i)
```

← What are the actual parameters?

Option 1: 5, 5 (left-to-right)

Option 2: 4, 4 (right-to-left)

Both options possible in C

→ unspecified semantics!

→ compiler decides

Specifying the Semantics of Programs

- Static semantics, e.g., types
- Dynamic semantics
  - Denotational
  - Axiomatic
  - Operational (our focus)

↳ Useful for:
  - Lang. implementation
  - Programming
  - Lang. design
  - Program analysis

1) Preliminaries

a) Transition system

- set Config of configurations or states
- binary relation $\rightarrow \subseteq \text{Config} \times \text{Config}$ ("transition relation")
  - ↳ $c \rightarrow c'$ .. change of state

$\rightarrow^*$ .. reflexive-transitive closure of $\rightarrow$

$\forall c.\ c \rightarrow^* c$

$\forall c, c', c''.\ c \rightarrow^* c' \wedge c' \rightarrow^* c'' \Rightarrow c \rightarrow^* c''$

deterministic: $c \rightarrow c_1 \wedge c \rightarrow c_2 \Rightarrow c_1 = c_2$

b) Rule induction

↳ define a set with:

- a finite set of basic elements (axioms)

$$\frac{}{t}$$

- a finite of rules that specify how to generate elements

$$\frac{h_1 \quad \dots \quad h_n}{c}$$

← hypotheses

← conclusion

Example 1: set of natural numbers

axiom: $\frac{}{0}$ rule: $\frac{n}{n+1}$

<u>Example 2:</u> Evaluation of expressions

↳ Set = pairs of AST & value

Notation: $E \Downarrow n$ .. expr. E evaluates to number n

Axioms: $\underbrace{1 \Downarrow 1,\ 2 \Downarrow 2,\ \text{etc.}}_{\text{axiom scheme}}$ $\quad n \Downarrow n$

same $\quad \dfrac{}{n \Downarrow n}$

Rules: $\underbrace{\dfrac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{+(E_1, E_2) \Downarrow n}}_{\text{rule scheme:}}$ if $n = (n_1 + n_2)$, etc.

$$\dfrac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{O_p(E_1, E_2) \Downarrow n} \quad \text{if } n = (n_1\ O_p\ n_2)$$

— — —

c) Proof tree

↳ show that element is in inductive set

Example 1

$$\dfrac{\dfrac{0}{1}}{2}$$

Example 2 Show that $-(+(3,4),1) \Downarrow 6$

axioms, # rules ?

3 2

$$\dfrac{\dfrac{\dfrac{\dfrac{}{3 \Downarrow 3} \quad \dfrac{}{4 \Downarrow 4}}{+(3,4) \Downarrow 7}\,(\text{plus}) \quad \dfrac{}{1 \Downarrow 1}}{-(+(3,4),1) \Downarrow 6}\,(\text{minus})$$

Plan for Today

- **Motivation & preliminaries**
- **Abstract syntax of SIMP** ←
- **An abstract machine for SIMP**
- **Structural operation semantics for SIMP**
  - Small-step semantics
  - Big-step semantics

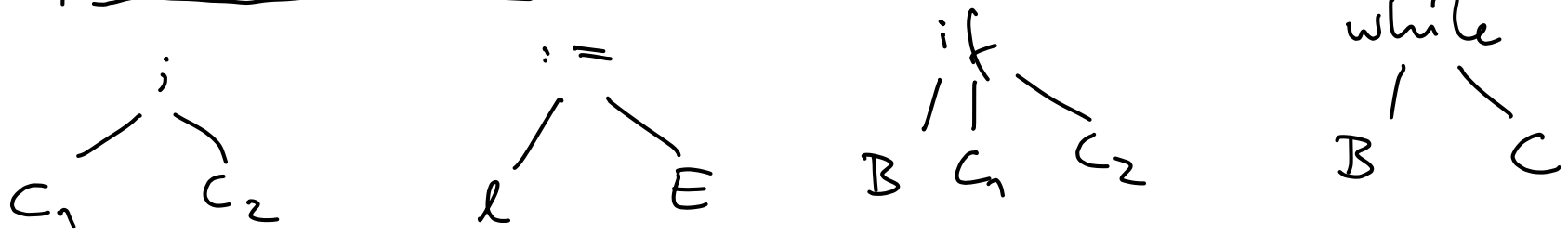
Abstract syntax of SIMP = simple imperative PL

- features: assignments, sequencing, conditionals, loops, integer variables

- abstract syntax:

$$P ::= C \mid E \mid B$$

a) <u>Commands:</u>



(sequence) (assignment to label l) (two-way selector) (while loop) skip (do nothing)

textual notation:

$$C ::= l := E \mid C ; C \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \mid$$
$$\text{while } B \text{ do } C \mid \text{skip}$$

b) <u>Integer expressions:</u>

$E ::= \; !l \mid n \mid E \; op \; E$

$op ::= \; + \mid - \mid * \mid /$

where n .. integers

$l \in L = \{ l_0, l_1, \dots \}$ .. memory locations

$!l$ .. value stored at address l

c) <u>Boolean expressions:</u>

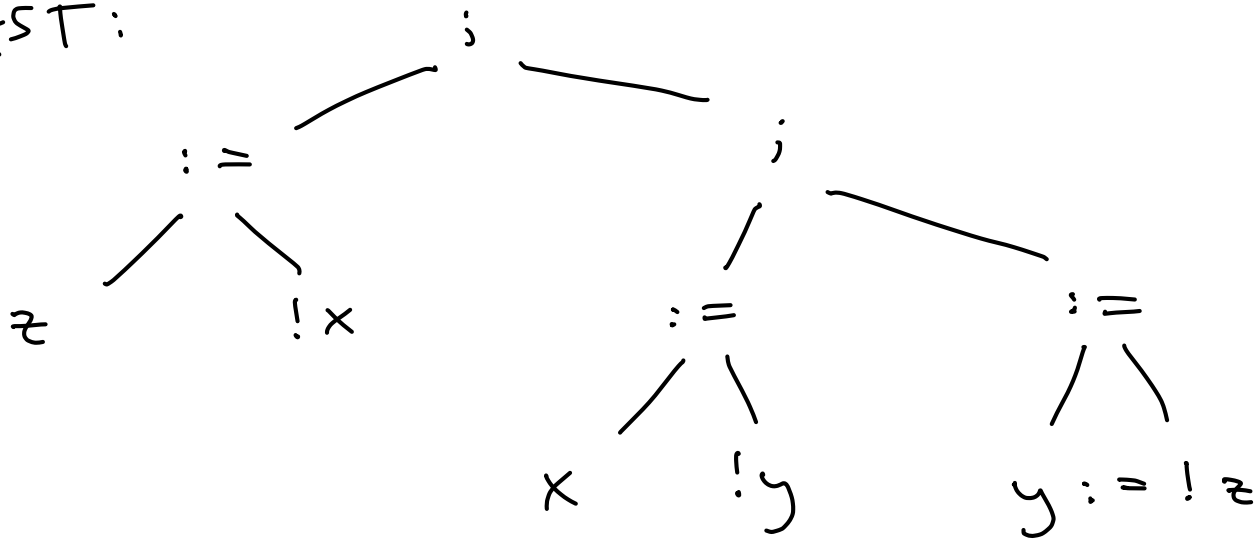
$B ::= \; True \mid False \mid E \; bop \; E \mid \neg B \mid B \wedge B$

$bop ::= \; < \mid > \mid =$

<u>Example 1:</u> $z := !x \; ; \; (x := !y \; ; \; y := !z)$

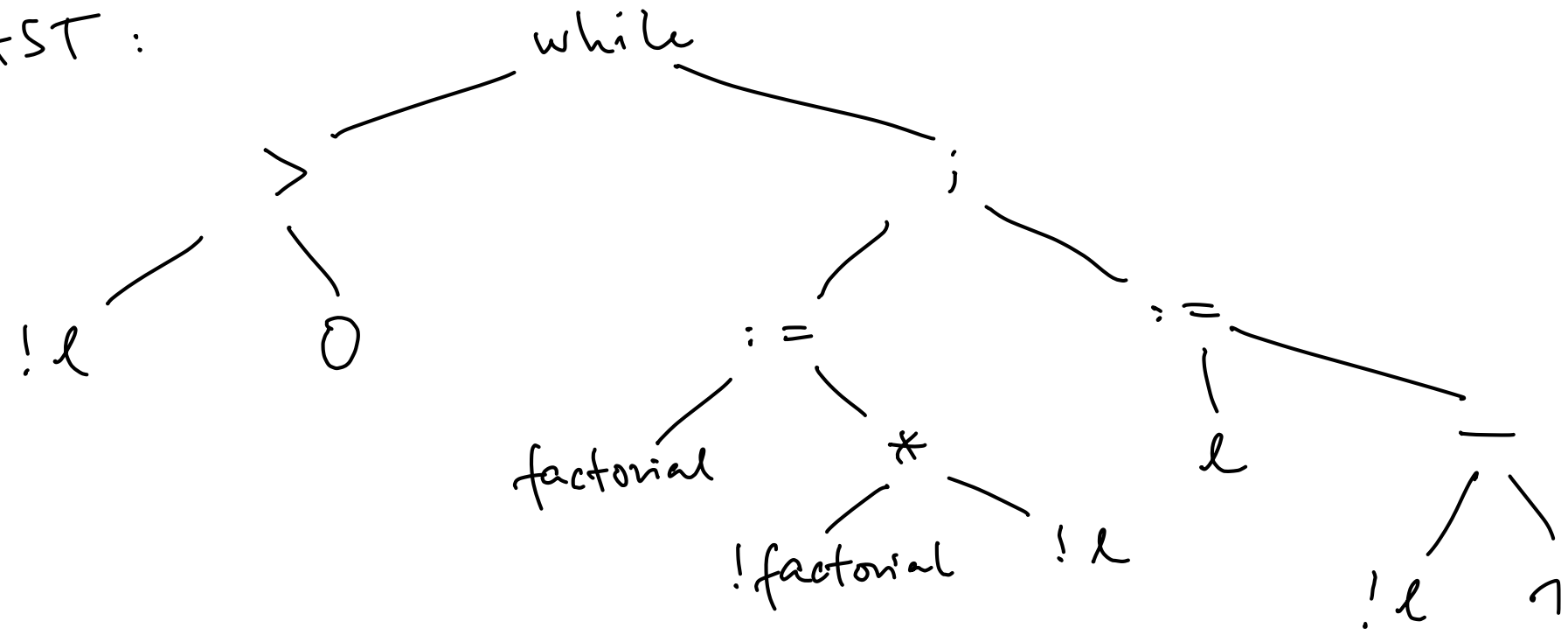
... swaps values of x and y

AST:



<u>Example 2:</u> while (!l > 0) do
(factorial := !factorial * !l ;
l := !l - 1)

AST :



Plan for Today

- **Motivation & preliminaries**
- **Abstract syntax of SIMP**
- **An abstract machine for SIMP** ←
- **Structural operation semantics for SIMP**
  - Small-step semantics
  - Big-step semantics

An Abstract Machine for SIMP

4 main elements:

- control stack s : stores instructions to execute
- auxiliary / result stack r : stores intermediate values
- processor : performs arithm. operations, etc.
- memory / store m : partial function mapping locations to integers

↳ Notation : $m[l \mapsto n]$ .. update m with the new mapping

↳ i.e. $m[l \mapsto n](l) = n$

$m[l \mapsto n](l') = m(l') \quad \forall l'.\ l' \neq l$

Abstract machine = transition system

- Configuration : $\langle c, r, m \rangle$

$c ::= nil \mid i \circ c$ instruction i pushed on top of c

↳ empty stack

$i ::= P \mid op \mid \neg \mid \wedge \mid bop \mid := \mid if \mid while$

$r ::= nil \mid P \circ r \mid \ell \circ r$

Model execution of SIMP programs as sequences of transitions from initial to final configuration

$\langle C \circ nil, nil, m \rangle$

Execute C in given memory state m

$\langle nil, nil, m \rangle$

Stops when stacks are empty

Transition rules: $\longrightarrow$

$$\langle c, r, m \rangle \longrightarrow \langle c', r', m' \rangle$$

a) Evaluating expressions

$$\langle n \circ c,\ r,\ m \rangle \longrightarrow \langle c,\ n \circ r,\ m \rangle$$

pop n from control stack, push onto result stack

$$\langle !\ell \circ c,\ r,\ m \rangle \longrightarrow \langle c,\ n \circ r,\ m \rangle \quad \text{if } m(\ell) = n$$

read from memory & push onto r

$$\langle (E_1\ op\ E_2) \circ c,\ r,\ m \rangle \longrightarrow \langle E_1 \circ E_2 \circ op \circ c,\ r,\ m \rangle$$

$$\langle op \circ c,\ n_2 \circ n_1 \circ r,\ m \rangle \longrightarrow \langle c,\ n \circ r,\ m \rangle \quad \text{if } n_1\ op\ n_2 = n$$

(similar rules for Boolean expressions)

Semantics of SIMP expressions

Value of E in state m is v
if there is a sequence of transitions

$$\langle E \circ c, r, m \rangle \rightarrow^* \langle c, v \circ r, m' \rangle$$

<u>Example:</u> What is the value of $c = a + b$
in state where $m = \{a \mapsto 3, b \mapsto 1\}$?

transitions ?