Program Analysis

Operational Semantics (Part 1)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2023/2024

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 in global scope rather than local scope

Correct answer: 5

Big Picture

Last lecture:

- Syntax of languages
- Representations of programs

Next 2–3 lectures:

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

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

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

Both options are legal in C!

→ Unspecified semantics

→ Compiler decides

Want: (Almost) all behavior is clearly specified

How to specify the semantics of a PL?

- Static types
- Dynamic semantics
    * Denotational
    * Axiomatic
    * Operational ⟵ Focus in this course

Preliminaries

a) Transition systems

- set Config of configurations or states
- binary relation $\rightarrow \subseteq Config \times Config$

  ("transition relation")

  $c \rightarrow c'$ ... transition (or change of state)

  ↳ ≈ step of computation

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

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

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

b) Rule induction

↳ define a set ("inductive set") with

* a finite set of basic elements ("axioms"): $\dfrac{}{t}$

* a finite set of rules that specify how to generate more elements: $\dfrac{h_1 \dots h_n}{c}$ (hypothesis: $h_1 \dots h_n$; conclusion: c)

↳ Ex. 1: set of natural numbers

axiom: $\dfrac{}{0}$

rule: $\dfrac{n}{n+1}$

Ex. 2: Evaluating arithmetic expressions, e.g., $+(3,4)$

↳ Set of pairs of AST and value

Notation: $E \Downarrow n$ "Expr. E evaluates to value n"

Axioms: $1 \Downarrow 1$, $2 \Downarrow 2$, etc. } axiom scheme: $n \Downarrow n$

Rules:

$$\frac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{+(E_1, E_2) \Downarrow n} \text{ if } n_1 + n_2 = n$$

rule scheme:

$$\frac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{Op(E_1, E_2) \Downarrow n} \text{ if } n_1 \; Op \; n_2 = n$$

c) Proof tree

↳ show that element is in an inductive set

Ex. 1

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

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

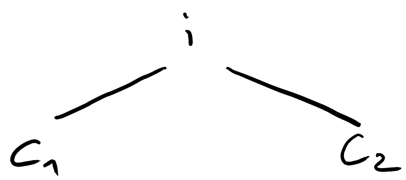
$$\dfrac{\dfrac{\dfrac{}{3 \Downarrow 3} \qquad \dfrac{}{4 \Downarrow 4}}{+(3,4) \Downarrow 7}\ \text{if...} \qquad \dfrac{}{1 \Downarrow 1}}{-(+(3,4),\ 1) \Downarrow 6}\ \text{if ...}$$

Abstract syntax of SIMP

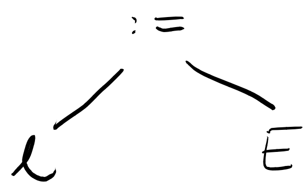
↳ simple, imperative lang.

- features: assignment, sequencing, conditional, loops, integer variables
- $P ::= C \mid E \mid B$

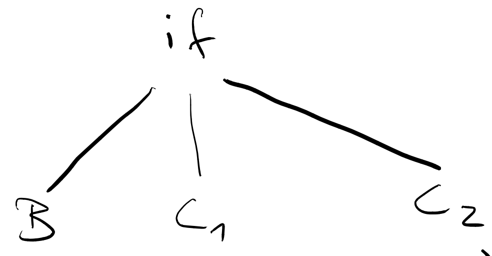
a) Commands



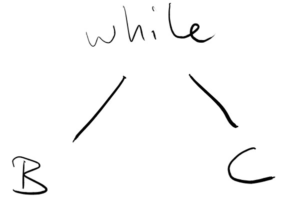
(sequence)



(assignment of expr. E to label ℓ)



(two-way selector)



(while loop)

skip

(do nothing)

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

b) Integer expressions

$E ::= \,!\ell \mid n \mid E \text{ op } E$

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

where n ... integer

$\ell \in L = \{\ell_0, \ell_1, \dots\}$ ... memory locations

$!\ell$ ... value stored at location ℓ

c) Boolean expressions

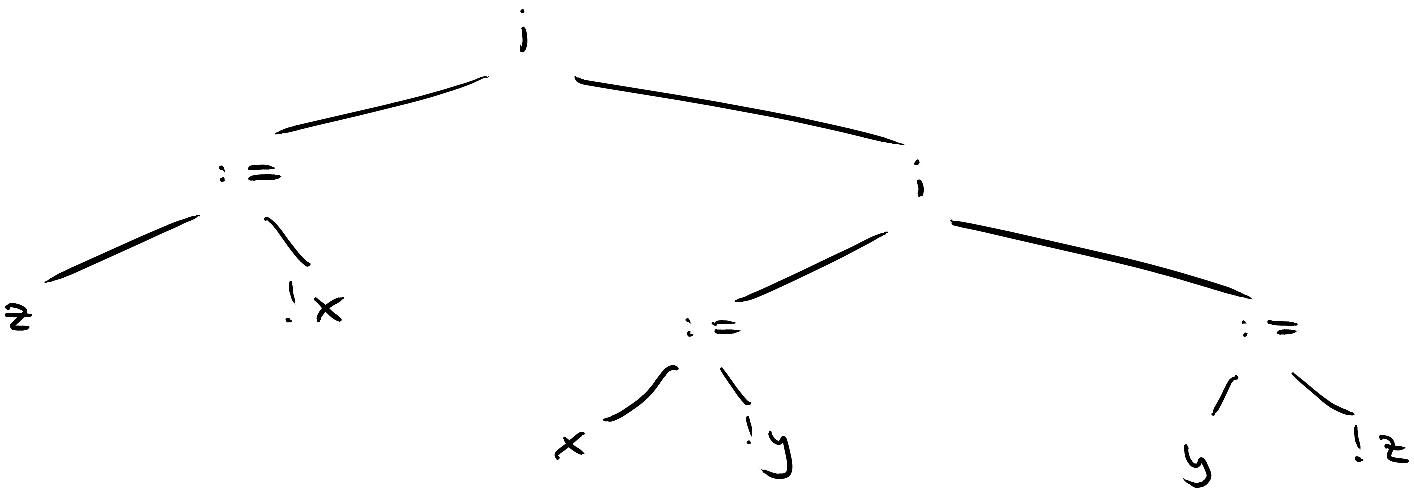
$B ::= \text{True} \mid \text{False} \mid E \text{ bop } E \mid \neg B \mid B \wedge B$

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

Ex. 1 $z := !x \; ; \; (x := !y \; ; \; y := !z)$

... swap values in x and y

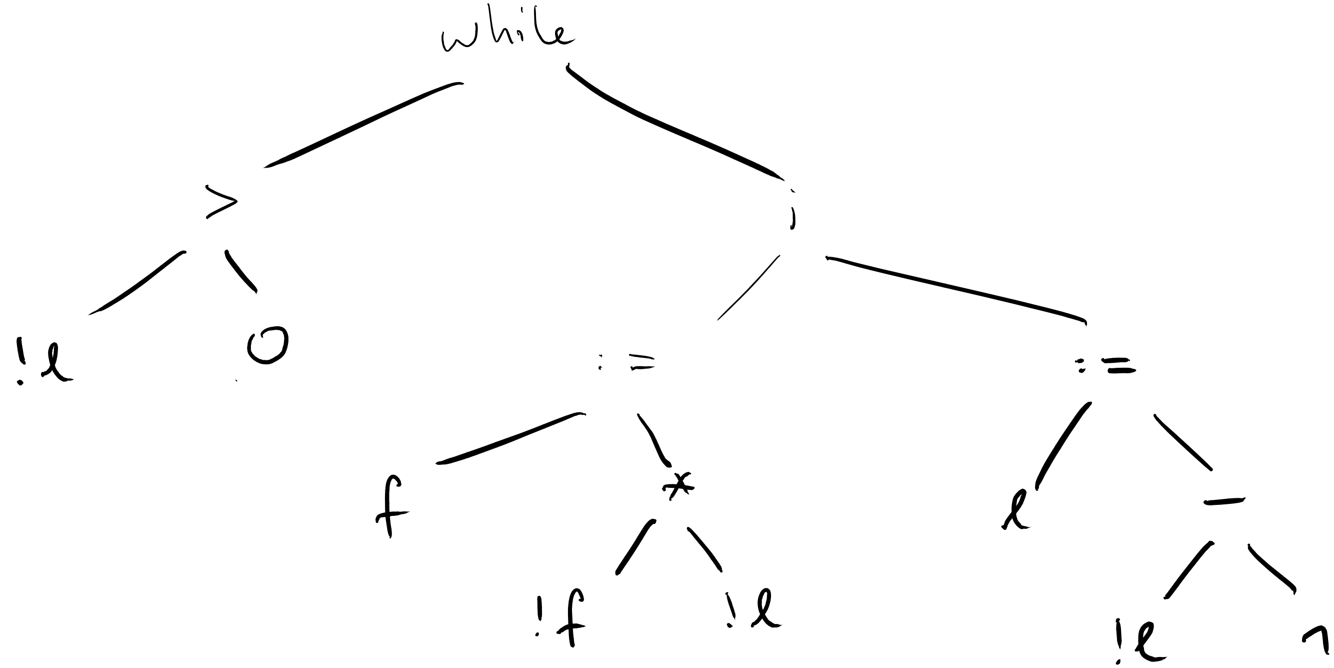
AST:



Ex. 2

```
while ( !l > 0) do (
    f := !f * !l ;
    l := !l - 1
)
```

AST:



Abstract machine for SIMP

4 elements:

- control stack c : store instructions to execute
- auxiliary / result stack r : stores intermediate results
- processor : perform arithmetic operations, comparisons, and boolean operations
- memory / state m : partial function mapping locations to integers

↳ Notation: $m[\ell \mapsto n]$ ... updating fct. m with new mapping $\ell \mapsto n$

↳ $m[\ell \mapsto n](\ell) = n$

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

Abstract machine = transition system

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

$c ::= nil \mid i \circ c$

empty stack — instruction i pushed on top of control stack c

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

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

Model execution of programs as sequences of transitions from initial state to final state

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

Execute C in a given memory state

Stop when all stacks empty

Transition rules : $\longrightarrow$

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

a) Evaluating expressions

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

↳ pop n from c and push it on r

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

↳ read memory at ℓ and push on r

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

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

(similar for Boolean expressions)