Program Testing and Analysis: Operational Semantics

Dr. Michael Pradel

Software Lab, TU Darmstadt
Warm-up Quiz

What does the following code print?

```javascript
var e = eval;

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

Options: 5, 7, or something else
var e = eval;

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

Correct answer: 5
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)

```javascript
var e = eval;

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

Correct answer: 5
Warm-up Quiz

```javascript
var e = eval;

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

Correct answer: 5

Define a function and call it immediately
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
Follow Up on Last Week

(Hand-written notes)
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 Semantic?**

Example (C code):

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

← What are the actual parameters passed to f?

Option 1: 5, 5 (left-to-right)
Option 2: 4, 4 (right-to-left)

Both options are possible in C

→ Unspecified semantics

→ Compiler decides

Want: (Almost) all behavior should be clearly specified
Specifying the Semantics of Programs

- Static semantics: E.g., types
- Dynamic semantics:
  - Denotational
  - Axiomatic
  - Operational (focus)

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

2) Transition system
- set Config of configurations or states
- binary relation \( \rightarrow \subseteq \text{Config} \times \text{Config} \)
  (**transition relation**)

\( \text{c} \rightarrow \text{c'} \) transition, or change of state

\( \Rightarrow \) step of computation

\( \rightarrow^* \) reflexive-transitive closure of \( \rightarrow \)

\( \forall \text{c}. \text{c} \rightarrow^* \text{c} \)

\( \forall \text{c}, \text{c'}, \text{c''}. \text{c} \rightarrow^* \text{c'} \land \text{c'} \rightarrow^* \text{c''} \Rightarrow \text{c} \rightarrow^* \text{c''} \)

deterministic: \( \text{c} \rightarrow \text{c}_1 \land \text{c} \rightarrow \text{c}_2 \Rightarrow \text{c}_1 = \text{c}_2 \)
6) **Rule induction**

- Define a set ("inductive set") with:
  - A finite set of basic elements (axioms): \( \text{Ax} \)
  - A finite set of rules that specify how to generate elements of the set

\[
\begin{align*}
  h_0 & \quad h_n \\
  \hline
  c
\end{align*}
\]

- Hypotheses (or premises)

- Conclusion

6) **Example**

- Set of natural numbers

  - **Axiom:** \( 0 \)
  - **Rule:** \( \frac{n}{n+1} \)
c) **Proof tree**

L> show that an element is in an inductive set

**Example 1:**

\[ \begin{array}{c}
0 \\
1 \\
\hline
2
\end{array} \]

\( \rightarrow 2 \text{ is a natural number} \)

**Example 2:** show that

\[ - (+ (3, 4), 1) \Downarrow 6 \]

**Quiz:**

- # axioms? 3
- # rules? 2

\[ \begin{array}{c}
3 \Downarrow 3 \\
4 \Downarrow 4 \\
1 \Downarrow 1 \\
\hline
+ (3, 4) \Downarrow 7 \\
\hline
- (+ (3, 4), 1) \Downarrow 6 \end{array} \]
Example 2: Evaluation of expressions, e.g. + (3, 4)

Set = pair of AST & value

Notation: \( E \downarrow n \) - expr. \( E \) evaluates to number \( n \)

Axioms:

\( 1 \downarrow 1 \), \( 2 \downarrow 2 \), etc.

Axiom scheme: \( n \downarrow n \).

Rules:

\[
\begin{align*}
E_1 \downarrow n_1 & \quad E_2 \downarrow n_2 \\
+ (E_1, E_2) & \downarrow n
\end{align*}
\]

if \( n = n_1 + n_2 \), etc.

Rule scheme:

\[
\begin{align*}
E_1 \downarrow n_1 & \quad E_2 \downarrow n_2 \\
\text{Op} (E_1, E_2) & \downarrow n
\end{align*}
\]

if \( n = n_1 \text{ Op } n_2 \)
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
Operational Semantics of an Imperative Language

1) Abstract syntax of SIMP
   - SIMP = simple imperative PL
   - features: assignments, sequencing, conditionals, loops, integer variables

   - abstract syntax:
     \[ P ::= C \mid E \mid B \]
a) Commands:

- `i := E` (assignment)
- `if B then C_1 else C_2` (two-way selector)
- `while B do C` (while loop)

Textual notation:

\[
C ::= C_1; C_2 | \ell := E | \text{if } B \text{ then } C_1 \text{ else } C_2 | \text{while } B \text{ do } C | \text{skip}
\]
b) Integer expressions

\[ E ::= \text{!} l \mid n \mid E \text{ op } E \]
\[ \text{op ::= } + \mid - \mid * \mid / \]

where
\[ n \ldots \text{ integer} \]
\[ l \in L = \{ l_0, l_1, \ldots \} \ldots \text{ memory locations} \]
\[ \text{!} l \ldots \text{ value stored at location } l \]

---

c) Boolean expressions

\[ B ::= \text{True} \mid \text{False} \mid E \text{ bop } E \mid -B \mid B \land B \]
\[ \text{bop ::= } \gt \mid \lt \mid = \mid \]
Example 1:  
\[ z := !x ; ( x := !y ; y := !z ) \]  
... swap values in x and y

AST:
Example 2

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

AST:

Quiz: # nodes: 15 # edges: 14
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
2) An Abstract Machine for SIMP

4 main elements:
- control stack c: stores instructions to execute
- auxiliary/results stack r: stores intermediate results
- processor: performs arithmetic ops, comparisons, and boolean ops.
- memory/store m: partial function mapping locations to integers

Notation: \( m[l \mapsto n] \) - update fact. \( m \) with new mapping \( l \mapsto n \).

i.e.
\[
\begin{align*}
    m[l \mapsto n](l) &= n \\
    m[l \mapsto n](l') &= m(l') \quad \forall l', l' \neq l
\end{align*}
\]
Abstract machine - transition system

Configuration: \( <c, r, m> \)

- \( c ::= \text{nil} \mid \text{op} \cdot c \)  — instr. i pushed on top of c
- \( r ::= \text{nil} \mid \text{P} \mid \text{Por} \mid \text{lor} \)

...
Model execution of programs as sequences of transitions from initial to final configurations.

\( \langle C, \text{nil}, m \rangle \) \quad \langle \text{nil}, \text{nil}, m \rangle \\

Execute \( C \) in a given memory state.

Stop when all stacks are empty.

Transition rules:

\( \langle C, r, m \rangle \rightarrow \langle C', r', m' \rangle \)
a) Evaluating expressions

\[ \langle n \circ c, r, m \rangle \rightarrow \langle c, n \circ r, m \rangle \]
- Pop \( n \) from control stack & push it onto \( r \)

\[ \langle !l \circ c, r, m \rangle \rightarrow \langle c, n \circ r, m \rangle \quad \text{if} \ m(l) = n \]
- Read memory at \( l \) & push result on \( r \)

\[ \langle (E_1 \circ E_2) \circ c, r, m \rangle \rightarrow \langle E_1 \circ E_2 \circ c, r, m \rangle \]

\[ \langle c \circ c, n_2 \circ n, r, m \rangle \rightarrow \langle c, n \circ r, m \rangle \quad \text{if} \ m(c) \circ n_2 = n \]

(similar for Boolean expr.)