Program Testing and Analysis:
Operational Semantics

Prof. 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
Warm-up Quiz

\[
\text{var } e = \text{eval};
\]

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

Correct answer: 5
Warm-up Quiz

```javascript
var e = eval;

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

Correct answer: 5
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);
})();
```

Define a function and call it immediately

Correct answer: 5
Warm-up Quiz

```javascript
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

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):

```c
int i = 5;
f(i++, --i) /* What are the actual arguments 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 \(\rightarrow\) Focus of lecture

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

a) Transition system
   - set Config of configurations or states
   - binary relation $\rightarrow \subseteq \text{Config} \times \text{Config}$
     ("transition relation")
   $c \rightarrow c'$ .. transition (or change of state)
     $\Rightarrow$ step of computation
     $\rightarrow^*$ .. reflexive, transitive closure of $\rightarrow$

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

deterministic : $c \rightarrow c_1 \land c \rightarrow c_2 \Rightarrow c_1 = c_2$
b) **Rule induction**

1. define a set ("inductive set") with:
   - a finite set of basic elements ("axioms"): \( t \)
   - a finite set of rules that specify how to generate elements of the set: \( b_n \rightarrow b_n \) \( \text{hypotheses} \)

2. **Example 1** set of natural numbers

   - **axiom:** 0
   - **rule:** \( n \rightarrow n + 1 \) \( \text{conclusion} \)
Example 2: Evaluation of expressions, e.g., $(3, 4)$

4) Set = pairs of AST & value

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

Axioms: $1 \Downarrow 1, 2 \Downarrow 2$, etc.

 axiom scheme: $n \Downarrow n$

Rules:

\[
\frac{E_1 \Downarrow n_1 \quad E_2 \Downarrow n_2}{(E_1, E_2) \Downarrow n} \quad \text{if } n = n_1 + n_2 \quad \text{etc.}
\]

rule scheme: $\frac{E_1 \Downarrow n_1 \quad E_2 \Downarrow n_2}{Op\ (E_1, E_2) \Downarrow n} \quad \text{if } n = n_1\ Op\ n_2$
c) Proof tree

4) show that an element is in an inductive set

Example 1

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

\[ \boxed{2} \]

\[ \rightarrow 2 \text{ is a natural number} \]

Example 2 show that

\[ (+(3,4), 7) \Downarrow 6 \]

\[ \text{QUIT: } \# \text{ axioms? } 3 \]

\[ \# \text{ rules? } 2 \]

\[ \begin{array}{c}
3 \Downarrow 3 \\
4 \Downarrow 4 \\
+(3,4) \Downarrow 7 \\
1 \Downarrow 1
\end{array} \]

\[ (+(3,4), 7) \Downarrow 6 \]
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: assignment, sequencing, conditionals, loops, integer variables
- abstract syntax:
  \[ P ::= C | E | B \]
a) Commands

\[
\begin{array}{c}
\text{\textbf{1}} \\
C_1 \quad C_2 \\
\text{(sequence)}
\end{array}
\]

\[
\begin{array}{c}
\text{i} \\
\text{=} \\
E
\end{array}
\]

(assignment of expression to label)

\[
\begin{array}{c}
\text{if} \\
B \quad C_1 \quad C_2
\end{array}
\]

(two-way selector)

\[
\begin{array}{c}
\text{while} \\
\text{skip}
\end{array}
\]

(do nothing)

(while loop)

\[
\begin{array}{c}
C := C; C \mid E := E \mid \text{if } B \text{ then } C_1 \text{ else } C_2
\end{array}
\]

\[
\begin{array}{c}
\text{while } B \text{ do } C \mid \text{skip}
\end{array}
\]

Textual notation:
b) Integer expressions

\[ E ::= !l \mid n \mid E \text{ op } E \]

\[ \text{op ::= } + \mid - \mid * \mid / \]

where

\[ n \text{ .. integer} \]

\[ l \in L = \{ \text{lo}, \text{lc}, \ldots \} \text{ .. memory location} \]

\[ !l \text{ .. value stored at location } l \]

c1) Boolean expressions

\[ B ::= \text{True} \mid \text{False} \mid E \text{ bop } E \mid \neg B \mid B \land B \]

\[ \text{bop ::= } > \mid < \mid = \]
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:

Quit: # 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 elements:

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

Let notation: \( m[l \mapsto n] \) updates \( m \) with new mapping \( l \mapsto n \)

Let \( m[l \mapsto n](l) = n \)

\( m[l \mapsto n](l') = m(l') \) for all \( l' \neq l \)
Abstract machine = transition system

Configuration: \( \langle c, r, m \rangle \)

- \( c ::= \text{nil} \mid \text{io c} \) \( \uparrow \) instruction is pushed on top of \( c \)
- empty stack

- \( i ::= \text{if} \mid \text{while} \)

- \( r ::= \text{nil} \mid \text{for} \mid \text{cor} \).
Model execution of programs as sequences of transitions from initial state to final state.

\[ \langle \text{Co-nil}, \text{nil}, m \rangle \quad \langle \text{nil}, \text{nil}, m' \rangle \]

Execute C in a given memory state m.

Stop when all stacks are empty.

Transition rules:

\[ \langle \text{c, r, m} \rangle \rightarrow \langle \text{c', r', m'} \rangle \]
a) Evaluating expressions

\[ \langle n \circ c, r, m \rangle \rightarrow \langle c, n \circ r, m \rangle \]

- pop \( n \) from \( c \) and push it onto \( r \)

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

- read memory at \( l \) & push it onto \( r \)

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

\[ \langle \circ \circ c, n \circ n \circ r, m \rangle \rightarrow \langle c, n \circ r, m \rangle \text{ if } n_1 \circ n_2 = n \]

(similar to Boolean expression)
Semantics of SIMP expressions

Value of $E$ in state $m$ is $v$ if there is a sequence of transitions $<E;c,r,m> \xrightarrow{*} <c, vor, m'>$

(similar for Boolean expressions)

Example: What is the value of $E = !a + !b$ in state $m = \{ a \rightarrow 3, b \rightarrow 1 \}$

Quiz: # transitions?
\[
\langle !a + !b \circ \text{nil}, \text{nil}, m \rangle \\
\to \langle !a \circ !b \circ \text{nil}, \text{nil}, m \rangle \\
\to \langle !b \circ \text{o ml}, 3 \circ \text{ml}, m \rangle \\
\to \langle \text{o nil}, 1 \circ 3 \circ \text{nil}, m \rangle \\
\to \langle \text{ml}, 4 \circ \text{ml}, m \rangle \\
\]

Answer: 4