Program Analysis – Lecture 2
Operational Semantics (Part 1)

Join the course on Ilias! See link on
http://software-lab.org/teaching/winter2019/pa/

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Winter 2019/2020
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

```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
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
var e = eval;

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

Correct answer: 5

Indirect eval():
Works in global scope rather than local scope
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 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 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 here

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

a) Transition system
- set Config of configurations or state
- binary relation \( \rightarrow \subseteq \text{Config} \times \text{Config} \)
  ("transition relation")
\[
c \rightarrow c' \quad \text{transition (or change of state)}
\]
\Rightarrow \text{step of computation}

\[\rightarrow^* \quad \text{reflexive, transitive closure of} \rightarrow\]

\[
\forall c. \ c \rightarrow^* c \\
\forall c, c', c''. \ c \rightarrow^* c' \land c' \rightarrow^* c'' \Rightarrow c \rightarrow^* c''
\]
deterministic: \( c \rightarrow c_1 \land c \rightarrow c_2 \Rightarrow c_1 = c_2 \)
b) **Rule induction**

Let define a set ("inductive set") with:
- a finite set of basic elements ("axioms"): (empty)
- a finite set of rules that specify how to generate more elements:

\[ h_1 \ldots h_n \rightarrow c \]

**Ex. 1** set of natural numbers

axiom: \( 0 \)

rule: \( n \rightarrow n + 1 \)
Ex. 2 Evaluation of expressions, e.g., \((3, 4)\)

- Set of pairs of ASTs & values

**Notation:** \(E \Downarrow n\) — expression \(E\) evaluates to number \(n\)

**Axioms:** \(1 \Downarrow 1, 2 \Downarrow 2, \text{ etc.}\)

**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:**

\[
\frac{E_1 \Downarrow n_1 \quad E_2 \Downarrow n_2}{\rho (E_1, E_2) \Downarrow n} \quad \text{if } n = n_1 \rho n_2
\]
Ex. 1

Ex. 2 Show that

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

Quiz: # axiom? 3
# rules? 2

\[
\begin{align*}
3 & \downarrow 3 & 4 & \downarrow 4 \\
\_ & \_ & \_ & \_ \\
+ (3, 4) & \downarrow 7 \\
\_ & \_ & \_ & \_ \\
- (+(3, 4), 1) & \downarrow 6
\end{align*}
\]
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: assignment, sequencing, conditional, loops, integer variables

- abstract syntax

  \[ P ::= C | E | B \]
a) **Commands**

- \( i := \)

  - \( l := E \) (assignment of expression \( E \) to label \( l \))

  - \( B \) (two-way selector)

- \( B \) (while loop)

  - \( C \) (while loop)

**Textual notation:**

- \( C := C; C \mid l := E \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \mid \text{while } B \text{ do } C \mid \text{skip} \)
5) **Integer expressions**

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

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

where \( n \) .. integer

\( l \in L = \{ l_0, l_1, \ldots \} \) ... memory locations

\( !l \) .. value stored at location \( l \)

6) **Boolean expressions**

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

\[ \text{bop} ::= > \mid < \mid = \]
Ex. 1 \[ z := !x ; (x := !y ; y := !z) \]

... swap values in x and y

AST:
Ex. 2 \[\text{while } (!e > 0) \text{ do } (\]
\[f := !f \times !e;\]
\[e := !e - 1)\]

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