Program Analysis

Operational Semantics (Part 1)

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Winter 2020/2021
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 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 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.

Note: Mistake in video is fixed
Specifying the semantics of programs

- Static semantics, e.g., type
- Dynamic semantics
  - Denotational
  - Axiomatic
  - Operational ← Focus here

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

a) Transition systems

* set Config of configurations or states
* binary relation $\rightarrow \subseteq \text{Config} \times \text{Config}$
  ("transition relation")

$c \rightarrow c'$ \quad \text{transition (or change of state)}
\quad \text{a step of computation}

\text{deterministic}: \quad c \rightarrow c_1 \land c \rightarrow c_2 \Rightarrow c_1 = c_2

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

\forall c. c \Rightarrow^* c \quad \forall c, c', c''. c \Rightarrow^* c' \land c' \Rightarrow^* c'' \Rightarrow c \Rightarrow^* c''
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 more elements:
     \[ b_1, \ldots, b_n \]

2) **Ex. 1** set of natural nums.

   - **axiom:** \( 0 \)
   - **rule:** \( \frac{n}{n+1} \)
Ex. 2 Evaluation of expressions, e.g., \((3, 4)\)

Let \(\mathcal{S} = \text{pairs of ASTs} \& \text{values}\)

Notation: \(E \Downarrow n\) .. expression \(E\) evaluates to \(n\)

Axioms: 
\[1 \Downarrow 1, \quad 2 \Downarrow 2, \quad \text{etc.}\]

axiom scheme: \(n \Downarrow n\)

Rules: 
\[E_1 \Downarrow n_1, \quad E_2 \Downarrow n_2 \quad \text{if} \quad n = n_1 + n_2 \quad \text{etc.}\]

\[
\begin{array}{c}
+ (E_1, E_2) \Downarrow n \quad \rule{4cm}{0.5pt}
\end{array}
\]

rule scheme:
\[E_1 \Downarrow n_1, \quad E_2 \Downarrow n_2 \quad \text{if} \quad n = n_1 \circ n_2 \]

\[
\begin{array}{c}
\circ (E_1, E_2) \Downarrow n
\end{array}
\]
c) Proof tree
To show that an element is in an inductive set

Ex. 1

Ex. 2 show that

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

\[\begin{align*}
0 \\
1 \\
2
\end{align*}\]

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

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