Structural operationanal semantics of imperative programming language

William Steingartner

william.steingartner@tuke.sk

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice, Slovakia

Semantics of programming languages

Spring 2022/2023

Theme 03 – Structural operational semantics

Structural operational semantics

In Structural operational semantics, the emphasis is on the **individual steps** of the execution, that is the execution of assignments and tests.

Configuration has the form

$$\langle S, s \rangle.$$

The **transition relation** in the structural operational semantics expresses the first step of the execution of S from state s:

$$\langle S, s \rangle \Rightarrow \alpha,$$

where α denotes

- **state** s',
- or **configuration** $\langle S', s' \rangle$, resp.

The symbol '$\Rightarrow$' denotes one-step **transition** in structural operational semantics.

Structural operational semantics

The transition relation has two forms:

- $\langle S, s\rangle \Rightarrow s'$ means that execution of S from s has **terminated** and the final state is s',
- $\langle S, s\rangle \Rightarrow \langle S', s'\rangle$ means that execution of S is **not completed** and the remaining computation is expressed by the intermediate configuration $\langle S', s'\rangle$.

We shall say that $\langle S', s'\rangle$ is *stuck* if there is no such transition

$$\langle S, s\rangle \Rightarrow s' \quad \text{or} \quad \langle S, s\rangle \Rightarrow \left\langle S', s'\right\rangle .$$

The execution of statement S from state s is **stopped**.

Semantics of statements

Derivation rules for language *Jane* are as follows:

$$\langle x{:=}e, s\rangle \Rightarrow s[x \mapsto \mathscr{E}[\![\, e \,]\!]s] \quad (1_{os})$$

$$\langle \texttt{skip}, s\rangle \Rightarrow s \quad (2_{os})$$

The rules for a **variable assignment** and **empty statement**:

- are axioms,
- are fully executed in one step,
- have not changed at all, only transition symbol changed to $\Rightarrow$.

Semantic of statements

The rules for **composition** express that to execute S_1, S_2 in state s, we first execute S_1 one step from s:

$$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1, S_2, s\rangle \Rightarrow \langle S_1', S_2, s'\rangle} \ (3^1_{os})$$

$$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1, S_2, s\rangle \Rightarrow \langle S_2, s'\rangle} \ (3^2_{os})$$

There are two possible outcomes:

- if the execution of S_1 has not been completed, we have to complete it before embarking on the execution of S_2 – the rule (3^1_{os}),
- if the execution of S_1 has been completed, we can start on the execution of S_2 – the rule (3^2_{os}),
- although we have two rules for composition, their application is clear.

Semantic of statements

From the rules for **conditional statement** we see that the first step in executing the statement is to perform the test and to select the appropriate branch:

$$\frac{\mathscr{B}[\![\, b \,]\!] s = \mathbf{tt}}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle} \; (4_{os}^{\mathbf{tt}})$$

$$\frac{\mathscr{B}[\![\, b \,]\!] s = \mathbf{ff}}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle} \; (4_{os}^{\mathbf{ff}})$$

Both rules are symmetric, they differ only in a value of the Boolean expression. We clearly now when to use concrete rule from them.

Semantic of statements

For the **loop** statement, there is only one axiom:

$$\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \Rightarrow \langle \texttt{if}\ b\ \texttt{then}\ (S, \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else}\ \texttt{skip}, s\rangle \qquad (5_{os})$$

- the axiom shows that the first step in the execution of `while`-construct is to unfold it one level, that is to rewrite it as a conditional,
- the test will therefore be performed in the second step of the execution, where one of the rules for the `if`-construct is applied.

After the first step, an initial state s remains unchanged.

Structural operational semantics of a program

Structural operational semantics of a program P is determined from a derivation sequence.
A **derivation sequence** of a statement S starting in one state s is either:

1. a **finite** sequence

$$\alpha_0, \alpha_1, \ldots, \alpha_n$$

of configurations satisfying:

- the first element, an **initial configuration** is $\alpha_1 = \langle P, s_0 \rangle$,
- the next element arises by applying some of derivation rules, $\alpha_i \Rightarrow \alpha_{i+1}$, for $0 \leq i < k$, $k \geq 0$,
- if the last element is state s' then it is a **final state** after the program P execution,
- if the last element is configuration then the execution of program is **stopped** and final state of program **does not exist**.

Structural operational semantics of program

2. an **infinite** sequence

$$\alpha_0, \alpha_1, \alpha_n, \ldots$$

of configurations satisfying $\alpha_0 = \langle S, s \rangle$ and $\alpha_i \Rightarrow \alpha_{i+1}$ for $0 \leq i$.

We introduce the following **conventions** for derivation sequences. We shall write:

- $\alpha_i \Rightarrow \alpha_{i+1}$ represents the execution of **one step** in program,
- $\alpha \Rightarrow^k \alpha'$ indicates that there are k steps in the execution from α to α',
- $\alpha \Rightarrow^* \alpha'$ indicates that there is a **finite number** of steps.

Example 1

Let P be a program $P = x{:=}y - 5; \texttt{while}\ x \leq y\ \texttt{do}\ (x{:=}x + 3; y{:=}y - x)$ with an initial state $s_0\ y = \mathbf{10}$.

The derivation sequence is:

$$
\begin{aligned}
\alpha_0 &= \langle P, s_0\rangle \Rightarrow \\
\alpha_1 &= \langle \texttt{while}\ x \leq y\ \texttt{do}\ (x{:=}x + 3; y{:=}y - x), s_1\rangle \Rightarrow \\
\alpha_2 &= \langle \texttt{if}\ \ x \leq y\ \texttt{then}\ (x{:=}x + 3; y{:=}y - x; \texttt{while}\ x \leq y\ \texttt{do}\ (x{:=}x + 3; y{:=}y - x)) \\
&\qquad\qquad \texttt{else}\ \ \texttt{skip}, s_1\rangle \Rightarrow \\
\alpha_3 &= \langle x{:=}x + 3; y{:=}y - x; \texttt{while}\ x \leq y\ \texttt{do}\ (x{:=}x + 3; y{:=}y - x), s_1\rangle \Rightarrow \\
\alpha_4 &= \langle y{:=}y - x; \texttt{while}\ x \leq y\ \texttt{do}\ (x{:=}x + 3; y{:=}y - x), s_2\rangle \Rightarrow \\
\alpha_5 &= \langle \texttt{while}\ x \leq y\ \texttt{do}\ (x{:=}x + 3; y{:=}y - x), s_3\rangle \Rightarrow \\
\alpha_6 &= \langle \texttt{if}\ \ x \leq y\ \texttt{then}\ (x{:=}x + 3; y{:=}y - x; \texttt{while}\ x \leq y\ \texttt{do}\ (x{:=}x + 3; y{:=}y - x)) \\
&\qquad\qquad \texttt{else}\ \ \texttt{skip}, s_3\rangle \Rightarrow \\
\alpha_7 &= s_3
\end{aligned}
$$

$$
\begin{aligned}
s_1 &= s_0[x \mapsto \mathbf{5}] & \mathscr{B}[\![\, x \leq y \,]\!]s_1 &= \mathbf{tt} \\
s_2 &= s_1[x \mapsto \mathbf{8}] & & \\
s_3 &= s_2[y \mapsto \mathbf{2}] & \mathscr{B}[\![\, x \leq y \,]\!]s_3 &= \mathbf{ff} \\
s &= s_3 = [x \mapsto \mathbf{8}, y \mapsto \mathbf{2}] &&
\end{aligned}
$$

Statement `stop`

Now we extend language *Jane* with the simple statement `stop`. The idea is that statement `stop` **stops** the execution of the complete program.

Formally, the new syntax of statements is given by:

$$S ::= \ldots \mid \texttt{stop}.$$

Structural operational semantics of given statement is defined as follows:

- an execution of the program is modeled by ensuring that the configurations of the form

  $$\langle \texttt{stop}, s \rangle$$

  are **stuck**.

Statement Stop

In **natural semantics**, the following statements are semantically equivalent:

$$\texttt{stop} \quad \text{and} \quad \texttt{while true do skip}$$

because semantic function is not defined for any initial state s.

From the **structural operation semantics** point of view, it is clear now that `stop` and `while true do skip` cannot be sematically equivalent because:

$$\begin{aligned}
\alpha_0 &= \langle \texttt{while true do skip}, s\rangle \Rightarrow \\
\alpha_1 &= \langle \texttt{if true then (skip; while true do skip) else skip}, s\rangle \Rightarrow \\
\alpha_2 &= \langle \texttt{skip; while true do skip}, s\rangle \Rightarrow \\
\alpha_3 &= \langle \texttt{while true do skip}, s\rangle \Rightarrow \ldots
\end{aligned}$$

This is an infinite derivation sequence whereas `stop` has none.

Statement `or`

We extend the language *Jane* with the command `or`, which introduces **nondeterminism** into the programming language.

We extend the **syntax** with a new alternative to the production rule for commands:

$$S ::= \ldots \mid S \ \texttt{or} \ S$$

Structural operational semantics of the statement `or` we define with two axioms:

$$\langle S_1 \ \texttt{or} \ S_2, s\rangle \Rightarrow \langle S_1, s\rangle \quad (6^1_{os})$$
$$\langle S_1 \ \texttt{or} \ S_2, s\rangle \Rightarrow \langle S_2, s\rangle \quad (6^2_{os})$$

We can use **any** of the given axioms, but there is no guarantee that the command will be executed according to the chosen rule. We see that even the structural operational semantics does not provide an unambiguous semantics of non-deterministic programs.

Properties of structural operational semantics

We shall say that the execution of statement S in state s

- **terminates** if and only if there is a finite derivation sequence starting with configuration $\langle S, s \rangle$,
- **loops** if and only if there is an infinite derivation sequence starting with configuration $\langle S, s \rangle$.

Structural operational semantics distinguishes **the reason** of non-termination:

- if the derivation sequence is infinite then program **loops**, in contrast with the situation
- if the derivation sequence is finite and the last element is a stuck then execution of program is **stopped** and final state does not exist.

Properties of structural operational semantics

We say that statements S_1 and S_2 are **semantically equivallent**

- if it holds for every state s

$$\langle S_1, s\rangle \Rightarrow^* \alpha \quad \text{iff} \quad \langle S_2, s\rangle \Rightarrow^* \alpha$$

  where α can be final state or configuration,
- if for both statements the outcomes are infinite sequences.

Lengths of derivation sequences for two semantically equivalent statements can be **different**.

Structural operational semantics of language $Jane$ is **deterministic**.

Semantic function

The meaning of statements can be summarized by a (partial) function from $\mathbf{State}$ to $\mathbf{State}$:

$$\mathscr{S}_{os} : \mathbf{Statm} \to (\mathbf{State} \rightharpoonup \mathbf{State}).$$

Properties:

- specification of function expresses that the result is change of state,
- symbol "$\rightharpoonup$" expresses that function is partial,
- function is given by

$$\mathscr{S}_{os}[\![S]\!]s = \begin{cases} s', & \text{if } \langle S, s\rangle \Rightarrow^* s', \\ \bot, & \text{otherwise,} \end{cases}$$

- symbol $\bot$ expresses that in given state function is not defined and the execution of statement when

$$\mathscr{S}_{os}[\![S]\!]s = \bot$$

is not defined.

Equivalence of natural and structural operational semantics

Natural and structural operational semantics:

- describe the meaning of programs in different ways,
- cover different approaches,
- must to provide **the same meaning** of one program,
- then it is necessary to prove an **equivalence** of both methods.

The proof proceeds by structural induction.

Proof of equivalence

Theorem. For every statement S of *Jane* we have

$$\mathscr{S}_{ns}[\![S]\!] = \mathscr{S}_{os}[\![S]\!].$$

This result expresses two properties:

- if the execution of S from some state terminates in one of the semantics then it also terminates in the other and the resulting states will be equal,
- if the execution of S from some state loops in one of the semantics then it will also loop in the other.

For every statement S we have

1. $\langle S, s\rangle \to s'$ implies $\langle S, s\rangle \Rightarrow^* s'$,
2. $\langle S, s\rangle \Rightarrow^k s'$ implies $\langle S, s\rangle \to s'$.

The **first implication** expresses that if exists a transition for statement S in natural semantics then there exists a finite derivation sequence in structural operational semantics which the last element is final state.

The **second implication** expresses that if a derivation sequence for statement S in structural operational semantics has a final state s then there exists a transition in natural semantics with the same final state.