Natural 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 – Natural Semantics

Basic notions

Operational semantics is concerned with how to execute programs. We are interested in **how the states are modified** during the execution of the statement.

- Semantics of expressions only **inspects the state** in order to determine the value of the expression.
- Semantics of statements **modifies** the states.

Basic notions

In operational semantics, the meaning of statements will be specified by transition system. Both kinds of semantics use configuration of the form

$$\langle S, s\rangle$$

representing that the statement S is to be executed from the state s.

An execution of the statement S in its initial state s causes the change of state. Transition relation in particular semantics looks as follows:

- $\langle S, s\rangle \to s'$ in natural semantics,
- $\langle S, s\rangle \Rightarrow s'$ in structural operational semantics.

A **definition of operational semantics** of programming language does not use semantic equations but:

- **axioms**, and
- **derivation rules**.

Basic notions in operational semantics

A **rule** has a general form:

$$\frac{transition_1,\ \ldots,\ transition_n,\ condition}{transition}\ (\mathrm{name})$$

where

- above the solid line **premises** are written,
- below the solid line stands one **conclusion**,
- $transition_i$ consists of S_i which is immediate constituent of S or statement constructed from the immediate constituents of S,
- $condition$ (not in every rule) has to be fulfilled whenever the rule is applied.

Furthermore,

- rule with an empty set of premises is called **axiom** and the solid line is omitted,
- rule without premises, only with condition is called **simple derivation rule**.

Derivation rule is a **schema** allowing set the data from concrete programming language.

Natural semantics

In the **natural semantics**, we are concerned with the relationship between the initial and the final state of an execution.

The purpose of natural semantics is to describe how the overall results of execution are obtained.

Transition relation in natural semantics we shall write as

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

Intuitively, this means that the execution of statement S from state s will terminate and the resulting state will be s'.

Natural Semantics

Rule in natural semantics has a general form

$$\frac{\langle S_1, s_0\rangle \rightarrow s_1, \ldots, \langle S_n, s_{n-1}\rangle \rightarrow s}{\langle S, s_0\rangle \rightarrow s} \quad (\mathrm{n}_{ns})$$

where

- statements $S_1, \ldots, S_{n-1}$ are **constituents** (parts, components) of statement S;
- s_0 is an **initial state**;
- s is a **final (resulting) state**;
- $s_1, \ldots, s_{n-1}$ are states during the program execution.

Natural semantics

A rule

$$\frac{\langle S_1, s_0\rangle \to s_1, \ldots, \langle S_n, s_{n-1}\rangle \to s}{\langle S, s_0\rangle \to s}\ (\mathrm{n}_{ns})$$

is read as follows:

if an execution of statement S_1 in state s_0 causes change of state to s_1, ..., and execution of statement S_n in state s_{n-1} causes change of state to state s, then the execution of the statement S in an initial state s_0 causes a change to the final state s.
Each transition is **a proposition**, and the rule says: from the true propositions – **premises** implies the true proposition – **conclusion**.

Natural semantics of an assignment

Natural semantics of a variable assignment is defined with the axiom

$$\langle x := e, s\rangle \to s[x \mapsto \mathscr{E}[\![\, e \,]\!]s] \qquad (1_{ns})$$

This axiom says:

- in an initial state s, statement is executed to yield a final state $s[x \mapsto \mathscr{E}[\![\, e \,]\!]s]$ which is as s except that x has the value $\mathscr{E}[\![\, e \,]\!]$.

Natural semantics of an **empty statement** is defined as follows:

$$\langle \mathtt{skip}, s\rangle \to s \qquad (2_{ns})$$

- This statement does not change the state.

Example 1

We consider the following statement and an initial state $s_0 = [x \mapsto \mathbf{3}]$:

$$x{:=}x + 1$$

Natural semantics of this statement we find by applying the rule (1_{ns}):

$$\langle x{:=}x + 1, s_0\rangle \to s_0[x \mapsto \mathscr{E}[\![\, x + 1 \,]\!]s_0]$$

After evaluation of an expression $x + 1$ in the state s_0:

$$\begin{aligned}
&\mathscr{E}[\![\, x + 1 \,]\!]s_0 = \mathscr{E}[\![\, x \,]\!]s_0 \oplus \mathscr{E}[\![\, 1 \,]\!]s_0 \\
&= s_0\ x \oplus \mathscr{N}[\![\, 1 \,]\!] \\
&= \mathbf{3} \oplus \mathbf{1} \\
&= \mathbf{4}
\end{aligned}$$

we get a result of an assignment statement:

$$\langle x{:=}x + 1, s_0\rangle \to s_0[x \mapsto \mathbf{4}]$$

The resulting state is $s = [x \mapsto \mathbf{4}]$.

☐

Natural semantics of statements sequence

Natural semantics of statements sequence is defined as follows:

$$\frac{\langle S_1, s\rangle \to s', \qquad \langle S_2, s'\rangle \to s''}{\langle S_1; S_2, s\rangle \to s''} \qquad (3_{ns})$$

This rule means intuitively:

- to execute $S_1; S_2$ from state s we must first execute S_1 from s. Assuming that this yields a final state s' we shall then execute S_2 from s'. The premises of the rule are concerned with the two statements S_1 and S_2, whereas the conclusion expresses a property of the composite statement itself.

Example 2

The following is an instance of the rule:

$$y := x + 5; x := x - 1$$

An initial state is $s_0 = [x \mapsto \mathbf{1}, y \mapsto \mathbf{1}]$.

Here S_1 is instantiated to $y := x + 5$, statement S_2 to $x := x - 1$:

$$\frac{\langle y := x + 5, s_0\rangle \to s_1, \qquad \langle x := x - 1, s_1\rangle \to s}{\langle y := x + 5; x := x - 1, s_0\rangle \to s}$$

The resulting state:

$$\begin{aligned} s_1 &= s_0[y \mapsto \mathscr{E}[\![\, x + 5 \,]\!] s_0] = s_0[y \mapsto \mathbf{6}] \\ s &= s_1[x \mapsto \mathbf{0}] \end{aligned}$$

☐

Natural semantics of conditional statement

For the **conditional statement** we have two rules:

$$\frac{\langle S_1, s\rangle \to s', \quad \mathscr{B}[\![\, b \,]\!]s = \mathbf{tt}}{\langle \texttt{if}\ \ b\ \ \texttt{then}\ \ S_1\ \ \texttt{else}\ \ S_2,\ s\rangle \to s'} \qquad (4^{\mathbf{tt}}_{ns})$$

$$\frac{\langle S_2, s\rangle \to s', \quad \mathscr{B}[\![\, b \,]\!]s = \mathbf{ff}}{\langle \texttt{if}\ \ b\ \ \texttt{then}\ \ S_1\ \ \texttt{else}\ \ S_2,\ s\rangle \to s'} \qquad (4^{\mathbf{ff}}_{ns})$$

The first rule says:

- to execute conditional statement $\texttt{if}\ \ b\ \ \texttt{then}\ \ S_1\ \ \texttt{else}\ \ S_2$ we simply execute S_1 provided that b evaluates to $\mathbf{tt}$ in the actual state.

The second rule says that:

- if b evaluates to $\mathbf{ff}$ then to execute $\texttt{if}\ \ b\ \ \texttt{then}\ \ S_1\ \ \texttt{else}\ \ S_2$ we just execute S_2.

Example 3

Taking $s_0\ x = \mathbf{1}$ the following is an instance of the rule $(4^{\mathbf{ff}}_{ns})$:

$$\texttt{if}\ \ \neg(x = 1)\ \ \texttt{then skip else}\ \ x{:=}\ x + 5$$

The first step is to evaluate the condition with the state s_0:

$$\mathscr{B}[\![\, \neg(x = 1)\,]\!] s_0 = \ldots = \mathbf{ff}$$

Then we apply the rule $(4^{\mathbf{ff}}_{ns})$:

$$\frac{\langle x{:=}x + 5, s_0\rangle \to s, \quad \mathscr{B}[\![\, \neg(x = 1)\,]\!] s_0 = \mathbf{ff}}{\langle \texttt{if}\ \ \neg(x = 1)\ \ \texttt{then skip else}\ \ x{:=}x + 5,\ s_0\rangle \to s}$$

The resulting state is:

$$s = s_0[x \mapsto \mathscr{E}[\![\, x + 5\,]\!]\ s_0] = s_0[x \mapsto \mathbf{6}]$$

$\square$

Natural semantics of the loop

Finally, we have one rule and one axiom expressing how to execute the **loop** statement:

$$\frac{\langle S, s\rangle \to s', \langle \texttt{while}\ b\ \texttt{do}\ S, s'\rangle \to s'', \mathscr{B}[\![\, b \,]\!]s = \mathbf{tt}}{\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s''} \qquad (5^{\mathbf{tt}}_{ns})$$

$$\frac{\mathscr{B}[\![\, b \,]\!]s = \mathbf{ff}}{\langle \texttt{while}\ b\ \texttt{do}\ \ S,\ s\rangle \to s} \qquad (5^{\mathbf{ff}}_{ns})$$

Intuitively, the meaning of the construct `while` b `do` S in the state s can be explained as follows:

- if the test b evaluates to true in the state s, then we first execute the body of the loop and then continue with the loop itself from the state so obtained;
- if the test b evaluates to false in the state s, then the execution of the loop terminates.

Example 4

Consider the following cycle

$$\texttt{while}\ \ (x \leq 2)\ \ \texttt{do}\ \ x{:=}y+2$$

and an initial state $s_0 = [x \mapsto \mathbf{0}, y \mapsto \mathbf{1}]$.

We apply the rule $(5^{\mathtt{tt}}_{ns})$ in the first step and the rule $(5^{\mathbf{ff}}_{ns})$ in the second step:

$$\frac{\langle x{:=}y+2, s_0\rangle \to s_1, \quad \dfrac{\mathscr{B}[\![\,(x \leq 2)\,]\!]s_1 = \mathbf{ff}}{\langle \texttt{while}\ (x \leq 2)\ \texttt{do}\ x{:=}y+2, s_1\rangle \to s}, \quad \mathscr{B}[\![\,(x \leq 2)\,]\!]s_0 = \mathtt{tt}}{\langle \texttt{while}\ (x \leq 2)\ \texttt{do}\ x{:=}y+2, s_0\rangle \to s}$$

where:

$$\begin{aligned} s_1 &= s_0[x \mapsto \mathbf{3}] \\ s = s_1 &= [x \mapsto \mathbf{3}, y \mapsto \mathbf{1}] \end{aligned}$$

$\square$

Derivation tree

When we use the axioms and rules to derive a transition $\langle S, s\rangle \to s'$ we obtain a **derivation tree**.

- The root of derivation tree is $\langle S, s\rangle \to s'$,
- the **leaves** are instances of axioms,
- the **internal nodes** are conclusions of instantiated rules and they have corresponding premises as their immediate sons.

A derivation tree is called **simple** if it is an instance of axiom, otherwise is is called **composite**.

Example 5

Consider a program $P = x{:=}y-5; \texttt{while}\ x \le y\ \texttt{do}\ (x{:=}x+3; y{:=}y-x)$ and an initial state $s_0\ y = \mathbf{10}$.
We apply the substitutions: $S_1 = x{:=}x+3; \quad S_2 = y{:=}y-x$. Then we construct a derivation tree:

$$\dfrac{\dfrac{}{\langle x{:=}y-5,s_0\rangle\to s_1} \qquad \dfrac{\dfrac{\dfrac{}{\langle x{:=}x+3,s_1\rangle\to s_3}\ \dfrac{}{\langle y{:=}y-x,s_3\rangle\to s_2}}{\langle S_1;S_2,s_1\rangle\to s_2} \quad \dfrac{\mathscr{B}[\![\, x\le y \,]\!]s_2=\mathbf{ff}}{\langle \texttt{while}\ x\le\ y\ \texttt{do}\ (S_1;S_2),s_2\rangle\to s}, \ \mathscr{B}[\![\, x\le y \,]\!]s_1=\mathbf{tt}}{\langle \texttt{while}\ x\le\ y\ \texttt{do}\ (S_1;S_2),s_1\rangle\to s}}{\langle P,s_0\rangle\to s}$$

States s_1, s_2, s arise by applying the derivation rules:

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

$\square$

Properties of natural semantics

We shall introduce the following terminology.

The execution of a statement S on a state s

- **terminates** if and only if there is a state s' such that
$$\langle S, s\rangle \rightarrow s',$$
- **always terminates** if it terminates in any state.

We say that the statement S

- **loops** if and only if there is **no** state s' such that
$$\langle S, s\rangle \rightarrow s',$$
- **never terminates** if it does not terminate for any state.

A program has a meaning in natural semantics only if it terminates after finite number of steps (statements).

Properties of Natural Semantics

We say that statement S_1 is **semantically equivalent** with the statement S_2 if it holds for all states s and s':

$$\langle S_1, s\rangle \to s' \qquad \text{iff} \qquad \langle S_2, s\rangle \to s'.$$

In other words: two different statements in the same initial state provide the same resulting state.

Example 6

The following two statements S_1 and S_2 are semantically equivalent.

$$\begin{aligned} S_1 &= \texttt{while}\ b\ \texttt{do}\ S, \\ S_2 &= \texttt{if}\ b\ \texttt{then}\ (S, \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else}\ \texttt{skip}. \end{aligned}$$

Proof: is in two stages. We must to prove two implications:

1. if $\langle S_1, s\rangle \to s'$ then $\langle S_2, s\rangle \to s'$,
2. if $\langle S_2, s\rangle \to s'$ then $\langle S_1, s\rangle \to s'$,

Properties of natural semantics

The natural semantics of $Jane$ is **deterministic** if for all elements S, s, s', s'' we have that

$$\langle S, s\rangle \to s' \quad \text{and} \quad \langle S, s\rangle \to s'' \quad \text{imply} \quad s' = s''.$$

Theorem: The natural semantics of $Jane$ is deterministic.
Proof: by using structural induction.

The semantic function

The meaning of statements can now be summarized as a (partial) function from $\mathbf{State}$ to $\mathbf{State}$. Its specification is:

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

- and this means that for every statement S we have a partial function $\mathscr{S}_{ns}[\![S]\!] \in \mathbf{State} \rightharpoonup \mathbf{State}$,
- symbol "$\rightharpoonup$" expresses that function is partially defined,
- semantic function is given by

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

- symbol $\bot$ denotes an undefined value, i.e. the execution of statement

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

does not provide result in a state s.