Programming Languages and Compiler Design

Natural Operational Semantics of Language **While**

Yliès Falcone, Jean-Claude Fernandez

Master of Sciences in Informatics at Grenoble (MoSIG)
Univ. Grenoble Alpes
(Université Joseph Fourier, Grenoble INP)

Academic Year 2015 - 2016

About Operational Semantics

Semantics is

- concerned with the *meaning* of grammatically correct programs;
- defined on abstract syntax trees, obtained after type analysis.

With Operational Semantics the meaning of a construct tells how to execute it.

Semantics is described in terms of *sequences of configurations*, which give the state-history of the machine.

Outline

Syntax of Language **While**

Semantics of Expressions in Language **While**

(Natural) Operational Semantics of Language **While**

Outline

Syntax of Language **While**

Semantics of Expressions in Language **While**

(Natural) Operational Semantics of Language **While**

Meta-Variables

Meta-variables:

- x: variable
- S: statement
- a: arithmetic expression
- b: Boolean expression

Meta-variables can be primed or sub-scripted

Example (Meta-Variables)

- variables: $x, x', x_1, x_2, \ldots$
- statements: $S, S_1, S', \ldots$
- arithmetic expressions: $a_1, a_2, \ldots$
- Boolean expressions: $b_1, b', b_2, \ldots$

Abstract Grammar of language **While**

Definition (Abstract Grammar of language **While**)

$$\begin{array}{rcl} S & ::= & x := a \mid \text{skip} \\ & & \mid S; S \\ & & \mid \text{if } b \text{ then } S \text{ else } S \text{ fi} \\ & & \mid \text{while } b \ \text{ do } S \text{ od} \end{array}$$

Remark This is an *inductive* definition:

- $x := a$ and skip are basis elements;
- $S; S$, if b then S else S fi, while b do S od are composition rules to define composite elements.

□

Syntactic Categories

- Numbers

$$n \in \mathbf{Num} = \{0, \ldots, 9\}^+$$

- Variables

$$x \in \mathbf{Var}$$

- Arithmetic expressions

$$\begin{array}{lcl} a & \in & \mathbf{Aexp} \\ a & := & n \mid x \mid a + a \mid a * a \mid a - a \end{array}$$

Num, **Var**, and **Aexp** are *syntactic categories*.

Remark Other operators for artihmetical expressions can be defined from the proposed ones. $\square$

Syntactic categories (ctd)

- Boolean expressions

$$\begin{array}{lll} b & \in & \textbf{Bexp} \\ b & := & \text{true} \mid \text{false} \mid a = a \mid a \leq a \mid \neg b \mid b \wedge b \end{array}$$

- Statements

$$\begin{array}{lll} S & \in & \textbf{Stm} \\ S & ::= & x := a \mid \text{skip} \mid S; S \\ & & \mid \text{if } b \text{ then } S \text{ else } S \text{ fi} \\ & & \mid \text{while } b \ \text{ do } S \text{ od} \end{array}$$

Bexp and **Stm** are *syntactic categories*.

Concrete vs. abstract syntax

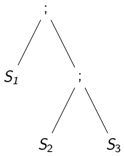
We focus on *abstract syntax* and abstract away concrete syntax.

- Term $S_1; S_2$ represents the tree, s.t.
  - the root is ;
  - left child is S_1 tree
  - right child is S_2 tree

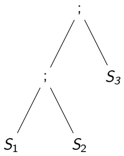
- Parenthesis shall be used to avoid ambiguities.

Example (Abstract Syntax Tree)

- $S_1; (S_2; S_3)$



- $(S_1; S_2); S_3$



We will only use the linear notation.

Outline - Natural Operational Semantics of Language **While**

Semantic domains

- Integers: $\mathbb{Z}$
- Booleans: $\mathbb{B} = \{\mathbf{tt}, \mathbf{ff}\}$
- States:

$$\mathbf{State} = \mathbf{Var} \to \mathbb{Z}$$

  Intuition: a state is a "memory"

Definition (Substituing a value to a variable)

Let $v \in \mathbb{Z}$. Then, $\sigma[y \mapsto v]$ denotes the state σ' such that:

$$\text{for all } x \in \mathbf{Var}, \sigma'(x) = \begin{cases} \sigma(x) & \text{if } x \neq y, \\ v & \text{otherwise.} \end{cases}$$

Example (Substitution)

For $\sigma = [x \mapsto 0, y \mapsto 1]$:

- $\sigma[x \mapsto 2] = [x \mapsto 2, y \mapsto 1]$,
- $\sigma(z) = \mathit{undef} = \sigma[x \mapsto 2](z)$.

Semantic functions

- Numerals: integers

$$\begin{array}{rcl} \mathcal{N} & : & \textbf{Num} \to \mathbb{N} \\ \mathcal{N}(n_1 \cdots n_k) & = & \Sigma_{i=1}^{k} n_i \times 10^{k-i} \end{array}$$

- Arithmetic expressions: for each state, a value in $\mathbb{Z}$

$$\mathcal{A} : \textbf{Aexp} \to (\textbf{State} \to \mathbb{Z})$$

$$\begin{array}{c} \mathcal{A}[n]\sigma = \mathcal{N}(n) \\ \mathcal{A}[x]\sigma = \sigma(x) \\ \mathcal{A}[a_1 + a_2]\sigma = \mathcal{A}[a_1]\sigma +_I \mathcal{A}[a_2]\sigma \\ \mathcal{A}[a_1 * a_2]\sigma = \mathcal{A}[a_1]\sigma *_I \mathcal{A}[a_2]\sigma \\ \mathcal{A}[a_1 - a_2]\sigma = \mathcal{A}[a_1]\sigma -_I \mathcal{A}[a_2]\sigma \end{array}$$

- *inductive /compositional* semantics: defined over the structure
- Caution: distinguish $*$ and $*_I$, $+$ and $+_I$, $-$ and $-_I$

- Boolean expressions: for each state, a value in $\mathbb{B}$

$$\mathcal{B} : \textbf{Bexp} \to (\textbf{State} \to \mathbb{B})$$

Remark Expressions can also be defined in an operational way. □

Semantic functions (ctd): some examples/exercises

Example (Semantic function for digits in base 2)

- Define numerals in base 2 (inductively)
- Give them a compositional semantics

$$n ::= 0 \mid 1 \mid n0 \mid n1$$

$$\begin{array}{rcl} \mathcal{N}(0) & = & 0 \\ \mathcal{N}(1) & = & 1 \\ \mathcal{N}(n0) & = & 2 * \mathcal{N}(n) \\ \mathcal{N}(n1) & = & 2 * \mathcal{N}(n) + 1 \end{array}$$

Semantic functions (ctd): some examples/exercises

Example (Semantic function for Boolean expressions)

Define an inductive semantics for Boolean expressions.

$$\mathcal{B} : \textbf{Bexp} \to (\textbf{State} \to \mathbb{B})$$

$$\begin{aligned}
\mathcal{B}[\mathsf{true}]\sigma &= \textbf{tt} \\
\mathcal{B}[\mathsf{false}]\sigma &= \textbf{ff} \\
\mathcal{B}[\neg b]\sigma &= \neg_{\mathbb{B}} \mathcal{B}[b]\sigma \\
\mathcal{B}[a_1 = a_2]\sigma &= \mathcal{A}[a_1]\sigma =_I \mathcal{A}[a_2]\sigma \\
\mathcal{B}[a_1 \leq a_2]\sigma &= \mathcal{A}[a_1]\sigma \leq_I \mathcal{A}[a_2]\sigma \\
\mathcal{B}[b_1 \wedge b_2]\sigma &= \mathcal{B}[b_1]\sigma \wedge_{\mathbb{B}} \mathcal{B}[b_2]\sigma
\end{aligned}$$

Semantic functions (ctd): some examples/exercises

Example (Negative integers)

We add $-a$ as a construct for arithmetical expressions.

- Extend the semantic function of arithmetical expressions (semantics of artihmetical expressions should remain compositional).

We have two possible solutions:

- $\mathcal{A}[-a]\sigma = 0 -_I \mathcal{A}[a]\sigma$ (preserves compositionality),
- $\mathcal{A}[-a]\sigma = \mathcal{A}[0 - a]\sigma$ (does not preserve compositionality).

Outline

Syntax of Language **While**

Semantics of Expressions in Language **While**

(Natural) Operational Semantics of Language **While**

Semantic functions

- Statements:

$$\mathcal{S} : \textbf{Stm} \rightarrow (\textbf{State} \overset{part.}{\longrightarrow} \textbf{State})$$

Function $\mathcal{S}$ gives the *meaning* of a statement S as a partial function from **State** to **State**.

Question: why is it a **partial** function?

Various semantic styles

- Axiomatic semantics allows to prove program properties (later in the course).
- Denotational semantics describes the effect of program execution (from a given state), *without telling how* the program is executed (later in the course).
- **Operational semantics** tells **how a program is executed**
  $\hookrightarrow$ It helps to write interpreters or code generators

Another important feature is *compositionality*: the semantics of a compound program is a function of the semantics of its components.

Operational Semantics

An operational semantics defines a transition system

Definition (Transition System)

A transition system is given by $(\Gamma, T, \rightarrow)$, where:

- Γ is the configuration set
- $T \subseteq \Gamma$ is the set of final configurations
- $\rightarrow \subseteq \Gamma \times \Gamma$ is the transition relation

Example (Transition System)

Execution of a DFA (defined on Σ) can be defined as a transition system:

- $\Gamma = Q \times \Sigma^*$
  - Q is the set of states of the DFA
  - Σ^* is the set of finite-words over Σ

  Configuration (q, w) means: the DFA is in state q and w is the remaining sequence to be read
- $T = \{(q, \epsilon) \mid q \in Q\}$
- $\rightarrow (q, a \cdot w) = (q', w)$ s.t. q' is the state reached in the DFA by firing a in state q

Natural Operational Semantics

- Defines the relationship between initial and final steps of an execution.
- This relationship is specified for each statement, w.r.t. a current **State**.

Transition system for Natural Operational Semantics

- Configurations: $\Gamma \subseteq \mathbf{Stm} \times \mathbf{State} \cup \mathbf{State}$.
- Transition relation: $(S, \sigma) \rightarrow \sigma'$
  - "The execution of S from σ *terminates* in state σ'"
  - Goal: to describe how the result of a program execution is obtained

Natural semantics: about rules

Semantics is defined by an inference system: axioms and rules.

Rules of the form:

$$\frac{(S_1, \sigma_1) \to \sigma_1' \quad (S_2, \sigma_2) \to \sigma_2' \quad \ldots \quad (S_n, \sigma_n) \to \sigma_n'}{(S, \sigma) \to \sigma'} \text{ if} \cdots$$

- $S_1, S_2, \ldots, S_n$ are immediate constituents of S, i.e., S is "built on" $S_1, \ldots, S_n$ or statements built from immediate constituents,
- $(S_1, \sigma_1) \to \sigma_1', (S_2, \sigma_2) \to \sigma_2', \ldots, (S_n, \sigma_n) \to \sigma_n'$ are called premises of the rule ; if $n = 0$, the rule is called axiom (schema) and the solid line is omitted,
- $(S, \sigma) \to \sigma'$ is the conclusion of the rule
- a rule may also have a condition (if $\cdots$).

Natural semantics: axioms and rules

Axioms

$$(x := a, \sigma) \to \sigma[x \mapsto \mathcal{A}[a]\sigma]$$

$$(\mathsf{skip}, \sigma) \to \sigma$$

Rule for Sequence Composition

$$\frac{(S_1, \sigma) \to \sigma' \quad (S_2, \sigma') \to \sigma''}{(S_1; S_2, \sigma) \to \sigma''}$$

Natural semantics: axioms and rules (ctd)

Rule for Conditional statements

$$\frac{(S_1, \sigma) \to \sigma'}{(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, \sigma) \to \sigma'} \quad \mathit{If}\, \mathcal{B}[b]\sigma = \mathbf{tt}$$

$$\frac{(S_2, \sigma) \to \sigma'}{(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, \sigma) \to \sigma'} \quad \mathit{If}\, \mathcal{B}[b]\sigma = \mathbf{ff}$$

Rule for While statements

$$\frac{(S, \sigma) \to \sigma' \quad (\text{while } b \text{ do } S \text{ od }, \sigma') \to \sigma''}{(\text{while } b \text{ do } S \text{ od }, \sigma) \to \sigma''} \quad \mathit{If}\, \mathcal{B}[b]\sigma = \mathbf{tt}$$

$$(\text{while } b \text{ do } S \text{ od }, \sigma) \to \sigma \qquad \mathit{If}\, \mathcal{B}[b]\sigma = \mathbf{ff}$$

Derivation tree

Represents/Describes an execution from a statement S and a state σ to a state σ'.

- Leaves correspond to (instantiation of) axioms
- Internal nodes corresponds to (instantiation of) inference rules.
- the root is $(S, \sigma) \rightarrow \sigma'$ (it is common to have the root at the bottom rather than at the top when drawing a derivation tree).

Example (Derivation Tree)

Consider $\sigma \in$ **State**:

$$\frac{(x := 1, \sigma) \rightarrow \sigma[x \mapsto 1] \quad (y := 5, \sigma[x \mapsto 1]) \rightarrow \sigma[x \mapsto 1][y \mapsto 5]}{(x := 1; y := 5, \sigma) \rightarrow \sigma[x \mapsto 1, y \mapsto 5]}$$

Construction of derivation tree

Given,

- A statement (abstract tree) S,
- a state σ,

we want to find σ', if it exists such that $(S, \sigma) \to \sigma'$.

The method tries to construct the tree from the root upwards $(S, \sigma) \to \sigma'$, starting from an axiom or a rule with a conclusion where the left-hand side "matches" the configuration (S, σ).

There are two cases :

- if it is an axiom and the condition of the axiom holds, then we can compute the final state and the construction of the derivation tree is completed,
- if it is a rule, then the next step is to try to construct a derivation tree for all the premises of the rule.

Construction of derivation tree: example

Let

- $S = (z := x; x := y); y := z$
- $\sigma_0 = [x \mapsto 2, y \mapsto 4, z \mapsto 0]$

Applying axioms and rules we obtain:

$$\dfrac{\dfrac{(z := x, \sigma_0) \to \sigma_1 \quad (x := y, \sigma_1) \to \sigma_2}{(z := x; x := y, \sigma_0) \to \sigma_2} \qquad (y := z, \sigma_2) \to \sigma_3}{((z := x; x := y); y := z, \sigma_0) \to \sigma_3}$$

with,

- $\sigma_1 = [x \mapsto 2, y \mapsto 4, z \mapsto 2]$,
- $\sigma_2 = [x \mapsto 4, y \mapsto 4, z \mapsto 2]$,
- $\sigma_3 = [x \mapsto 4, y \mapsto 2, z \mapsto 2]$.

Example

Let

- S_0 : while $x > 1$ do $y := y * x; x := x - 1$ od
- S_1 : $y := y * x; x := x - 1$
- $\sigma_{31} = [x \mapsto 3, y \mapsto 1]$

We try to find $\sigma?$ such that $(S_0, \sigma_{31}) \to \sigma?$.

$$\frac{T_1 \quad T_2}{(S_0, \sigma_{31}) \to \sigma?}$$

Construction of T_1

$$\frac{(y := y * x, \sigma_{31}) \to \sigma_{33} \quad (x := x - 1, \sigma_{33}) \to \sigma_{23}}{(S_1, \sigma_{31}) \to \sigma_{23}}$$

Construction of T_2

$$\frac{T_3 \quad T_4}{(S_0, \sigma_{23}) \to \sigma?}$$

Construction of T_3

$$\frac{(y := y * x, \sigma_{23}) \to \sigma_{26} \quad (x := x - 1, \sigma_{26}) \to \sigma_{16}}{(S_1, \sigma_{23}) \to \sigma_{16}}$$

Construction of T_4

(while $x > 1$ do $y := y * x; x := x - 1$ od $, \sigma_{16}) \to \sigma_{16}$

Example cont.

The construction of derivation tree stops when we find σ_{16} because in this state, $\sigma_{16}(x) = 1$ and $\mathcal{B}[x > 1]_{\sigma_{16}} = \mathbf{ff}$.

Finally, we find $\sigma? = \sigma_{16}$ and the derivation tree is:

$$\frac{T_1 \quad \dfrac{T_3 \quad (S_0, \sigma_{16}) \to \sigma_{16}}{(S_0, \sigma_{23}) \to \sigma_{16}}}{(S_0, \sigma_{31}) \to \sigma_{16}}$$

Example (Derivation trees)

What is the semantics of:

1. $x := 2; \text{if } x > 0 \text{ then } x := x + 1 \text{ else } x := x - 1 \text{ fi}$
2. $x := 2; \text{while } x > 0 \;\; \text{do } x := x - 1 \text{ od}$
3. $x := 2; \text{while } x > 0 \;\; \text{do } x := x + 1 \text{ od}$

Terminology

Consider a statement S and a state σ.

Definition (Termination/Looping)

The execution of S on σ

- terminates iff there is a state σ' s.t. $(S, \sigma) \to \sigma'$;
- loops iff there is no state σ' s.t. $(S, \sigma) \to \sigma'$.

Statement S

- always terminates iff the execution of S terminates on any state σ;
- always loops iff the execution of S loops on any state σ.

Another iterative construct

Example (Adding constructs to **While**)

We add the two following iterative statements to language **While**.
Give their corresponding semantic rules.

$$
\begin{array}{rcl}
S & ::= & \text{iterate } n \text{ times } S \\
 & & \mid \text{for } x{:=}a \text{ to } a \text{ loop } S
\end{array}
$$

Natural semantics is deterministic

Theorem

For all statements $S \in \mathbf{Stm}$, for all states σ, σ' and σ'':

1. *If $(S, \sigma) \to \sigma'$ and $(S, \sigma) \to \sigma''$ then $\sigma' = \sigma''$.*
2. *If $(S, \sigma) \to \sigma'$, then there does not exist any infinite derivation tree.*

Proof.

By induction on the structure of the derivation tree.
We will do it during the exercise session. □

Semantic function $\mathcal{S}_{ns}$

Definition (The semantic function $\mathcal{S}_{ns}$)

$$\mathcal{S}_{ns}[S]\sigma = \begin{cases} \sigma' & \text{if } (S, \sigma) \to \sigma', \\ \text{undef} & \text{otherwise,} \end{cases}$$

(because of looping executions, it is a partial function).

Example (Applying the Semantic function)

- $\mathcal{S}_{ns}[x := 2][x \mapsto 0] = [x \mapsto 2]$ because $(x := 2, [x \mapsto 0]) \to [x \mapsto 2]$.
- $\mathcal{S}_{ns}[\text{while true do skip od }]\sigma = \text{undef}$, for any $\sigma \in \textbf{State}$.

Summary

Summary of NOS of language **While**

Definition of the While programming language:

- Syntax (inductive definitions of the syntactic categories).
- Semantics for arithmetical and Boolean expressions.
- Semantics for statements.
- Termination of programs.