

Semantics and Analysis of Programs for Security

Operational semantics

Thomas Jensen

PAS/SdL, M2R Informatique, U. Rennes 1

October 2011



Outline

1 Introduction to semantics

2 Operational semantics



Why semantics?

Goal: give “meaning” to programs.

Why?

- ▶ Avoid ambiguities
- ▶ Provide a basis for reasoning about programs (correctness proofs, program transformations, static analyses)

For whom?

- ▶ Language designers
- ▶ Language implementors
- ▶ Programmers

How to define a language

Syntax - Grammars

- ▶ Define the programming language

Static semantics

- ▶ Define the set of programs to be considered meaningful.

Dynamic semantics

- ▶ Define program execution.

Various formalisms for dynamic semantics:

- ▶ operational (transition systems, abstract machines)
- ▶ denotational (sets, functions, relations)
- ▶ axiomatic (program logics)

Operational semantics

Consider the program

```
y:=1;
while not(x=1) do (y:=x*y; x:=x-1)
```

“ y is initialized to 1, then one tests if x is equal to 1. If ‘yes’, then stop, if ‘no’, y gets the value of multiplying the old value of y and of x and x is decremented. Then one tests if . . . ”

Three styles:

- ▶ Abstract machines
- ▶ Structural operational semantics (“small-step” semantics)
- ▶ Natural semantics (“big-step” semantics)

Structural: syntax-directed definition in the shape of an inference system.



Denotational semantics

Now describe the program in terms of input-output behaviour:

```
y:=1;
while not(x=1) do (y:=x*y; x:=x-1)
```

“ The program is a function that takes an input state s_a and returns a state s_r . The result state s_r is identical to s_a except that it maps x to the value 1 and y to n! where n is the value of x in s_a . ”

Compositional definition of functions, relations . . . following the syntax.

Mathematical objects that abstract (hides) details of the implementation (focus on **what** is being computed, not **how**).



Axiomatic semantics

```
y:=1;
while not(x=1) do (y:=x*y; x:=x-1)
```

“ If $x = n$ before the execution (pre-condition) then $y = n!$ after execution of the program (if it terminates)”

Two varieties

- ▶ Partial correctness.
- ▶ Total correctness.

Syntax-directed definition of a program logic.

High-level logical formulae (pre- and post-conditions) for proving properties.



The *While* language

Syntax

- ▶ Integers ($n \in \mathbf{Num}$)
- ▶ Variables ($x \in \mathbf{Var}$)
- ▶ Arithmetic expressions ($a \in \mathbf{Aexp}$)

$$a ::= n \mid x \mid a_1 + a_2 \mid \dots$$
- ▶ Boolean expressions ($b \in \mathbf{Bexp}$)

$$b ::= \mathbf{true} \mid \mathbf{false} \mid a_1 = a_2 \mid$$

$$\mathbf{not} \ b \mid b_1 \mathbf{and} \ b_2 \mid \dots$$
- ▶ Commands ($S \in \mathbf{Stm}$)

$$S ::= x := a \mid \mathbf{skip} \mid S_1 ; S_2 \mid$$

$$\mathbf{if} \ b \mathbf{then} \ S_1 \mathbf{else} \ S_2 \mid$$

$$\mathbf{while} \ b \mathbf{do} \ S$$



Semantics of *While* expressions

- ▶ We suppose given a function $\mathcal{N} : \mathbf{Num} \rightarrow \mathbb{Z}$
- ▶ The state (the machine's memory) is represented by a function

$$s \in \mathbf{State} = \mathbf{Var} \rightarrow \mathbb{Z}$$

- ▶ Reading a value:
 $(s \ x) : \text{the value of } x \text{ in state } s$
- ▶ Writing a value:
 $s' = s[y \mapsto v] : \text{the state } s \text{ in which the value of } y \text{ is replaced by } v$

Arithmetic Expressions

First example of a denotational semantics:

$$\begin{aligned} \mathcal{A} : \mathbf{Aexp} &\rightarrow \mathbf{State} \rightarrow \mathbb{Z} \\ \mathcal{A}[[n]]s &= \mathcal{N}[[n]] \\ \mathcal{A}[[x]]s &= s \ x \\ \mathcal{A}[[a_1 + a_2]]s &= \mathcal{A}[[a_1]]s + \mathcal{A}[[a_2]]s \end{aligned}$$

The semantics is **compositional**: *Define the meaning of each syntactic construction from the meaning of its constituent parts.*

Proof technique: induction on the structure of expressions.

To prove a property of all arithmetic expressions:

- 1 Show the property true for each atomic expression.
- 2 Show the property true for each composite expression, **under the hypothesis** that it is true for its constituent parts.

Boolean expressions

$$\mathcal{B} : \mathbf{Bexp} \rightarrow \mathbf{State} \rightarrow \mathbb{B}$$

$$\mathcal{B}[\mathbf{true}]_s = \mathbf{tt}$$

$$\mathcal{B}[\mathbf{false}]_s = \mathbf{ff}$$

$$\mathcal{B}[a_1 = a_2]_s = \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[a_1]_s = \mathcal{A}[a_2]_s \\ \mathbf{ff} & \text{if } \mathcal{A}[a_1]_s \neq \mathcal{A}[a_2]_s \end{cases}$$

$$\mathcal{B}[\mathbf{not } b]_s = \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[b]_s = \mathbf{ff} \\ \mathbf{ff} & \text{if } \mathcal{B}[b]_s = \mathbf{tt} \end{cases}$$

$$\mathcal{B}[b_1 \mathbf{and } b_2]_s = \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[b_1]_s = \mathbf{tt} \\ & \text{and } \mathcal{B}[b_2]_s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[b_1]_s = \mathbf{ff} \\ & \text{or } \mathcal{B}[b_2]_s = \mathbf{ff} \end{cases}$$

Exercises

Exercise

Suppose $(s \ x = 3)$. Show that $\mathcal{B}[\mathbf{not}(x = 1)]_s = \mathbf{tt}$

Exercise

We extend the language **Bexp** with the construction b_1 or b_2 .

- ▶ Extend the semantic function \mathcal{B} to give a compositional semantics for this construction.
- ▶ Prove that for all \bar{b} belonging to the extended language there exists a b belonging to the original language such that:

$$\mathcal{B}[b] = \mathcal{B}[\bar{b}]$$

References

- ▶ H.R. Nielson and F. Nielson, *Semantics with Applications - A Formal Introduction*, Wiley 1992. (chapter 1)
 - ▶ http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html
- ▶ G. Winskel, *The Formal Semantics of Programming Languages* MIT Press, 1993
- ▶ D. Schmidt, *Denotational semantics*, Allyn and Bacon, 1986

Outline

1 Introduction to semantics

2 Operational semantics

Operational semantics

References

- ▶ H.R. Nielson and F. Nielson, *Semantics with Applications - A Formal Introduction*, Wiley 1992. (chapter 2)
- ▶ G. Plotkin, *A Structural Approach to Operational Semantics*, Technical Report, Aarhus University, 1981.
- ▶ G. Kahn, *Natural Semantics*, In Proc. of the Symposium on Theoretical Aspects of Computer Science, LNCS 247, pp. 22–39, Springer-Verlag, 1987.

Operational semantics

We consider the imperative language *While*:

$$S ::= x := a \mid \text{skip} \mid S_1 ; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S$$

- ▶ Describe how the execution of *While* programs is done.
- ▶ *Structural Operational Semantics* (SOS)
“small-step semantics”
 - ▶ What are the individual evaluation steps?
 - ▶ Proof by induction on the length (number of steps) of an execution.
- ▶ *Natural semantics* (NS)
“big-step semantics”
 - ▶ How to obtain the final result of a computation
 - ▶ Proof by induction on the structure of derivation trees.
- ▶ Semantic descriptions based on the syntax/grammar of the language
- ▶ Descriptions take the form of **transition systems**.

Transition systems

A transition system is a triple $(\Gamma, T, \rightsquigarrow)$ where

- ▶ Γ is a set of **configurations**
- ▶ $T \subseteq \Gamma$ a set of **final** configurations (normal forms)
- ▶ $\rightsquigarrow \subseteq \Gamma \times \Gamma$ a transition relation

The transition relation will be defined by an inference system, i.e. a set of axioms

$$\gamma \rightsquigarrow \gamma' \quad \text{if } \dots$$

and a set of inference rules

$$\frac{\gamma_0 \rightsquigarrow \gamma'_0 \quad \dots \quad \gamma_i \rightsquigarrow \gamma'_i}{\gamma_j \rightsquigarrow \gamma'_j} \quad \text{if } \dots$$

Transition systems (2)

The transition systems will be of the form $(\Gamma, T, \rightsquigarrow)$ with

- ▶ $\Gamma = \{(S, s) \mid S \in \mathbf{Stm}, s \in \mathbf{State}\} \cup \mathbf{State}$
- ▶ $T = \mathbf{State}$

So, either $(S, s) \rightsquigarrow (S', s')$ or $(S, s) \rightsquigarrow s'$

By default, the systems are

- ▶ deterministic and
- ▶ without *blocking* configurations.

We'll define two transition relations

\rightarrow : small-step evaluation

\Downarrow : big-step evaluation

Structural operational semantics (SOS)

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

$$(\text{skip}, s) \rightarrow s$$

$$\frac{(S_1, s) \rightarrow s'}{(S_1 ; S_2, s) \rightarrow (S_2, s')}$$

$$\frac{(S_1, s) \rightarrow (S'_1, s')}{(S_1 ; S_2, s) \rightarrow (S'_1 ; S_2, s')}$$

$$(\text{if } b \text{ then } S_1 \text{ else } S_2, s) \rightarrow (S_1, s) \quad \text{if } \mathcal{B}[[b]]s = \mathbf{tt}$$

$$(\text{if } b \text{ then } S_1 \text{ else } S_2, s) \rightarrow (S_2, s) \quad \text{if } \mathcal{B}[[b]]s = \mathbf{ff}$$

$$(\text{while } b \text{ do } S, s) \rightarrow (\text{if } b \text{ then } (S ; \text{while } b \text{ do } S) \text{ else skip}, s)$$


Exercises

Exercise

Apply the inference rules to the following commands and initial states:

- ▶ $(z := x; x := y); y := z$
with $s_0 x = 5, s_0 y = 7, s_0 z = 0$
- ▶ $y:=1; \text{while not}(x=1) \text{ do } (y:=x*y; x:=x-1)$
with $s_0 x = 3$

Exercise

The While language is extended with the construction `repeat S until b`.

- ▶ Extend the SOS accordingly.

Exercise

- ▶ Give an SOS to the arithmetic expressions (**Aexp**) of the While language.



Some notation

The **transition relations** is of form $(S, s) \rightarrow \gamma$

- ▶ $\gamma = (S', s')$ execution of S has not terminated (there is (S', s') left to execute)
- ▶ $\gamma = s'$ execution of S has terminated (s' is the final configuration)

An **execution** is modeled by a sequence of derivations

$$\gamma_0, \dots, \gamma_p, \dots \quad \text{with } \gamma_i = (S_i, s_i) \text{ and } \gamma_i \rightarrow \gamma_{i+1}$$

\rightarrow^* : the relation “reduces in a finite number of transitions” (reflexive and transitive closure of \rightarrow).

\rightarrow^+ : the relation “reduces in a finite non-zero number of transitions” (transitive closure of \rightarrow)

\rightarrow^i : the relation “reduces in exactly i transitions”.



Some definitions

- 1 Execution of (S, s) **loops** iff there exists an infinite derivation sequence starting from (S, s)
- 2 Execution of (S, s) **terminates** iff there exists a finite derivation sequence starting from (S, s)
- 3 S_1 is **semantically equivalent** to S_2 iff

$$\forall s (S_1, s) \rightarrow^* s' \Leftrightarrow (S_2, s) \rightarrow^* s'$$

- 4 The **meaning** of a command is given by the semantic function

$$\mathcal{S}_{s0s} : \mathbf{Stm} \rightarrow \mathbf{State} \rightarrow \mathbf{State}$$

defined by

$$\mathcal{S}_{s0s} \llbracket S \rrbracket s = \begin{cases} s' & \text{if } (S, s) \rightarrow^* s' \\ \mathbf{undef} & \text{otherwise} \end{cases}$$



Proof technique associated with SOS

Induction on the length of derivation sequences.

- 1 Prove the property for sequences of length 0
- 2 Prove the property for sequences of length $k + 1$, under the hypothesis that it is true for sequences of length k and less.

Exercises

Exercise

Prove that if $(S_1, s) \rightarrow^k s'$ then $(S_1 ; S_2, s) \rightarrow^k (S_2, s')$

Exercise

Prove that $(S_1 ; S_2, s) \rightarrow^* (S_2, s')$ does **not** imply $(S_1, s) \rightarrow^* s'$

Exercise

Prove that if $(S_1 ; S_2, s) \rightarrow^k s''$ then there exists s' , k_1 and k_2 such that $(S_1, s) \rightarrow^{k_1} s'$, $(S_2, s') \rightarrow^{k_2} s''$ and $k = k_1 + k_2$.

Exercise

Prove that $S_1 ; (S_2 ; S_3)$ and $(S_1 ; S_2) ; S_3$ are semantically equivalent.

Natural semantics (NS)

- $(x := a, s) \Downarrow s[x \mapsto \mathcal{A}[[a]]s]$
- $(\text{skip}, s) \Downarrow s$
- $$\frac{(S_1, s) \Downarrow s' \quad (S_2, s') \Downarrow s''}{(S_1 ; S_2, s) \Downarrow s''}$$
- $$\frac{(S_1, s) \Downarrow s'}{(\text{if } b \text{ then } S_1 \text{ else } S_2, s) \Downarrow s'} \quad \text{if } \mathcal{B}[[b]]s = \mathbf{tt}$$
- $$\frac{(S_2, s) \Downarrow s'}{(\text{if } b \text{ then } S_1 \text{ else } S_2, s) \Downarrow s'} \quad \text{if } \mathcal{B}[[b]]s = \mathbf{ff}$$
- $$\frac{(S, s) \Downarrow s' \quad (\text{while } b \text{ do } S, s') \Downarrow s''}{(\text{while } b \text{ do } S, s) \Downarrow s''} \quad \text{if } \mathcal{B}[[b]]s = \mathbf{tt}$$
- $(\text{while } b \text{ do } S, s) \Downarrow s \quad \text{if } \mathcal{B}[[b]]s = \mathbf{ff}$



Exercises

Exercise

Apply the NS inference rules to the following program and start state:

- ▶ $(z := x; x := y); y := z$
with $s_0 x = 5, s_0 y = 7, s_0 z = 0$
- ▶ $y:=1; \text{while not}(x=1) \text{ do } (y:=x*y; x:=x-1)$
with $s_0 x = 3$

Exercise

The While language is extended with the construction `repeat S until b`

Extend the Natural Semantics accordingly.



Definitions

An execution is represented by a **derivation tree** of $(S, s) \Downarrow s'$

- ① Execution of (S, s) terminates iff there exists a state s' such that $(S, s) \Downarrow s'$
- ② Execution of (S, s) loops iff there is no state s' such that $(S, s) \Downarrow s'$
- ③ S_1 is semantically equivalent to S_2 iff

$$\forall s (S_1, s) \Downarrow s' \Leftrightarrow (S_2, s) \Downarrow s'$$

- ④ The *meaning* of a command is given by the semantic function

$$\mathcal{S}_{nat} : \mathbf{Stm} \rightarrow \mathbf{State} \rightarrow \mathbf{State}$$

defined by

$$\mathcal{S}_{nat} \llbracket S \rrbracket s = \begin{cases} s' & \text{if } (S, s) \Downarrow s' \\ \mathbf{undef} & \text{otherwise} \end{cases}$$

Proof technique associated with NS

Induction principle for derivation trees.

- ① Prove the property for the leaves (the axioms of the inference system)
- ② For each inference rule, prove the property for the conclusion of the rule, **under the hypothesis** that the property holds for each of the premises.

Exercises

Exercise

Check that the Natural Semantics of While is deterministic. Formally, check the property

$$\text{if } (S, s) \Downarrow s' \text{ and } (S, s) \Downarrow s'' \text{ then } s' = s''$$

Exercise

Prove that $S_1 ; (S_2 ; S_3)$ et $(S_1 ; S_2) ; S_3$ are semantically equivalent.

Exercise

Prove that

while b **do** S

and

if b **then** $(S ; \text{while } b \text{ do } S)$ **else skip**

are semantically equivalent.



An equivalence of two semantics

Theorem

$$\mathcal{S}_{nat} = \mathcal{S}_{sos}$$

The theorem is a direct consequence of the following two lemmas:

Lemma

For all commands S and state s , we have

$$(S, s) \Downarrow s' \Rightarrow (S, s) \rightarrow^* s'$$

Lemma

For all commands S and state s , we have

$$(S, s) \rightarrow^k s' \Rightarrow (S, s) \Downarrow s'$$



Proof of Lemma 1

By induction on the derivation tree of $(S, s) \Downarrow s'$.

- ▶ **Case :** $(x := a, s) \Downarrow s[x \mapsto \mathcal{A}[[a]]s]$

we have $(x := a, s) \rightarrow s[x \mapsto \mathcal{A}[[a]]s]$

- ▶ **Case :**

$$\frac{(S_1, s) \Downarrow s' \quad (S_2, s') \Downarrow s''}{(S_1 ; S_2, s) \Downarrow s''}$$

$(S_1, s) \rightarrow^* s'$ and $(S_2, s') \rightarrow^* s''$ *(induction hypothesis)*

$(S_1 ; S_2, s) \rightarrow^* (S_2, s')$ *(Exercise)*

$(S_1 ; S_2, s) \rightarrow^* s''$

Proof of Lemma 1

- ▶ **Case :**

$$\frac{(S, s) \Downarrow s' \quad (\text{while } b \text{ do } S, s') \Downarrow s''}{(\text{while } b \text{ do } S, s) \Downarrow s''} \quad \mathcal{B}[[b]]s = \mathbf{tt}$$

We have the following derivation

$(\text{while } b \text{ do } S, s)$
 $\rightarrow (\text{if } b \text{ then } (S ; \text{while } b \text{ do } S) \text{ else skip}, s)$
 $\rightarrow (S ; \text{while } b \text{ do } S, s)$

and since

$(S, s) \rightarrow^* s'$ and $(\text{while } b \text{ do } S, s') \rightarrow^* s''$ *(induction hypothesis)*

we have $(S ; \text{while } b \text{ do } S, s) \rightarrow^* s''$ *(Exercise)*

- ▶ same idea for the other cases

Proof of Lemma 2

Induction on the length of the derivation sequence of $(S, s) \rightarrow^k s'$.

If $k = 0$ then trivial.

Otherwise, suppose the Lemma holds for $k \leq k_0$ and prove it for a derivation of length $k_0 + 1$.

- ▶ **Case** : $x := a$ trivial ($k_0 = 0$)
- ▶ **Case** : $(S_1 ; S_2, s) \rightarrow^{k_0+1} s''$
 $(S_1, s) \rightarrow^{k_1} s'$ and $(S_2, s') \rightarrow^{k_2} s''$ with $k_1 + k_2 = k_0 + 1$ (Exercise)
 $(S_1, s) \Downarrow s'$ and $(S_2, s') \Downarrow s''$ (induction hypothesis)
 $(S_1 ; S_2, s) \Downarrow s''$

Proof of Lemma 2

- ▶ **Case** : $(\text{while } b \text{ do } S, s)$
 $\rightarrow (\text{if } b \text{ then } (S ; \text{while } b \text{ do } S) \text{ else skip}, s)$
 $\rightarrow^{k_0} s''$
 $(\text{if } b \text{ then } (S ; \text{while } b \text{ do } S) \text{ else skip}, s) \Downarrow s''$
(induction hypothesis)
 and $(\text{while } b \text{ do } S, s) \Downarrow s''$ (this was an Exercise)
- ▶ same technique for the other cases

Extension of *While* (1) : termination operator

We extend the *While* language with the operator `abort` .

Informal description: *the operator halts execution of the program.*

One way of modeling `abort` : semantic rules stay unchanged and the configurations of form (abort, s) are blocking.

- ▶ SOS : `abort` is different from `skip` and from `while true do skip` .
- ▶ NS : `abort` is different from `skip` but equivalent to `while true do skip` .

One solution that allows the NS to distinguish between termination by `abort` and non-termination:

$$(\text{abort}, s) \Downarrow s_{\text{abort}}$$

This requires to modify all other rules to take the special state s_{abort} into account.



Extensions to *While* (2) : non-deterministic choice

Extend *While* with the non-deterministic choice operator $S_1 \square S_2$.

Informal description: *choose non-deterministically to execute one of S_1 and S_2 .*
The language now becomes non-deterministic.

- ▶ Formalisation as a SOS

$$\begin{aligned} (S_1 \square S_2, s) &\rightarrow (S_1, s) \\ (S_1 \square S_2, s) &\rightarrow (S_2, s) \end{aligned}$$

- ▶ Formalisation as a NS

$$\frac{\frac{(S_1, s) \Downarrow s'}{(S_1 \square S_2, s) \Downarrow s'}}{(S_2, s) \Downarrow s'}}{(S_1 \square S_2, s) \Downarrow s'}$$

The SOS can choose an expression that loops.

Natural semantics will always choose to eliminate non-termination as a possible behaviour.

Ex: $(x := 1) \square (\text{while true do skip})$



Extension of *While* (3) : parallel composition

Extend the *While* language with the parallel operator $S_1 \text{ par } S_2$.

Intuition: *execution of S_1 and S_2 are inter-leaved*. The language becomes non-deterministic.

- ▶ Formalisation as a SOS

$$\frac{(S_1, s) \rightarrow (S'_1, s')}{(S_1 \text{ par } S_2, s) \rightarrow (S'_1 \text{ par } S_2, s')}$$

$$\frac{(S_1, s) \rightarrow s'}{(S_1 \text{ par } S_2, s) \rightarrow (S_2, s')}$$

$$\frac{(S_2, s) \rightarrow (S'_2, s')}{(S_1 \text{ par } S_2, s) \rightarrow (S_1 \text{ par } S'_2, s')}$$

$$\frac{(S_2, s) \rightarrow s'}{(S_1 \text{ par } S_2, s) \rightarrow (S_1, s')}$$

- ▶ Formalisation as a NS

?

Interleaving of executions is not expressible as an extension of the natural Semantics.

NS of a functional language:

Language $\mathcal{L}_{\mathbb{B}}$ of expressions over boolean lists:

$$\begin{aligned} L & ::= \text{nil} \mid \text{cons } B L \mid \text{tl } L \\ & \quad \mid \text{if } B \text{ then } L_1 \text{ else } L_2 \\ B & ::= \text{true} \mid \text{false} \mid \text{null } L \mid \text{hd } L \end{aligned}$$

Semantic domains :

$$\mathbb{B} = \{\mathbf{tt}, \mathbf{ff}\} \quad \mathbb{L} = \{[]\} \cup \{b:l \mid b \in \mathbb{B}, l \in \mathbb{L}\}$$

Define a Natural Semantics for $\mathcal{L}_{\mathbb{B}}$: