

Operational Semantics

Benoît Montagu / Thomas Jensen

<https://epicure.gitlabpages.inria.fr/m2-sos/>

SOS, Master Recherche Science Informatique, Univ Rennes

2024–2025

Introduction

Programming language semantics

- ▶ What is a semantics?
 - ▶ A definition of the meaning of programs: what/how do they compute?
 - ▶ A *formal* specification of programs' behaviour
 - ▶ A “ground truth” users of a language should agree on
- ▶ What do I need a semantics of programs for?
 - ▶ To make sure my program computes what it should compute
 - ▶ To make sure the optimization passes of my compiler are correct
 - ▶ To make sure my static analyser does not miss any bug
 - ▶ To get formal guarantees of a program's behaviour
- ▶ In this course: a quick introduction to this topic

Modelling the behaviour of programs

- ▶ A program semantics is a *model* of what a program computes
- ▶ Every model has its limitation:
 - Infinite behaviours Intermediate steps of computation Call stack
 - Data representation Execution time Micro-architectural details
- ▶ The choice of *what* to model depends on what you want to *guarantee*
- ▶ Example: if I want to reason about the accesses to uninitialized data, there is no need to model the execution time!
- ▶ **Analogy with physics:** how to model the trajectory of an electron?
 - No gravity Newtonian gravity Friction (linear/non-linear)
 - Special relativity (close to light speed) General relativity (massive objects)

Idealising a language

Real life programming languages are **complex**

👉 The specification of the C language: \approx 500 pages

In semantics, we start a small set of features and grow over time

The standard starting point, **imperative programming**:

- ▶ Variables (of type `int`)
- ▶ Assignments of arithmetic expressions (involving variables) to variables
- ▶ Conditionals on boolean expressions derived from variables
- ▶ While statements

We create an **idealised language** that combines these features: `WHILE`

3/33

WHILE: An imperative toy language

The WHILE language

$n \in \mathbf{Num}$	$x \in \mathbf{Var}$	integers and variables
$a \in \mathbf{Aexp}$	$a ::= n \mid x \mid a_1 + a_2 \mid \dots$	arithmetic expressions
$b \in \mathbf{Bexp}$	$b ::= \mathbf{true} \mid \mathbf{false} \mid a_1 = a_2 \mid a_1 > a_2$ $\mid \mathbf{not} \ b \mid b_1 \ \mathbf{and} \ b_2 \mid \dots$	boolean expressions
$c \in \mathbf{Cmd}$	$c ::= x := a \mid \mathbf{skip} \mid c_1 ; c_2$ $\mid \mathbf{if} \ b \ \mathbf{then} \ c_1 \ \mathbf{else} \ c_2 \mid \mathbf{while} \ b \ \mathbf{do} \ c$	commands (i.e., programs)

This definition is called a **BNF grammar**:

- ▶ Different syntactic categories
- ▶ Certain basic categories are assumed: numbers, variables
- ▶ Valid programs are described by **abstract syntax trees** (ASTs)

Example: the factorial program

```
result := 1; while n > 1 do (result := n * result; n := n - 1)
```

4/33

Different semantics of WHILE

There are several ways of expressing the mathematical behaviour of a program, say:
`fact(n) := (result := 1; while (n > 0) do (result := n * result; n := n - 1))`

Informally:

1. `fact` always terminates
2. If `n` is equal to $k \in \mathbb{N}$ before the execution, at the end `result = k!`

Different families of models based on **memory states** $\sigma \in \mathbf{Var} \rightarrow \mathbb{Z}$:

- ▶ **Denotational:** programs become functions on memory states.

$$\llbracket P \rrbracket = \sigma \mapsto \sigma[n \mapsto 1, \mathbf{result} \mapsto \sigma(n)!]$$

- ▶ **Axiomatic:** programs become memory predicate transformers.

$$\{n = k\} P \{n = 1 \wedge \mathbf{result} = k!\}$$

- ▶ **Operational:** programs are executed by a mathematical machine.

$$(P, \sigma) \rightarrow^* (\mathbf{skip}, \sigma[n \mapsto 1, \mathbf{result} \mapsto \sigma(n)!])$$

This course: focus on operational semantics.

5/33

References

- ▶ G. Winskel, *The Formal Semantics of Programming Languages* MIT Press, 1993 (chapters 2 and 3)
- ▶ 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.

6/33

Semantics of expressions

Interpretation of values

Define the sets and functions used to describe the meaning of expressions.

- ▶ Values : integers and booleans

$$\mathbb{Z} \quad \mathbb{B} = \{\mathbf{tt}, \mathbf{ff}\}$$

- ▶ Interpretation function for constants

$$\mathcal{N} \in \mathbf{Num} \rightarrow \mathbb{Z}$$

- ▶ A memory state or environment, σ maps variables to values

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

Reading the value of variable x in σ $\sigma(x)$

Updating σ by setting a new value v for x $\sigma' = \sigma[x \mapsto v]$

- ▶ Semantics of expressions

$$\mathcal{A} \in \mathbf{Aexp} \rightarrow \mathbf{State} \rightarrow \mathbb{Z} \quad \mathcal{B} \in \mathbf{Bexp} \rightarrow \mathbf{State} \rightarrow \mathbb{B}$$

Expressions denote functions from states to integer values.

Notation: $\mathcal{A}(a)(\sigma)$ traditionally written $\mathcal{A}[[a]]\sigma$.

7/33

Arithmetic expressions

Remember: The set of arithmetic expressions is defined **inductively**

$$a \in \mathbf{Aexp} ::= n \mid x \mid a_1 + a_2 \mid \dots$$

$\mathcal{A}[[\cdot]]$ is defined by induction on the definition of **Aexp**, following the structure of expressions.

$$\mathcal{A} \in \mathbf{Aexp} \rightarrow \mathbf{State} \rightarrow \mathbb{Z}$$

$$\mathcal{A}[[n]]\sigma = \mathcal{N}[[n]]$$

$$\mathcal{A}[[x]]\sigma = \sigma(x)$$

$$\mathcal{A}[[a_1 + a_2]]\sigma = \mathcal{A}[[a_1]]\sigma + \mathcal{A}[[a_2]]\sigma$$

Note: $+$ is the syntactic operator, $+$ is the sum operator defined on integers.

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

8/33

Boolean expressions

Similarly, define $\mathcal{B}[\cdot]$ by induction on the definition of **Bexp**.

$$\begin{aligned}\mathcal{B} \in \mathbf{Bexp} &\rightarrow \mathbf{State} \rightarrow \mathbb{B} \\ \mathcal{B}[\mathbf{true}] \sigma &= \mathbf{tt} \\ \mathcal{B}[\mathbf{false}] \sigma &= \mathbf{ff} \\ \mathcal{B}[a_1 = a_2] \sigma &= \mathcal{A}[a_1] \sigma = \mathcal{A}[a_2] \sigma \\ \mathcal{B}[a_1 < a_2] \sigma &= \mathcal{A}[a_1] \sigma < \mathcal{A}[a_2] \sigma \\ \mathcal{B}[\mathbf{not} \ b] \sigma &= \neg(\mathcal{B}[b] \sigma) \\ \mathcal{B}[b_1 \ \mathbf{and} \ b_2] \sigma &= \mathcal{B}[b_1] \sigma \wedge \mathcal{B}[b_2] \sigma\end{aligned}$$

where $\neg, \wedge, =$ are operators defined on booleans and integers.

9/33

Proof technique: Structural induction

The set of arithmetic expressions **Aexp** is defined **inductively**

$$a ::= n \mid x \mid a_1 + a_2 \mid \dots$$

Structural induction

To prove a property \mathcal{P} of all arithmetic expressions:

1. Base cases: show the property for each atomic expression
2. Inductive cases: show the property for each composite expression, **under the hypothesis** that it holds on its constituent parts

Formally, the induction principle for arithmetic expressions is :

$$\left. \begin{array}{l} \forall n \in \mathbf{Num}, \mathcal{P}(n) \\ \wedge \quad \forall x \in \mathbf{Var}, \mathcal{P}(x) \\ \wedge \quad \forall a_1, a_2 \in \mathbf{Aexp}, \mathcal{P}(a_1) \wedge \mathcal{P}(a_2) \Rightarrow \mathcal{P}(a_1 + a_2) \end{array} \right\} \Rightarrow \forall a \in \mathbf{Aexp}, \mathcal{P}(a)$$

Vocabulary: the above $\mathcal{P}(a_1)$ and $\mathcal{P}(a_2)$ are called the **induction hypotheses**.

10/33

Exercises

Exercise 3.1

Let $\sigma \in \mathbf{State}$ and $x \in \mathbf{Var}$ such that $\sigma(x) = 3$. Show that $\mathcal{B}[\mathbf{not}(x = 1)] \sigma = \mathbf{tt}$.

Exercise 3.2 (At home)

We extend the language **Aexp** with the unary minus operator construction $-a$. Extend the semantic function \mathcal{A} to give a compositional semantics for this construction.

Exercise 3.3 (At home)

We extend the language **Bexp** with the construction $b_1 \ \mathbf{or} \ b_2$ (disjunction).

- ▶ Extend the 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 $\mathcal{B}[b] = \mathcal{B}[\bar{b}]$.

11/33

Natural semantics of WHILE

Pre-requisite: inductive relations

Inductively defined relations are usually formalised by a **rule system**.

A rule is of the form:

$$\text{RULENAME} \frac{\text{premise}_1 \quad \dots \quad \text{premise}_n}{\text{conclusion}} \text{ if side conditions}$$

where premise_i and conclusion are elements of the relation being defined.

► It reads:

“If premise_1 and ... premise_n hold,
and if side conditions are satisfied,
then conclusion holds”

- Premises must be, in turn, justified by rules.
- The conclusion holds whenever there is a **finite derivation tree** whose leaves are axioms of the system.
- A rule with no premise is called an axiom.

12/33

Inductive definition: an example

Example: inductive definition of \leq on non-negative integers.

We define $\leq \in \wp(\mathbb{Z} \times \mathbb{Z})$ inductively as follows:

$$\text{LEQZERO} \frac{}{0 \leq n} \text{ if } n \in \mathbb{N} \qquad \text{LEQSUC} \frac{(n-1) \leq (m-1)}{n \leq m}$$

To prove that $2 \leq 5$, you need to produce a finite derivation whose conclusion is $2 \leq 5$:

$$\text{LEQSUC} \frac{\text{LEQSUC} \frac{\text{LEQZERO} \frac{}{0 \leq 3} \text{ } 3 \in \mathbb{N}}{1 \leq 4}}{2 \leq 5}$$

13/33

Natural (or big-step) semantics (NS)

Big-step transition relation for WHILE programs: $\Downarrow \subseteq (\text{Cmd} \times \text{State}) \times \text{State}$

$(c, \sigma_1) \Downarrow \sigma_2$ reads:

“The command c in the starting state σ_1 evaluates to the final state σ_2 ”

$$\begin{array}{l} \text{ASSIG} \frac{}{(x := a, \sigma) \Downarrow \sigma[x \mapsto \mathcal{A}[[a]]\sigma]} \qquad \text{SKIP} \frac{}{(\text{skip}, \sigma) \Downarrow \sigma} \\ \text{SEQ} \frac{(c_1, \sigma) \Downarrow \sigma' \quad (c_2, \sigma') \Downarrow \sigma''}{(c_1 ; c_2, \sigma) \Downarrow \sigma''} \qquad \text{IFT} \frac{(c_1, \sigma) \Downarrow \sigma'}{(\text{if } b \text{ then } c_1 \text{ else } c_2, \sigma) \Downarrow \sigma'} \text{ if } \mathcal{B}[[b]]\sigma = \text{tt} \\ \text{IFE} \frac{(c_2, \sigma) \Downarrow \sigma'}{(\text{if } b \text{ then } c_1 \text{ else } c_2, \sigma) \Downarrow \sigma'} \text{ if } \mathcal{B}[[b]]\sigma = \text{ff} \\ \text{WHI1} \frac{(c, \sigma) \Downarrow \sigma' \quad (\text{while } b \text{ do } c, \sigma') \Downarrow \sigma''}{(\text{while } b \text{ do } c, \sigma) \Downarrow \sigma''} \text{ if } \mathcal{B}[[b]]\sigma = \text{tt} \\ \text{WHI2} \frac{}{(\text{while } b \text{ do } c, \sigma) \Downarrow \sigma} \text{ if } \mathcal{B}[[b]]\sigma = \text{ff} \end{array}$$

14/33

Big-step executions and semantics

A big-step execution of a WHILE command is simply a **derivable** $(c, \sigma) \Downarrow \sigma'$

We say that (c, σ)

- **terminates** iff there exists σ' such that $(c, \sigma) \Downarrow \sigma'$
- **diverges/blocks** iff there is no state σ' such that $(c, \sigma) \Downarrow \sigma'$

Commands c_1 and c_2 are **semantically equivalent** iff

$$\forall \sigma, \sigma'. (c_1, \sigma) \Downarrow \sigma' \Leftrightarrow (c_2, \sigma) \Downarrow \sigma'$$

Remark: any two diverging programs are semantically equivalent

15/33

Exercises

Exercise 4.1 (At home)

The WHILE language is extended with the construction `repeat c until b`, that always executes `c`, and then repeats only if `b` evaluates to `tt`. Extend the NS accordingly.

Exercise 4.2 (At home)

Donner une SN aux expressions arithmétiques (**Aexp**) du langage WHILE

16/33

Induction principle for inductive relations

Induction principle for derivation trees

$$\frac{\frac{\vdots}{P_1} \quad \overline{P_2}}{P}$$

1. Prove the property for the axioms of the rule system
2. For each rule, prove the property for the conclusion of the rule, **under the induction hypothesis** that the property holds for each of the premises, and that side conditions are satisfied.

Intuition: the property is proved:

- ▶ to hold for the leaves of the tree,
- ▶ and to propagate to any possible derivable conclusion.

17/33

Exercises

Exercise 4.3 (At home)

Prove that the NS of WHILE is deterministic, i.e.:
if $(c, \sigma) \Downarrow \sigma_1$ and $(c, \sigma) \Downarrow \sigma_2$, then $\sigma_1 = \sigma_2$.

Exercise 4.4 (At home)

Prove that $c_1 ; (c_2 ; c_3)$ and $(c_1 ; c_2) ; c_3$ are semantically equivalent.

Hint: induction is not necessary here.

Exercise 4.5 (At home)

Prove that `while b do c`
and `if b then (c ; while b do c) else skip`
are semantically equivalent.

Hint: induction is not necessary here.

18/33

Structural Operational Semantics

Small-step semantics

What is it?

A relation that models a **step by step** evaluation, called a **transition system**.

This is opposed to the natural semantics, that is a *big-step* relation: it describes the result of evaluation *directly*.

Benefits of small-step semantics:

- ▶ Model intermediate evaluation states (in addition to the final one)
- ▶ Model more precisely programs with infinite behaviours
- ▶ Distinguish stuck (*i.e.*, faulty) programs from diverging programs

Other names:

- ▶ Structural Operational Semantics (SOS)
- ▶ Reduction semantics

19/33

Transition systems

Transition system

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

- ▶ Γ is a set of **configurations** (states of the machine)
- ▶ $T \subseteq \Gamma$ is a set of **final/terminal** configurations
- ▶ $\rightsquigarrow \subseteq \Gamma \times \Gamma$ is a **transition relation**

A transition system $(\Gamma, T, \rightsquigarrow)$ is:

- ▶ **deterministic** when relation \rightsquigarrow is functional

$$\gamma \rightsquigarrow \gamma_1 \text{ and } \gamma \rightsquigarrow \gamma_2 \text{ implies } \gamma_1 = \gamma_2$$

- ▶ **non-blocking** when relation \rightsquigarrow is total on $\Gamma \setminus T$

$$\text{for all } \gamma \in \Gamma \setminus T, \text{ there exists } \gamma' \text{ such that } \gamma \rightsquigarrow \gamma'$$

The notion of program **execution** will be defined on top of \rightsquigarrow .

20/33

Transition systems for WHILE: configurations

To run a WHILE program, we need a command $c \in \mathbf{Cmd}$, and a state $\sigma \in \mathbf{State}$.

For WHILE, configurations are defined:

- ▶ $\Gamma = \mathbf{Cmd} \times \mathbf{State} = \{(c, \sigma) \mid c \in \mathbf{Cmd}, \sigma \in \mathbf{State}\}$
- ▶ Final configurations: $T = \{\mathbf{skip}\} \times \mathbf{State}$

So, either :

- ▶ $(c, \sigma) \rightsquigarrow (c', \sigma')$
execution of c has not terminated, and (c', σ') is left to execute
- ▶ or $c = \mathbf{skip}$, and the execution of c has terminated

21/33

Small-step transition relation for WHILE

Rule system defining the small-step transition relation.

Precisely: **rule schemas**, to be instantiated on particular commands and states.

$$\text{ASSIG} \frac{}{(x := a, \sigma) \rightarrow (\mathbf{skip}, \sigma[x \mapsto \mathcal{A}[[a]]\sigma])} \quad \text{SEQ1} \frac{}{(\mathbf{skip} ; c, \sigma) \rightarrow (c, \sigma)}$$

$$\text{SEQ2} \frac{(c_1, \sigma) \rightarrow (c'_1, \sigma')}{(c_1 ; c_2, \sigma) \rightarrow (c'_1 ; c_2, \sigma')}$$

$$\text{IFT} \frac{}{(\mathbf{if } b \text{ then } c_1 \text{ else } c_2, \sigma) \rightarrow (c_1, \sigma)} \text{ if } \mathcal{B}[[b]]\sigma = \mathbf{tt}$$

$$\text{IFE} \frac{}{(\mathbf{if } b \text{ then } c_1 \text{ else } c_2, \sigma) \rightarrow (c_2, \sigma)} \text{ if } \mathcal{B}[[b]]\sigma = \mathbf{ff}$$

$$\text{WHI} \frac{}{(\mathbf{while } b \text{ do } c, \sigma) \rightarrow (\mathbf{if } b \text{ then } (c ; \mathbf{while } b \text{ do } c) \text{ else } \mathbf{skip}, \sigma)}$$

22/33

Small-step semantics of WHILE: an example

Let us consider the execution steps of the following programs:

- ▶ skip
- ▶ `(x := 1; y := 2); x := x + y`
- ▶ `x := 0; if (0 <= x) then y := 1 else y := 2`
- ▶ `n := 3; while (0 < n) do n := n - 1`
- ▶ `n := 3; while (0 < n) do n := n + 1`

23/33

Small-step executions and semantics

A small-step **execution** of a WHILE command is a sequence of configurations

$\gamma_0, \dots, \gamma_p, \dots$ such that, for each i , $\gamma_i \rightarrow \gamma_{i+1}$

We write:

- \rightarrow^* Reflexive and transitive closure of \rightarrow : finite number of transitions
- \rightarrow^+ Transitive closure of \rightarrow : finite, non-zero number of transitions
- \rightarrow^i Exactly i transitions

An execution of (c, σ) :

- ▶ **terminates** iff there exists (c', σ') such that $(c, \sigma) \rightarrow^* (c', \sigma')$ and (c', σ') is final
- ▶ **diverges** iff there exists an infinite transition sequence starting from (c, σ)
- ▶ **is stuck** iff (c, σ) is not final and $(c', \sigma') \not\rightarrow$
- ▶ **goes wrong** iff there exists (c', σ') such that $(c, \sigma) \rightarrow^* (c', \sigma')$ and (c', σ') is stuck

24/33

Exercises

Exercise 5.1 (At home)

Give an SOS to the arithmetic expressions (**Aexp**) of the WHILE language. Is your corresponding transition system deterministic? Explain why.

Exercise 5.2 (At home ☆)

Show that for all σ with $\sigma(n) \geq 1$:

$$(P, \sigma) \rightarrow^* (\text{skip}, \sigma')$$

with $\sigma'(\text{result}) = \sigma(n)!$ where P is the factorial program:

```
result := 1; while n > 1 do (result := n * result; n := n - 1)
```

25/33

Equivalence between NS and SOS

An equivalence of two semantics

Theorem 1

For all c and all σ , we have $\langle c, \sigma \rangle \Downarrow \sigma'$ iff $\langle c, \sigma \rangle \rightarrow^* (\text{skip}, \sigma')$

We prove the two implications of Theorem 1 separately.

26/33

Proof of the direct implication

We want to prove the following lemma:

Lemma 2

If $\langle c, \sigma \rangle \Downarrow \sigma'$, then $\langle c, \sigma \rangle \rightarrow^* (\text{skip}, \sigma')$

We first need to prove the following intermediate lemma:

Lemma 3

If $\langle c_1, \sigma \rangle \rightarrow^* (\text{skip}, \sigma')$, then $\langle c_1 ; c_2, \sigma \rangle \rightarrow^* \langle c_2, \sigma' \rangle$

Exercise 6.1 (At home)

Prove Lemma 3.

27/33

Proof of Lemma 2

Goal: for all command c and states σ, σ' : $\langle c, \sigma \rangle \Downarrow \sigma' \Rightarrow \langle c, \sigma \rangle \rightarrow^* (\text{skip}, \sigma')$.

We proceed by induction on the derivation tree of $\langle c, \sigma \rangle \Downarrow \sigma'$.

Case $\langle x := a, \sigma \rangle \Downarrow \sigma[x \mapsto \mathcal{A}[[a]]\sigma]$

Immediate from the SOS axiom $\langle x := a, \sigma \rangle \rightarrow (\text{skip}, \sigma[x \mapsto \mathcal{A}[[a]]\sigma])$

Case $\langle \text{skip}, \sigma \rangle \Downarrow \sigma$

Immediate: $\langle \text{skip}, \sigma \rangle \rightarrow^* (\text{skip}, \sigma)$

Case

$$\frac{\langle c_1, \sigma \rangle \Downarrow \sigma' \quad \langle c_2, \sigma' \rangle \Downarrow \sigma''}{\langle c_1 ; c_2, \sigma \rangle \Downarrow \sigma''}$$

Thus:

$\langle c_1, \sigma \rangle \rightarrow^* (\text{skip}, \sigma')$ and $\langle c_2, \sigma' \rangle \rightarrow^* (\text{skip}, \sigma'')$ (by IH)

$\langle c_1 ; c_2, \sigma \rangle \rightarrow^* \langle c_2, \sigma' \rangle$ (by Lemma 3)

$\langle c_1 ; c_2, \sigma \rangle \rightarrow^* \sigma''$ (by transitivity of transition sequences)

28/33

Proof of Lemma 2

Case

$$\text{WHI1} \frac{\langle c, \sigma \rangle \Downarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \Downarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma''} \mathcal{B}[[b]]\sigma = \text{tt}$$

The induction hypothesis gives us:

$\langle c, \sigma \rangle \rightarrow^* (\text{skip}, \sigma')$ and $\langle \text{while } b \text{ do } S, \sigma' \rangle \rightarrow^* (\text{skip}, \sigma'')$

Therefore, $\langle c ; \text{while } b \text{ do } c, \sigma \rangle \rightarrow^* \langle \text{while } b \text{ do } c, \sigma' \rangle$ (by Lemma 3)

And thus, $\langle c ; \text{while } b \text{ do } c, \sigma \rangle \rightarrow^* (\text{skip}, \sigma'')$ (by transitivity)

Moreover, according to the SOS, we have:

$$\begin{aligned} \langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow & (\text{if } b \text{ then } \langle c ; \text{while } b \text{ do } c \rangle \text{ else } \text{skip}, \sigma) \\ & \rightarrow \langle c ; \text{while } b \text{ do } c, \sigma \rangle \end{aligned}$$

Therefore, $\langle \text{while } b \text{ do } S, \sigma \rangle \rightarrow^* (\text{skip}, \sigma'')$ (by transitivity)

Other cases same idea (left as an exercise)

29/33

Proof of the reverse implication

We want to prove the following lemma:

Lemma 4

If $(c, \sigma) \rightarrow^* (\text{skip}, \sigma')$, then $(c, \sigma) \Downarrow \sigma'$

We first need to prove the following intermediate lemma:

Lemma 5

If $(c, \sigma) \rightarrow (c', \sigma')$ and $(c', \sigma') \Downarrow \sigma''$, then $(c, \sigma) \Downarrow \sigma''$

Exercise 6.2 (At home)

Using Lemma 5, prove Lemma 4.

30/33

Proof of Lemma 5

Goal: for all c, σ, σ' : if $(c, \sigma) \rightarrow (c', \sigma')$ and $(c', \sigma') \Downarrow \sigma''$, then $(c, \sigma) \Downarrow \sigma''$

We proceed by induction on $(c, \sigma) \rightarrow (c', \sigma')$.

Case ASSIG We have $(x := a, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto \mathcal{A}[a]\sigma])$

We have $(\text{skip}, \sigma[x \mapsto \mathcal{A}[a]\sigma]) \Downarrow \sigma''$ by hypothesis, therefore

$\sigma'' = \sigma[x \mapsto \mathcal{A}[a]\sigma]$ because only rule ASSIG of NS can apply.

Thus, we need to prove $(x := a, \sigma) \Downarrow \sigma[x \mapsto \mathcal{A}[a]\sigma]$. This is given by rule ASSIG of NS.

Case SEQ1 We have $(\text{skip}; c, \sigma) \rightarrow (c, \sigma)$

We have $(c, \sigma) \Downarrow \sigma''$ by hypothesis. We obtain $(\text{skip}; c, \sigma) \Downarrow \sigma''$ using rules SKIP and SEQ of NS.

31/33

Proof of Lemma 5

Case SEQ2 We have $(c_1; c_2, \sigma) \rightarrow (c'_1; c_2, \sigma')$ with $(c_1, \sigma) \rightarrow (c'_1, \sigma')$.

We also have $(c'_1; c_2, \sigma) \Downarrow \sigma''$ by hypothesis.

Therefore, there exists σ_0 such that $(c'_1, \sigma') \Downarrow \sigma_0$ and $(c_2, \sigma_0) \Downarrow \sigma''$

because only the rule SEQ of NS can apply.

Thus, $(c_1, \sigma) \Downarrow \sigma_0$ by induction hypothesis.

Then, we obtain $(c_1; c_2, \sigma) \Downarrow \sigma''$ by rule SEQ of NS.

32/33

Proof of Lemma 5

Case WHI We have

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

There are two cases:

Subcase $\mathcal{B}[b]\sigma = \text{tt}$

Only rule IFT and then SEQ of the NS can apply to the second hypothesis. Therefore, there exists σ_0 such that $(c, \sigma) \Downarrow \sigma_0$ and $(\text{while } b \text{ do } c, \sigma_0) \Downarrow \sigma''$. We obtain $(\text{while } b \text{ do } c, \sigma) \Downarrow \sigma''$ by rule WHI1 of the NS.

Subcase $\mathcal{B}[b]\sigma = \text{ff}$

Only rule IFE and then SKIP of the NS can apply to the second hypothesis. Therefore, $\sigma'' = \sigma$. We obtain $(\text{while } b \text{ do } c, \sigma) \Downarrow \sigma$ by rule WHI2 of the NS.

Other cases Similar techniques apply (exercise at home).

33/33