UNIVERSITÉ Grenoble Alpes

Programming-Language Semantics and Compiler Design / Sémantique des Langages de Programmation et Compilation

Tutorials / Travaux Dirigés (Part 1)

Univ. Grenoble Alpes — UFR IM^{2}AG

Master of Science in Informatics at Grenoble (MoSIG)
Master 1 informatique (M1 info)

Academic Year 2025 - 2026

CONTENTS

SERIES

ONE

NATURAL OPERATIONAL SEMANTICS OF LANGUAGE WHILE

In this exercise series, we consider language While and its natural operational semantics where configurations belong to $(\mathsf{Stm} \times \mathsf{State}) \cup \mathsf{State}$, i.e., are of the form (S, σ) or σ.

1.1 Properties of Arithmetic and Boolean Expressions

1.1.1 Variables

Exercise 1 — Locating variables

What is the set of variables in the following arithmetic expressions.

1. $x + 1$
2. $3 * x + y$

Exercise 2 — Variables of arithmetic expressions

1. Define, in a compositional manner, a function $\mathrm{vars} : \mathsf{Aexp} \to \mathcal{P}(\mathsf{Vars})$ that computes the set of variables for an arithmetic expression[1].
2. Let $a \in \mathsf{Aexp}$ and $\sigma, \sigma' \in \mathsf{State}$ be such that for all $x \in \mathrm{vars}(a)$, $\sigma(x) = \sigma'(x)$. Prove that $\mathcal{A}[a](\sigma) = \mathcal{A}[a](\sigma')$.

Exercise 3 — Free variables of Boolean expressions

1. Define, in a compositional manner, a function $\mathrm{vars} : \mathsf{Bexp} \to \mathcal{P}(\mathsf{Vars})$ that computes the variables of Boolean expressions.
2. Let $b \in \mathsf{Bexp}$ and $\sigma, \sigma' \in \mathsf{State}$ be such that for all $x \in \mathrm{vars}(b)$, $\sigma(x) = \sigma'(x)$. Prove that $\mathcal{B}[b](\sigma) = \mathcal{B}[b](\sigma')$.

1.1.2 State update and substitution

Exercise 4 — Computing some state updates

Indicate the values of variables in the following states, where $\sigma = [x \mapsto 1, y \mapsto 2, z \mapsto 3]$:

1. $\sigma[x \mapsto \mathcal{A}[y + z](\sigma)]$
2. $\underbrace{\sigma[x \mapsto \mathcal{A}[2 * x + y](\sigma)]}_{\sigma'}[y \mapsto \mathcal{A}[x](\sigma')]$

[1] For a set E, $\mathcal{P}(E)$ denotes the set of all subsets of E, called the powerset of E, sometimes also written 2^E because if E has n elements then $\mathcal{P}(E)$ has 2^n elements.

Substitution of $x \in \mathbf{Var}$ by an arithmetic expression $a' \in \mathsf{Aexp}$ in another arithmetic expressions $a \in \mathsf{Aexp}$, written $a[a'/x]$, consists in replacing every occurrence of x by a' in a.

Exercise 5 — Defining Substitution for arithmetic expressions

We consider the arithmetic expressions defined in the course lecture.

1. Define formally substitution for arithmetic expressions.
2. Prove that for all $a, a' \in \mathsf{Aexp}, x \in \mathsf{Vars}, \sigma \in \mathsf{State} : \mathcal{A}[a[a'/x]](\sigma) = \mathcal{A}[a](\sigma[x \mapsto \mathcal{A}[a'](\sigma)])$.

Exercise 6 — Defining substitution for Boolean expressions

1. Define substitution for Boolean expressions where variables are replaced by arithmetic expressions.
2. Prove that for all $b \in \mathsf{Bexp}, a \in \mathsf{Aexp}, x \in \mathsf{Vars}, \sigma \in \mathsf{State} : \mathcal{B}[b[a/x]](\sigma) = \mathcal{B}[b](\sigma[x \mapsto \mathcal{A}[a](\sigma)])$.

1.2 Natural Operational Semantics of Language **While**

Exercise 7 — Computing some (simple) derivation trees

Consider the state σ_0 which maps all variables but x and y to 0, maps x to 5, and y to 7. Remind that, following the conventions defined in class, `;` is right-associative, i.e., $S_1; S_2; S_3$ reads as $S_1; (S_2; S_3)$[2].

1. Give a derivation tree for the following statements executed in σ_0:
   a) $x := y; x := z; y := z$,
   b) $z := x; x := y; y := z$.
2. Give a derivation tree for the following statements:
   a) `if` x `+` y `>=` 3 `then` $y := x$ `else` $x := y$ `fi`,
   b) `if` y `>=` x `then` $y := x$ `else` $x := y$ `fi`.

Exercise 8 — Computing some derivation trees

Consider the empty state []. Give a derivation tree for the following statement:

$y := 1 : x := 2;$ `while` x `>=` 1 `do` $y := y$ `*` $x; x := x$ `-` 1 `od`.

Exercise 9 — Computing some (simple) derivation trees

Consider state σ_0 where x has value 17 and y has value 5.

1. Construct a derivation tree for the following statement when executed in σ_0:

   $z := 0;$ `while` y `<=` x `do` $z := z$ `+` $1; x := x$ `-` y `od`

Exercise 10 — Execution terminates or loops?

For each of the following statements (where x designates a variable of type $\mathbb{Z}$), argue whether:

- its execution loops in every state, or
- its execution stops in every state, or
- there are states from which the execution terminates, and some from which it does not.

1. `while` 1 `<=` x `do` $y := y$ `*` $x;\ x := x$ `-` 1 `od`
2. `while true do skip od`

[2] This is an arbitrary convention. Choosing left-associativity would not change the semantics.

[3] Recall that a state is a partial function for variable names Vars to $\mathbb{Z}$. The empty state is thus the partial function undefined for every variable name.

3. `while not` $(x = 1)$ `do` $y := y * x;\ x := x - 1$ `od`

Exercise 11 — Execution terminates or loops? - proofs

Prove your answers for the previous exercise.

Exercise 12 — Operational semantics for arithmetic and Boolean expressions

The purpose of this exercise is to define an alternative definition to the semantics of arithmetic and Boolean expressions. The semantics given in the lecture course is a declarative semantics defined inductively on the syntax of arithmetic expressions. In this exercise, we want to define an operational semantics.

1. Define an operational semantics for the set of arithmetic expressions **AExp**. The semantics should have two kinds of configurations:
   - (a, σ) denoting that a has to be evaluated in state σ, and
   - v denoting the final value (an element of $\mathbb{Z}$).

   That is the set of configurations of the underlying transition system is

   $$\mathit{Configurations} = (\mathsf{Aexp} \times \mathsf{State}) \cup \mathbb{Z}.$$

2. Prove that the semantics of a defined in this way is equivalent to the one defined in the lecture course (i.e., equivalent to the semantics defined by the inductive function $\mathcal{A}$).
3. Define an operational semantics for the set of Boolean expressions Bexp.
4. Prove that the meaning of a defined in this way is equivalent to the one defined in the lecture course (i.e., equivalent to the meaning defined by the inductive function $\mathcal{A}$).

Exercise 13 — A "subset" of language While

We consider the language defined by the following BNF:

$$S ::= x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}$$

1. What can we say about termination of programs written in this language?
2. Prove your statement.

Exercise 14 — Determinism of the natural operational semantics of language While

Prove that the natural operational semantics of language While is deterministic.

Exercise 15 — Associativity of sequential composition

1. Prove that, for all statements S_1, S_2, S_3, the following statements are semantically equivalent:
   - $S_1; (S_2; S_3)$, and
   - $(S_1; S_2); S_3$.

   You may use the fact that the semantics is deterministic.
2. Prove that, in general, $S_1; S_2$ is not semantically equivalent to $S_2; S_1$.

1.3 Natural Operational Semantics of Extensions of Language While

Exercise 16 — Extending language While with construct `repeat ... until ...`

We want to add the following statement to language While:

`repeat` S `until` b

The informal semantics of this construct is that statement S should be executed until the Boolean condition b becomes true.

1. Provide the semantics rules in order to define `repeat` S `until` b without using the while b do $\cdots$ od construction.
2. Prove that the following statements are semantically equivalent:
   - `repeat` S `until` b, and
   - S; `if` b `then skip else (repeat` S `until` b`) fi`.
3. We want to prove that statement `repeat` S `until` b does not add expressiveness to language While. To do so, give a function which transforms every program with the statement `repeat` S `until` b into a program in language While. Is the given transformation computable? Compare the size of a program and its image resulting of the transformation.

Exercise 17 — Extending language While with construct `for ... from ... to ... do ...`

We want to add another iterative construct to language While. Consider the statement

`for` x `from` a_1 `to` a_2 `do` S.

where the first expression a_1 is the initial value that x is assigned to, the second expression a_2 is the limit that x should be assigned to. Moreover, the "step" of the loop is fixed.
The purpose of this exercise is to extend the semantics of language While by providing appropriate rules for this construct (and without using the `while` $\cdots$ `do` $\cdots$ `od` construct). You may need to assume that you have an "inverse" to $\mathcal{N}$, so that there is a numeral for each number that may arise during computation. There are several alternatives depending on what is allowed for S. Examples of criteria are:

- Evaluation of a_1 and a_2 are done once (at the beginning) or each time the loop body is executed.
- Evaluation of a_1 and a_2 are done each time the loop body is executed.

1. Provide semantics rules for an alternative where the first criterion holds.
2. Provide semantics rules for for an alternative where the second criterion holds.
3. Consider the state σ which maps x to 5. Evaluate the following statement in σ:

   $y := 1$; `for` z `from` 1 `to` x `do` $y := y + x; x := x - 1$.

Exercise 18 — Extending language While with constructs loop and break

We extend the language While with a "break" statement, which terminates the closest enclosing "while" loop. The semantics of the language must be modified, so that transitions now have the form "$(S, \sigma) \hookrightarrow (S', \sigma')$, where S' is either `skip`, meaning that S terminates normally, or break, meaning that the closest enclosing "while" loop has to be terminated.

1. Define semantic rules for all constructs of this extended language.
2. Check that the new semantics is conservative, i.e., if S is a statement of language While and $(S, \sigma) \to \sigma'$ in the semantics of While, then $(S, \sigma) \hookrightarrow (\texttt{skip}, \sigma')$ in the semantics of the extended language.
3. We say that a statement S of the extended language is well-formed if every "break" statement is enclosed in a "while" statement. Define a predicate wf(S) that holds iff S is well-formed.
4. Show that for every well-formed statement S, if $(S, \sigma) \hookrightarrow (S', \sigma')$ then $S' = \texttt{skip}$.
5. Show that for every Boolean condition b and statement S, the statement "`while` b `do` S `od`" is semantically equivalent to " `while tt do if` b `then` S `else break fi od` ".

SERIES

TWO

STRUCTURAL OPERATIONAL SEMANTICS

In this series we start from the structural operational semantics of language While as seen in the course.

Exercise 19 — Computing some derivation sequences

Give a derivation sequence and the associated derivation trees for the following statements in the following states:

1. $z := x; x := y; y := z$ in $[x \mapsto 2, y \mapsto 5, z \mapsto 7]$,
2. $y := 1;$ `while` $\neg(x = 1)$ `do` $y := y * x; x := x - 1$ `od` in $[x \mapsto 5, y \mapsto 7]$,
3. $z := 0;$ `while` $y \leq x$ `do` $z := z + 1; x := x - y$ `od` in $[x \mapsto 17, y \mapsto 5]$.

Exercise 20 — Some properties of the structural semantics

Prove the following claims:

1. If $(S_1, \sigma) \Rightarrow^k \sigma'$ then $(S_1; S_2, \sigma) \Rightarrow^k (S_2; \sigma')$.
   That is the execution of S_1 is not influenced by the statement following it.
2. If $(S_1; S_2, \sigma) \Rightarrow^k \sigma''$ then there exists σ' and k_1 such that: $(S_1, \sigma) \Rightarrow^{k_1} \sigma'$ and $(S_2, \sigma') \Rightarrow^{k-k_1} \sigma''$.
   That is, if it takes k steps to execute $S_1; S_2$ in σ, then there exists an integer k_1 s.t. it takes k_1 steps to execute S_1 in σ to yield a final configuration σ'. Then executing S_2 in σ' takes $k - k_1$ steps.

Exercise 21 — Determinism of the structural operational semantics

1. Show that the structural operational semantics of While is deterministic.
2. Deduce that there is exactly one derivation sequence starting in a configuration (S, σ).
3. Argue that a statement S of While cannot both terminate and loop in a state σ and hence it can not both be always terminating and always looping.

Exercise 22 — Equivalence between the natural and the structural semantics

Prove the following claims.

1. If $(S, \sigma) \to \sigma'$ then $(S, \sigma) \Rightarrow^* \sigma'$.
   It means that if the execution of a statement terminates in a given state in the natural semantics, then it will terminate in the same state in the structural semantics.
2. If $(S, \sigma) \Rightarrow^k \sigma'$ then $(S, \sigma) \to \sigma'$.
   It means that if the execution of a statement terminates in a given state in the operational semantics, then it will terminate in the same state in the operational semantics.
3. Deduce that the natural semantics is equivalent to the structural semantics.

Exercise 23 — Equivalence between some statements

1. Show that the two following statements are semantically equivalent (You may use the fact that the semantics is deterministic.):
   - S; `skip`, and
   - S.
2. Same question for
   - `repeat` S `until` b, and
   - S; `while` $\neg b$ `do` S `od`.

Exercise 24 — Associativity of sequential composition in the structural semantics

1. Prove that, for all statements S_1, S_2, S_3, the following statements are semantically equivalent:
   - $S_1; (S_2; S_3)$
   - $(S_1; S_2); S_3$

   You may use any result of the previous exercises.
2. Prove that, in general, $S_1; S_2$ is not semantically equivalent to $S_2; S_1$.

SERIES

THREE

NATURAL OPERATIONAL SEMANTICS OF PROC AND PPROC

In this exercise series, we consider languages Proc and pProc and their natural operational semantics.

Exercise 25

Compute the semantics of the following program:

```
global x, y
proc F is var z in x := x + 1; z := y; y := z + 1 end
in x := 3; y := 2; F
```

Exercise 26

Compute the semantics of the following program intended to compute and store in y the factorial of 2:

```
global x, y
proc fact is if x ≤ 1 then y := 1 else x := x − 1; fact; y := y ∗ x fi end
in x := 2; fact
```

What is wrong with this program?

Exercise 27

Compute the semantics of the following program:

```
proc F(x) is F(x + 1) end
in F(0)
```

What can you observe?

Exercise 28 — Procedure with input-output parameter

Propose a transition rule for a process with a single input-output parameter, defined by

$$\texttt{proc } F(!?z) \texttt{ is var } x_1, \ldots, x_n \texttt{ in } S \texttt{ end}$$

where $x_1, \ldots, x_n, z \in \mathbf{Var}$, and called in the form $F(!?w)$, where $w \in \mathbf{Var}$.
Informally: upon procedure call, the value of the actual parameter w is assigned to the formal parameter z; upon return, the value of z is assigned back to w.

Exercise 29 — Semantics of language pProc

Define the syntax and semantics of the language pProc (*pure* Proc) defined in class as Proc plus the constraint that every variable should be declared locally (therefore pProc has no global variables, but local variables at the top-level). Propose a semantic relation $\rightarrow_S$ that is as simple as possible.

Exercise 30

Write in pProc a recursive program with the same intent as the one of Exercise 26 (factorial), and compute its semantics.

Exercise 31 — Initialized variables

We change the syntax of Proc, so that variables (either declared as global or local) are systematically initialized: the declaration of variable x now has the form $x := a$.
In a declaration of the form `var` $x_1 := a_1, \ldots, x_n := a_n$, the expressions $a_1, \ldots, a_n$ are evaluated in the global state. For instance, $x_1 := 1, x_2 := x_1 + 1$ has a semantics iff x_1 is declared as a global variable, and the value of x_2 is computed from the value of this global variable.
Define the semantics of this language. Try to get rid of as many components as you can in configurations.

Exercise 32 — Functions

We extend the language pProc (the version of Proc with parameters and without global variables defined in class) with the notion of function:

- A function is a special kind of procedure that has an arbitrary number of formal input parameters $y_1, \ldots, y_m$ and a single output parameter z. The syntax of a function is `fun` $F(y_1, \ldots, y_m)$ is z `var` $x_1, \ldots, x_n$ `in` S `end`
- Functions are called from arithmetic expressions in the form $F(a_1, \ldots, a_m)$. The value of such a function call is the value of the output parameter z upon termination of the function body S evaluated in a state where each formal input parameter y_i has the value of a_i in the state of the caller.

Provide a definition of $\mathcal{A}[F(a_1, \ldots, a_m)](\sigma)$.
Hint: The definition should now depend on the relation $\rightarrow_S$.

SERIES

FOUR

AXIOMATIC SEMANTICS - HOARE LOGIC

Exercise 33

Prove that the following Hoare triples are valid.

1. $\{x = 0\}\ x := x + 1; x := x + 1\ \{x = 2\}$
2. $\{x > 0\}\ y := 1\ \{x = x * y\}$
3. $\{x > a\}\ x := x + 1\ \{x > a + 1\}$
4. $\{x = 4 * a + 4\}\ a := a + 2\ \{x = 4 * a - 4\}$
5. $\{x > a\}\ x := x + 1; x := x + x\ \{x > 2a + 2\}$
6. $\{x \geq 0\}$ `if` $x \geq 0$ `then` $y := 8$ `else` $y := 9$ `fi` $\{y = 8\}$

Exercise 34

Prove that the following Hoare triples are valid.

1. $\{y \geq 0\}\ S\ \{z = y!\}$, where S is:

```
x := y ;
z := 1 ;
while x > 1 do
   z := z * x ;
   x := x - 1
od
```

2. $\{x \geq 0\}\ S\ \{x = 2 \times a + b\}$, where S is:

```
a := x div 2 ;
if even(x) then
   b := 0
else
   b := 1
fi
```

3. $\{a \geq 0 \wedge b > 0\}\ S\ \{a = b \times q + r \wedge r \geq 0 \wedge r < b\}$, where S is:

```
q := 0 ;
r := a ;
while b <= r do
  q := q + 1 ;
  r := r - b ;
od
```

4. $\{b \geq 0\}\ S\ \{p = b^2\}$, where S is:

```
c := b ;
p := 0 ;
while c > 0 do
  p := p + b ;
  c := c - 1 ;
od
```

5. $\{n \geq 1\}\ S\ \{p = m \times n\}$, where S is:

```
p := 0 ;
c := 1 ;
while c <= n do
  p := p + m ;
  c := c + 1
od
```

6. $\{n \geq 0\}\quad S\quad \{x = fib(n) \wedge y = fib(n + 1)\}$, where S is:

```
x := 0 ;
y := 1 ;
c := n ;
while c > 0 do
  t := x + y; x := y;
  y := t; c := c - 1
od
```

where $fib(0) = 0$, $fib(1) = 1$, and $fib(n + 2) = fib(n + 1) + fib(n)$ for all $n \geq 0$ (Fibonacci's sequence).

Exercise 35

Prove that the Hoare triple $\{\mathsf{true}\}\ S\ \{p = |a - b|\}$ is valid, where S is:

```
if a > b then p := a-b else p := b-a fi
```

Exercise 36

Let S be the following program:

```
u := 0;
while x > 1 do
   if even(x) then
      x := x / 2; y := y * 2
   else
      x := x - 1; u := u + y
   fi
od;
y := y + u
```

Use Hoare logic to show that $\{x = x_0 \wedge y = y_0 \wedge x > 0\}\ S\ \{y = x_0 * y_0\}$

Exercise 37

We consider the pre-condition and the post-condition in the incomplete Hoare triple below:

$$\{x = n \wedge y = m \wedge m \geq 0 \wedge n \geq 0\}\ \ldots\ \{z = n * m\}$$

1. Give a program S which satisfies the specification and which uses only addition and subtraction.
2. Demonstrate using Hoare logic that your program satisfies the specification.

Exercise 38

Let S be a statement of While. Let $\mathsf{assigned}(S)$ denote the set of variables assigned in S.

1. Define $\mathsf{assigned}(S)$ inductively on S
2. Let $\{P\}\ S\ \{Q\}$ be a valid Hoare triple. Show that for all predicate R such that $\mathrm{vars}(R) \cap \mathsf{assigned}(S) = \{\}$, $\{P \wedge R\}\ S\ \{Q \wedge R\}$ is a valid Hoare triple. Hint: use structural induction on the proof tree of $\{P\}\ S\ \{Q\}$.

Exercise 39

Let S be a statement of While and y be a variable that does not occur in S. Let $\{P\}\ S\ \{Q\}$ be a valid Hoare triple. Show that if y does not occur in P, S, and Q, then $\{P[y/x]\}\ S[y/x]\ \{Q[y/x]\}$ is a valid Hoare triple.

Exercise 40

We consider the pre-condition and the post-condition in the incomplete Hoare triple below:

$$\{x = n \wedge y = m \wedge m \geq 0 \wedge n \geq 0\}\ S\ \{z = n^m\}$$

1. Give a program that satisfies the specification and which uses only addition and subtraction. You may use procedures with parameters.
2. Demonstrate using Hoare logic that your program satisfies the specification.

Exercise 41

We consider the statement

$$\texttt{for}\ i\ \texttt{from}\ 1\ \texttt{to}\ n\ \texttt{do}\ S$$

where n is the denotation of a natural number in $\mathbb{N}$ and S is a statement in which i is not modified.

1. Recall the operational semantics rule for this construct.
2. Give a rule in axiomatic semantics (Hoare logic) for this construct.

Exercise 42

We add the statement:

`repeat` S `until` b

to the While language.

1. Give an inference rule for `repeat` S `until` b.
2. Demonstrate the correctness of your rule.
3. Is the rule complete? (prove your answer)

Exercise 43

1. Demonstrate that the predicate transformer wp seen in class for statements without loops satisfies $wp(S, P \wedge Q) \iff (wp(S, P) \wedge wp(S, Q))$.
2. Do the following hold?
   a) $wp(S, P \vee Q) \iff (wp(S, P) \vee \wp(S, Q))$.
   b) $wp(S, \neg P) \iff \neg\wp(S, P)$

Exercise 44

We consider the following predicate transformer:

$$\text{post}(S, P) = \{\sigma' \mid \exists\sigma \cdot \sigma \models P \wedge (S, \sigma) \to \sigma'\}$$

1. Prove that: $\{P\}\ S\ \{Q\}$ iff $\text{post}(S, P) \Rightarrow Q$.
2. Prove that: $\{P\}\ S\ \{\text{post}(S, P)\}$.
3. Deduce that the Hoare logic is complete.

Programming-Language Semantics and Compiler Design
(Sémantique des Langages de Programmation et Compilation)