

# Compilation and Program Analysis (#3b): Semantics **ENSL Only**

Ludovic Henrio

Master 1, ENS de Lyon et Dpt Info, Lyon1

2021-2022



# Me: Ludovic Henrio



web:~lhenrio.github.io

email:~ludovic.henrio@ens-lyon.fr

# Credits

JC Filiâtre (ENS Ulm) / JC Fernandez (Grenoble) /  
Nielson-Nielson-Hankin (Book)

## Note on organisation:

- 1: Course
- 2: **exercises and proofs during the course** ;
- 3: **exercises and proofs done after the course (if we have the time)**

- 1 Generalities on semantics
- 2 Operational semantics for mini-while
- 3 Comparing the different semantics

# Semantics

On the abstract syntax we will define a/the semantics. Different kinds of semantics:

- axiomatic
- denotational
- by translation
- operational semantics (natural, structural)

# Axiomatic Semantics (Hoare logic)

(*An axiomatic basis for computer programming*, 1969)

Characterisation by properties on variables, using triples of the form:

$$\{P\} i \{Q\}$$

“if  $P$  is true before the instruction  $i$ , then  $Q$  is true afterwards”

Example :

$$\{x \geq 0\} x := x + 1 \{x > 0\}$$

Example of generating rule:

$$\{P[x \leftarrow E]\} x := E \{P(x)\}$$

► proving properties of programs.

# Denotational Semantics

Associates to an expression  $e$  its mathematical meaning  $\llbracket e \rrbracket$  that represents its computation.

Example : arithmetic expressions with a unique variable  $x$ :

$$e ::= x \mid n \mid e + e \mid e * e \mid \dots$$

You must choose a domain for the mathematical meaning with adequate operations (trivial example for expressions).

$$\llbracket x \rrbracket = x$$

$$\llbracket n \rrbracket = \mathcal{N}(n)$$

$$\llbracket e_1 + e_2 \rrbracket = \llbracket e_1 \rrbracket + \llbracket e_2 \rrbracket$$

$$\llbracket e_1 * e_2 \rrbracket = \llbracket e_1 \rrbracket \times \llbracket e_2 \rrbracket$$

# Semantics by translation

(or Strachey denotational semantics)

We can define the semantics of a language by translation into a language whose semantics is already known.

$$\begin{aligned} \llbracket x = v + v' \rrbracket &= \begin{array}{l} y = \text{get } v; \\ z = \text{get } v'; \\ x = y + z \end{array} \end{aligned}$$



# Operational Semantics

Computations from the program to its computed value.

Operates directly on the abstract syntax. 2 kinds (examples for expressions):

- “natural” or “*big-steps semantics*”, evaluates the program in one step

$$e \longrightarrow v$$

- “by reduction” or “*small-steps semantics*”, repeat the evaluation until a result is obtained:

$$e \rightarrow e_1 \rightarrow e_2 \rightarrow \dots \rightarrow v$$

In general results do not need to be a value.

**Note:** different notations (arrows) exist:  $\Downarrow$  /  $\Rightarrow$  /  $\dots \vdash \dots \rightarrow \dots$

► language specification and proving properties of languages.

- 1 Generalities on semantics
- 2 Operational semantics for mini-while
- 3 Comparing the different semantics

# mini-while

(abstract) grammar:

$S(Smt)$	$::=$	$x := e$	assign
		$skip$	do nothing
		$S_1; S_2$	sequence
		$\text{if } b \text{ then } S_1 \text{ else } S_2$	test
		$\text{while } b \text{ do } S \text{ done}$	loop

# Semantics of expressions

We denote  $State = Var \rightarrow \mathbf{Z}$ .

This kind of state is sometimes called “store”. States are denoted by  $\sigma$ . Update is denoted by  $\sigma[y \mapsto n]$ .

Semantics of arithmetic expressions – Val:  $\mathcal{A} \rightarrow State \rightarrow \mathbf{Z}$  (in each state an integer value): **On board**

$$Val(n, \sigma) = \mathcal{N}(n)$$

$$Val(x, \sigma) =$$

$$Val(e + e', \sigma) =$$

$$Val(e \times e', \sigma) =$$

# Semantics of boolean expressions

$Val : \mathcal{B} \rightarrow State \rightarrow \mathbf{Z}$  **Exercise during course**

$(b ::= tt \mid ff \mid x \mid b \wedge b \mid \dots \mid e < e \mid \dots)$

## TD2: Exercise 1

### Semantics of arithmetic expressions

Show the two following properties:

- 1 Let  $e \in \mathcal{A}$  a given arithmetic expression. Let  $\sigma, \sigma'$  be two states. Show that if  $(\forall x \in \text{Vars}(e), \sigma(x) = \sigma'(x))$ , then  $\text{Val}(e, \sigma) = \text{Val}(e, \sigma')$ .
- 2 Let  $e, e' \in \mathcal{A}$ , show that:

$$\text{Val}(e[e'/x], \sigma) = \text{Val}(e, \sigma[x \mapsto \text{Val}(e', \sigma)])$$

# Natural semantics (big step) for mini-while 1/2

$$\longrightarrow: Stm \rightarrow (State \rightarrow State)$$

$$(x := e, \sigma) \longrightarrow \sigma[x \mapsto Val(e, \sigma)]$$

$$(skip, \sigma) \longrightarrow \sigma$$

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

## Natural semantics (big step) for mini-while 2/2

$$\frac{Val(b, \sigma) = tt \quad (S_1, \sigma) \longrightarrow \sigma'}{(if\ b\ then\ S_1\ else\ S_2, \sigma) \longrightarrow \sigma'}$$

$$\frac{Val(b, \sigma) = ff \quad (S_2, \sigma) \longrightarrow \sigma'}{(if\ b\ then\ S_1\ else\ S_2, \sigma) \longrightarrow \sigma'}$$

$$\frac{Val(b, \sigma) = tt \quad (S, \sigma) \longrightarrow \sigma' \quad (while\ b\ do\ S\ done, \sigma') \longrightarrow \sigma''}{(while\ b\ do\ S\ done, \sigma) \longrightarrow \sigma''}$$

$$\frac{Val(b, \sigma) = ff}{(while\ b\ do\ S\ done, \sigma) \longrightarrow \sigma}$$



# Example

**Compute the semantics (leaves are axioms, nodes are rules) of:**

- $x := 2; \text{while } x > 0 \text{ do } x := x - 1 \text{ done}$
- $x := 2; \text{while } x > 0 \text{ do } x := x + 1 \text{ done}$





# Using the semantics to prove properties

Example: determinism

In mini-while there is a single way to evaluate a program.

## Theorem: Determinism

For all  $S$ , for all  $\sigma, \sigma', \sigma''$  :

- If  $(S, \sigma) \rightarrow \sigma'$  and  $(S, \sigma) \rightarrow \sigma''$  then  $\sigma' = \sigma''$ .
- If  $(S, \sigma) \rightarrow \sigma'$ , there is no infinite derivation.

**Proof by induction on the structure of the derivation tree.**

# Structural Op. Semantics (SOS = small step) for mini-while 1/2

$$(x := e, \sigma) \Rightarrow \sigma[x \mapsto Val(e, \sigma)]$$

$$(\text{skip}, \sigma) \Rightarrow \sigma$$

$$\frac{(S_1, \sigma) \Rightarrow \sigma'}{((S_1; S_2), \sigma) \Rightarrow (S_2, \sigma')} \quad \frac{(S_1, \sigma) \Rightarrow (S'_1, \sigma')}{((S_1; S_2), \sigma) \Rightarrow (S'_1; S_2, \sigma')}$$

$$\frac{Val(b, \sigma) = tt}{(\text{if } b \text{ then } S_1 \text{ else } S_2, \sigma) \Rightarrow (S_1, \sigma)}$$

$$\frac{Val(b, \sigma) = ff}{(\text{if } b \text{ then } S_1 \text{ else } S_2, \sigma) \Rightarrow (S_2, \sigma)}$$

# Structural Op. Semantics (SOS = small step) for mini-while 2/2

$$\begin{aligned} &(\text{while } b \text{ do } S \text{ done}, \sigma) \Rightarrow \\ &(\text{if } b \text{ then } (S; \text{while } b \text{ do } S \text{ done}) \text{ else skip}, \sigma) \end{aligned}$$

# Exercises

**Compute the semantics** (leaves are axioms, nodes are rules)  
of:

- $x := 2; \text{while } x > 0 \text{ do } x := x - 1 \text{ done}$
- $x := 2; \text{while } x > 0 \text{ do } x := x + 1 \text{ done}$

**How to prove determinism for the SOS semantics? What is the structure of the proof?**







- 1 Generalities on semantics
- 2 Operational semantics for mini-while
- 3 Comparing the different semantics

## Comparison : divergence

In general a program diverges if it runs forever.

In mini-while, a program diverges in state  $\sigma$  iff:

- NAT: no successor to  $(S, \sigma)$ .
- SOS: infinite sequence beginning with  $(S, \sigma)$ .

Note: in other languages/semantics there might be other reasons to have no successor (see later in course), and you could have no successor in the SOS without reaching a final state.

# Comparison: equivalence of programs

Semantics is also useful for defining program equivalence, in mini-while it is quite simple:

Two mini-while programs  $S_1$  and  $S_2$  are semantically equivalent iff:

- NAT:  $\forall \sigma, \sigma', (S_1, \sigma) \longrightarrow \sigma' \text{ iff } (S_2, \sigma) \longrightarrow \sigma'$
- SOS:  $\forall \sigma$ :
  - for all config (blocking or not):  $(S_1, \sigma) \Rightarrow^* \sigma' \text{ iff } (S_2, \sigma) \Rightarrow^* \sigma'$
  - there exists an infinite sequence from  $(S_1, \sigma)$  iff same for  $(S_2, \sigma)$

# Are the two semantics equivalent?

$$\mathcal{S}_{NS}[S]\sigma = \begin{cases} \sigma' & \text{If } (S, \sigma) \longrightarrow \sigma' \\ \text{undef} & \text{else} \end{cases}$$

$$\mathcal{S}_{SOS}[S]\sigma = \begin{cases} \sigma' & \text{If } (S, \sigma) \Rightarrow^* \sigma' \\ \text{undef} & \text{else} \end{cases}$$

## Theorem

$$\mathcal{S}_{NS} = \mathcal{S}_{SOS}$$

Proof: see next slides ...

# Equivalence of semantics 1/2

## Proposition

If  $(S, \sigma) \longrightarrow \sigma'$  then  $(S, \sigma) \Rightarrow^* \sigma'$ .

## Lemma for Proposition

If  $(S_1, \sigma) \Rightarrow^k \sigma'$  then  $((S_1; S_2), \sigma) \Rightarrow^k (S_2, \sigma')$

**Proof: structural induction on  $S$  (derivation tree).**

## Equivalence of semantics 2/2

### Proposition

If  $(S, \sigma) \Rightarrow^k \sigma'$  then  $(S, \sigma) \longrightarrow \sigma'$ .

### Lemma for Proposition

If  $(S_1; S_2, \sigma) \Rightarrow^k \sigma''$  then there exists  $\sigma', k_1$  such that  
 $(S_1, \sigma) \Rightarrow^{k_1} \sigma'$  and  $(S_2, \sigma) \Rightarrow^{k-k_1} \sigma''$

**Proof: induction on  $k$ .**

# Expressing parallelism

SOS can express interleaving, NAT cannot:

$$\frac{(S_1, \sigma) \Rightarrow (S'_1, \sigma')}{((S_1 || S_2), \sigma) \Rightarrow (S'_1 || S_2, \sigma')} \quad \frac{(S_2, \sigma) \Rightarrow (S'_2, \sigma')}{((S_1 || S_2), \sigma) \Rightarrow (S_1 || S'_2, \sigma')}$$

... more later in the course.



# Mini-while is not exactly mini-C

## variable initialisation!

- **variable declarations**

- Main problem is scope of variables ( $x$  may not refer to the same variable depending on the point in the program)
- see course on typing

- Expression **evaluation**

restricted to expressions without side-effect, the val function has to be encoded as a set of instructions (a more precise semantics would define several reduction steps)

- **print-int and print-string** (operational semantics not much interesting)
- Mini-C will have **functions** ... defined later in the course

# Conclusion

We have seen different kinds of semantics and compared them briefly.

We have shown how to define operational semantics.

- For expression evaluation
- On mini-while

And how to reason on them to derive language properties (or at least properties of the semantics).

Next course on typing will illustrate more about properties.  
possible additional exercise: **repeat**.

# Empty slides for exercise





















