Compilation and Program Analysis (#3b): Semantics

Ludovic Henrio

Master 1, ENS de Lyon et Dpt Info, Lyon1

2021-2022

Intro

Contact me:

web: lhenrio.github.io

email: ludovic.henrio@ens-lyon.fr

Credits: JC Filliâtre / JC Fernandez / Nielson-Nielson-Hankin / Laure Gonnord

Note on organisation:

1: Course

2: **exercises and proofs during the course** ;

3: **exercises and proofs done at the end the course if we have the time**

Semantics

We will first define an abstract syntax for our language

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 $[\![e]\!]$ that represents its computation.

Example : arithmetic expressions with a unique variable x:

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

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

$$\begin{aligned} [\![x]\!] &= x \\ [\![n]\!] &= \mathcal{N}(n) \\ [\![e_1 + e_2]\!] &= [\![e_1]\!] + [\![e_2]\!] \\ [\![e_1 * e_2]\!] &= [\![e_1]\!] \times [\![e_2]\!] \end{aligned}$$

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.

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

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:

$$e \to e_1 \to e_2 \to \cdots \to v$$

In general results do not need to be a value.

Note: different notations (arrows) exist: $\Downarrow$ / $\Rightarrow$ / ... $\vdash$ ... $\to$ ...

▶ language specification and proving properties of languages.

mini-while

(abstract) grammar:

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

Semantics of expressions

We denote $State = Var \to \mathbf{Z}$.
This kind of state is sometimes called "store". States are denoted by σ. Access is denoted $\sigma(x)$. Update is denoted by $\sigma[y \mapsto n]$.

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

$$
\begin{aligned}
Val(n,\sigma) &= \mathcal{N}(n) \\
Val(x,\sigma) &= \\
Val(e+e',\sigma) &= \\
Val(e\times e',\sigma) &=
\end{aligned}
$$

Semantics of boolean expressions

$Val : \mathcal{B} \rightarrow State \rightarrow \mathbf{Z}$ **Exercise at the end of course**

$(b ::= tt \mid \mathit{ff} \mid x \mid b \wedge b \mid ... \mid e < e \mid ...)$

TD2: Exercise 1

Semantics of arithmetic expressions

Show the two following properties (first one at the end of the course):

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

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

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

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

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

$$(\mathtt{skip}, \sigma) \longrightarrow \sigma$$

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

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

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

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

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

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

Example

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

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

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

The Proof is by induction on the structure of the derivation tree.

do the proof

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

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

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

$$\frac{(S_1, \sigma) \Rightarrow \sigma'}{\big((S_1; S_2), \sigma\big) \Rightarrow (S_2, \sigma')} \qquad \frac{(S_1, \sigma) \Rightarrow (S_1', \sigma')}{\big((S_1; S_2), \sigma\big) \Rightarrow (S_1'; S_2, \sigma')}$$

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

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

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

$$(\texttt{while}\ b\ \texttt{do}\ S\ \texttt{done}, \sigma) \Rightarrow$$

$$(\texttt{if}\ \ b\ \ \texttt{then}\ \ (S; \texttt{while}\ b\ \texttt{do}\ S\ \texttt{done})\ \ \texttt{else}\ \ \texttt{skip}, \sigma)$$

Exercises

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

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

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

Comparison : divergence

In general a program diverges if it runs forever.

In mini-while, a program diverges in state σ iff:

- NAT: no successor to (S, σ).
- SOS: infinite sequence begining with (S, σ).

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

Are the two semantics equivalent?

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

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

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 the derivation tree for $(S, \sigma) \longrightarrow$.

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 σ', 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:

$$\frac{(S_1, \sigma) \Rightarrow (S_1', \sigma')}{\big((S_1||S_2), \sigma\big) \Rightarrow (S_1'||S_2, \sigma')} \quad \frac{(S_2, \sigma) \Rightarrow (S_2', \sigma')}{\big((S_1||S_2), \sigma\big) \Rightarrow (S_1||S_2', \sigma')}$$

... 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 m ore about properties.

possible additional exercise: **repeat**.