Lab Session 2
Operational Semantics and Interpreters

Interpretation and Compilation of Languages

Nova School of Science and Technology
Mário Pereira   mjp.pereira@fct.unl.pt

Version of March 18, 2025

1 Preamble – The While Language

Before we begin the exercises, let us recap the definition of the While language as presented during lectures.

Syntax. Figure 1 gives the syntax of the While language.

e	::=		Expressions
	|	v	integer or Boolean constant
	|	x	variable
	|	$e \oplus e$	binary operator ($+$, $<$, $\ldots$)
s	::=		Statements
	|	x`:=`e	assignment
	|	`if` (e) s `else` s	conditional
	|	`while` (e) s	loop
	|	s`;` s	sequence
	|	`skip`	do nothing
v	::=		Values
	|	n	integer value
	|	b	Boolean value

Figure 1: Syntax of the While language.

Operational Semantics. As in the lectures, we define the big steps operational semantics for our While language, using environments (functions from variables to values) which we denote by σ. For instance, $\sigma = \{a \mapsto 34, b \mapsto 55\}$ is an environment that maps 34 to the variable a and 55 to the variable b. We denote by $\sigma\{x \mapsto v\}$ the environment in which the value for x has been set to v.

We then state the big step operational semantics of WHILE by defining two relations $\sigma, e \Downarrow v$ and $\sigma, s \Downarrow \sigma$ that read respectively "in environment σ, expression e has value v" and "in environment σ, the evaluation of statement s terminates and leads to environment σ'". Figure 2 defines the operational semantics of the WHILE language, via inference rules (as explained in the course).

$$\frac{}{\sigma, n \Downarrow n} \qquad \frac{}{\sigma, b \Downarrow b} \qquad \frac{x \in \mathtt{dom}(\sigma)}{\sigma, x \Downarrow \sigma(x)}$$

$$\frac{\sigma, e_1 \Downarrow n_1 \quad \sigma, e_2 \Downarrow n_2 \quad n \stackrel{\text{def}}{=} n_1 \oplus n_2 \quad \oplus \in \{+, -, *, \ldots\}}{\sigma, e_1 \oplus e_2 \Downarrow n}$$

$$\frac{\sigma, e_1 \Downarrow b_1 \quad \sigma, e_2 \Downarrow b_2 \quad b \stackrel{\text{def}}{=} b_1 \oplus b_2 \quad \oplus \in \{<, \ldots\}}{\sigma, e_1 \oplus e_2 \Downarrow b}$$

`Semantics for expressions`

$$\frac{}{\sigma, \mathtt{skip} \Downarrow \sigma} \qquad \frac{\sigma, s_1 \Downarrow \sigma_1 \quad \sigma_1, s_2 \Downarrow \sigma_2}{\sigma, s_1\,;\,s_2 \Downarrow \sigma_2}$$

$$\frac{\sigma, e \Downarrow v}{\sigma, x\,\mathtt{:=}e \Downarrow \sigma\{x \mapsto v\}}$$

$$\frac{\sigma, e \Downarrow \mathtt{true} \quad \sigma, s_1 \Downarrow \sigma_1}{\sigma, \mathtt{if}\ (e)\ s_1\ \mathtt{else}\ s_2 \Downarrow \sigma_1} \qquad \frac{\sigma, e \Downarrow \mathtt{false} \quad \sigma, s_2 \Downarrow \sigma_2}{\sigma, \mathtt{if}\ (e)\ s_1\ \mathtt{else}\ s_2 \Downarrow \sigma_2}$$

$$\frac{\sigma, e \Downarrow \mathtt{true} \quad \sigma, s \Downarrow \sigma_1 \quad \sigma_1, \mathtt{while}\ (e)\ s \Downarrow \sigma_2}{\sigma, \mathtt{while}\ (e)\ s \Downarrow \sigma_2} \qquad \frac{\sigma, e \Downarrow \mathtt{false}}{\sigma, \mathtt{while}\ (e)\ s \Downarrow \sigma}$$

`Semantics for statements`

Figure 2: Big-step operational semantics of the While language.

1.1 Semantically Equivalent Programs

Two statements s_1 and s_2 of the WHILE language are said to be semantically equivalent if for all states σ and σ'

$$\sigma, s_1 \Downarrow \sigma' \qquad \text{if and only if} \qquad \sigma, s_2 \Downarrow \sigma'$$

Exercise 1. Show that the statement

$$\mathtt{if}\ (e)\ s_1\ \mathtt{else}\ s_2$$

is semantically equivalent to

$$\mathtt{if}\ (\neg\, e)\ s_2\ \mathtt{else}\ s_1$$

where $\neg\, e$ stands for the Boolean negation of the value of expression e. You may consider the following operational semantic rules for $\sigma, \neg\, e \Downarrow b$:

$$\frac{\sigma, e \Downarrow \mathtt{true}}{\sigma, \neg\, e \Downarrow \mathtt{false}} \qquad \frac{\sigma, e \Downarrow \mathtt{false}}{\sigma, \neg\, e \Downarrow \mathtt{true}}$$

□

Exercise 2. Show that the statement

$$\texttt{while}\,(b)\ s$$

is semantically equivalent to

$$\texttt{if}\,(b)\ s;\texttt{while}\,(b)\ s\ \texttt{else skip}$$

□

1.2 Extensions to the While Language

Consider we extend the WHILE language with

- pair constructors (e, e) and get the left and the right pair components denoted respectively by $\texttt{fst}\ e$ and $\texttt{snd}\ e$;
- $\texttt{repeat}\ s\ \texttt{until}\ b$ loops.

The syntax of the language is extended as follows:

$$\begin{array}{lcll}
e & ::= & & \textsc{Expressions} \\
 & \ldots & & \\
 & | & (e, e) & \text{pair construction} \\
 & | & \texttt{fst}\ e & \text{first projection} \\
 & | & \texttt{snd}\ e & \text{second projection} \\
 & & & \\
s & ::= & & \textsc{Statements} \\
 & \ldots & & \\
 & | & \texttt{repeat}\ s\ \texttt{until}\ e & \text{repeat loop} \\
 & & & \\
v & ::= & & \textsc{Values} \\
 & \ldots & & \\
 & | & (v, v) & \text{pair value}
\end{array}$$

1.3 Pairs

Exercise 3. Complete the definition of $\sigma, e \Downarrow v$ by giving the inference rules for the operation semantics of pairs. Concretely, define the inference rule for evaluating (e_1, e_2) and the inference rules to evaluate $\texttt{fst}\ e$ and $\texttt{snd}\ e$ in a given environment σ. □

Exercise 4. Give a derivation for the evaluation

$$\{x \mapsto (21, 34)\}, \texttt{if}\ (\texttt{fst}\ x > 0)\ x\texttt{:=}(\texttt{snd}\ x, \texttt{fst}\ x+\texttt{snd}\ x)\ \texttt{else}\ x\texttt{:=}0 \Downarrow \{x \mapsto (34, 55)\}$$

□

Exercise 5. Consider the following WHILE program:

```
i := 0;
p := 0;
while (i < 5)
  p := (i + 1, p);
  i := i + 1
```

Consider σ to be the final environment to which the above program evaluates. What is the value of $\sigma(\mathtt{p})$?

A $(((((0,1),2),3),4),5)$

B $(5,(4,(3,(2,(1,0)))))$

C $(0,(1,(2,(3,(4,5)))))$

D there is not such value, since the program produces a runtime error.

$\square$

1.4 Repeat Loop

Exercise 6. We now consider the `repeat` s `until` e loop. Intuitively, its semantics is that we execute the statement s then check whether the condition e holds: if e evaluates to `true`, we leave the loop. Otherwise, e evaluates to `false`, and the loop is executed again. Define the inferences rules for each case of the `repeat` s `until` e. $\square$

Exercise 7. As one may notice, `repeat`-loops do not extend the power of the language. In fact, they can be simply rewritten (for instance by a compiler phase) using sequence and a `while` loop. Explain how this rewriting can be done. (You can simply write an equation `repeat` s `until` $e = \ldots$) $\square$