

# 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 15, 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 ( $+$ , $<$ , $\dots$ )
$s ::=$	STATEMENTS
$x := e$	assignment
$\text{if } (e) \text{ } s \text{ else } s$	conditional
$\text{while } (e) \text{ } s$	loop
$s ; s$	sequence
$\text{skip}$	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  $\sigma$ . 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  $\sigma$ , expression  $e$  has value  $v$ ” and “in environment  $\sigma$ , the evaluation of statement  $s$  terminates and leads to environment  $\sigma'$ ”. Figure 2 defines the operational semantics of the WHILE language, via inference rules (as explained in the course).

$$\begin{array}{c}
\frac{}{\sigma, n \Downarrow n} \quad \frac{}{\sigma, b \Downarrow b} \quad \frac{x \in \text{dom}(\sigma)}{\sigma, x \Downarrow \sigma(x)} \\
\frac{\sigma, e_1 \Downarrow n_1 \quad \sigma, e_2 \Downarrow n_2 \quad n \stackrel{\text{def}}{=} n_1 \oplus n_2 \quad \oplus \in \{+, -, *, \dots\}}{\sigma, e_1 \oplus e_2 \Downarrow n} \\
\frac{\sigma, e_1 \Downarrow b_1 \quad \sigma, e_2 \Downarrow b_2 \quad b \stackrel{\text{def}}{=} b_1 \oplus b_2 \quad \oplus \in \{<, \dots\}}{\sigma, e_1 \oplus e_2 \Downarrow n} \\
\textbf{Semantics for expressions} \\
\\
\frac{}{\sigma, \text{skip} \Downarrow \sigma} \quad \frac{\sigma, s_1 \Downarrow \sigma_1 \quad \sigma_1, s_2 \Downarrow \sigma_2}{\sigma, s_1; s_2 \Downarrow \sigma_2} \\
\frac{\sigma, e \Downarrow v}{\sigma, x := e \Downarrow \sigma\{x \mapsto v\}} \\
\frac{\sigma, e \Downarrow \text{true} \quad \sigma, s_1 \Downarrow \sigma_1}{\sigma, \text{if } (e) \text{ } s_1 \text{ else } s_2 \Downarrow \sigma_1} \quad \frac{\sigma, e \Downarrow \text{false} \quad \sigma, s_2 \Downarrow \sigma_2}{\sigma, \text{if } (e) \text{ } s_1 \text{ else } s_2 \Downarrow \sigma_2} \\
\frac{\sigma, e \Downarrow \text{true} \quad \sigma, s \Downarrow \sigma_1 \quad \sigma_1, \text{while } (e) \text{ } s \Downarrow \sigma_2}{\sigma, \text{while } (e) \text{ } s \Downarrow \sigma_2} \quad \frac{\sigma, e \Downarrow \text{false}}{\sigma, \text{while } (e) \text{ } s \Downarrow \sigma} \\
\textbf{Semantics for statements}
\end{array}$$

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  $\sigma$  and  $\sigma'$

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

**Exercise 1.** Show that the statement

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

is semantically equivalent to

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

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

$$\frac{\sigma, e \Downarrow \text{true}}{\sigma, \neg e \Downarrow \text{false}} \quad \frac{\sigma, e \Downarrow \text{false}}{\sigma, \neg e \Downarrow \text{true}}$$

□

**Exercise 2.** Show that the statement

`while (b) s`

is semantically equivalent to

`if (b) s; while (b) s 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 `fst e` and `snd e`;
- `repeat s until b` loops.

The syntax of the language is extended as follows:

<code>e</code>	::=	EXPRESSIONS
...		
	<code>(e, e)</code>	pair construction
	<code>fst e</code>	first projection
	<code>snd e</code>	second projection
<code>s</code>	::=	STATEMENTS
...		
	<code>repeat e until s</code>	repeat loop
<code>v</code>	::=	VALUES
...		
	<code>(v, v)</code>	pair value

## 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 `fst e` and `snd e` in a given environment  $\sigma$ . □

**Exercise 4.** Give a derivation for the evaluation

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

□

**Exercise 5.** Consider the following WHILE program:

```

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

```

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

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.

□

## 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**. □

**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** = ...) □