Homework 2: Operational Semantics for WHILE

CS 252: Advanced Programming Languages
Arun Murugan
San José State University

1 Introduction

For this assignment, you will implement the semantics for a small imperative language, named WHILE.

The language for WHILE is given in Figure 1. Unlike the Bool* language we discussed previously, WHILE supports *mutable references*. The state of these references is maintained in a *store*, a mapping of references to values. ("Store" can be thought of as a synonym for heap.) Once we have mutable references, other language constructs become more useful, such as sequencing operations ($e_1; e_2$).

2 Big-step semantics

3 Small-step semantics

The small-step semantics for WHILE are given in Figure **??**. Most of these rules are fairly straightforward, but there are a couple of points to note with the [SS-WHILE] rule. First of all, this is the only rule that makes a more complex expression when it has finished. (This rule is much cleaner when specified with the big-step operational semantics.)

Secondly, note the final value of this expression once the while loop completes. It will *always* be `false` when it completes. We could have created a special value, such as `null`, or we could have made the while loop a statement that returns no value. Both choices, however, would complicate our language needlessly.

4 YOUR ASSIGNMENT

Part 1: Rewrite the operational semantic rules for WHILE in LaTeX to use big-step operational semantics instead. Submit both your LaTeX source and the generated PDF file.

Extend your semantics with features to handle boolean values. **Do not treat these a binary operators.** Specifically, add support for:

- `and`
- `or`
- `not`

The exact behavior of these new features is up to you, but should seem reasonable to most programmers.

Part 2: Once you have your semantics defined, download `WhileInterp.hs` and implement the `evaluate` function, as well as any additional functions you need. Your implementation must be consistent with your operational semantics, *including your extensions for* `and`*,* `or`*, and* `not`. Also, you may not change any type signatures provided in the file.

Finally, implement the interpreter to match your semantics.

Zip all files together into `hw2.zip` and submit to Canvas.

$e ::=$		*Expressions*
	x	variables/addresses
	v	values
	$x := e$	assignment
	$e; e$	sequential expressions
	$e \; op \; e$	binary operations
	$\texttt{if } e \texttt{ then } e \texttt{ else } e$	conditional expressions
	$\texttt{while } (e) \; e$	while expressions
$v ::=$		*Values*
	i	integer values
	b	boolean values
$op ::=$	$+ \mid - \mid * \mid / \mid > \mid >= \mid < \mid <=$	*Binary operators*

Figure 1: The WHILE language

Runtime Syntax:

$$\sigma \in Store = variable \to v$$

Evaluation Rules: $\boxed{e, \sigma \to e1, \sigma1}$

$$[\textsc{bs-seqctx}] \quad \frac{\begin{array}{c} e1, \sigma \Downarrow v1, \sigma1 \\ e2, \sigma1 \Downarrow v2, \sigma2 \end{array}}{e1; e2, \sigma \Downarrow v2, \sigma2} \qquad [\textsc{bs-var}] \quad \frac{\sigma(x) = v}{x, \sigma \Downarrow v, \sigma}$$

$$[\textsc{bs-seq}] \quad \frac{}{v; e, \sigma \Downarrow v1, \sigma1} \qquad [\textsc{bs-val}] \quad \frac{}{x, \sigma \Downarrow v, \sigma}$$

$$[\textsc{bs-op}] \quad \frac{\begin{array}{c} e1, \sigma \Downarrow v1, \sigma1 \\ e2, \sigma1 \Downarrow v2, \sigma2 \\ v = v_1 \; op \; v_2 \end{array}}{v_1 \; op \; v_2, \sigma \Downarrow v, \sigma2} \qquad [\textsc{bs-assignctx}] \quad \frac{e1, \sigma \Downarrow v1, \sigma1}{x := e1, \sigma \Downarrow v1, \sigma1[x := v1]}$$

$$[\textsc{bs-ifctxtrue}]\quad \frac{\begin{array}{c} e_1, \sigma \Downarrow \texttt{true}, \sigma 1 \\ e_2, \sigma 1 \Downarrow v, \sigma 2 \end{array}}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \Downarrow v, \sigma 2}$$

$$[\textsc{bs-ifctxfalse}]\quad \frac{\begin{array}{c} e_1, \sigma \Downarrow \texttt{false}, \sigma 1 \\ e_3, \sigma 1 \Downarrow v, \sigma 2 \end{array}}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \Downarrow v, \sigma 2}$$

$$[\textsc{bs-whileTrue}]\quad \frac{\begin{array}{c} e_1, \sigma \Downarrow \texttt{true}, \sigma 1 \\ e_2, \sigma 1 \Downarrow v, \sigma 2 \\ \texttt{while } (e_1)\ e_2, \sigma 2 \Downarrow v1, \sigma 3 \end{array}}{\texttt{while } (e_1)\ e_2, \sigma 2 \Downarrow v1, \sigma 3}$$

$$[\textsc{bs-whileFalse}]\quad \frac{e_1, \sigma \Downarrow \texttt{false}, \sigma 1}{\texttt{while } (e_1)\ e_2, \sigma \Downarrow false, \sigma 1}$$

$$[\textsc{bs-and}1]\quad \frac{\begin{array}{c} e1, \sigma \Downarrow \texttt{true}, \sigma 1 \\ e2, \sigma 1 \Downarrow \texttt{true}, \sigma 2 \end{array}}{e1ANDe2, \sigma \Downarrow \texttt{true}, \sigma 2}$$

$$[\textsc{bs-and}2]\quad \frac{\begin{array}{c} e1, \sigma \Downarrow \texttt{true}, \sigma 1 \\ e2, \sigma 1 \Downarrow \texttt{false}, \sigma 2 \end{array}}{e1ANDe2, \sigma \Downarrow \texttt{false}, \sigma 2}$$

$$[\textsc{bs-and}3]\quad \frac{e1, \sigma \Downarrow \texttt{false}, \sigma 1}{e1ANDe2, \sigma \Downarrow \texttt{false}, \sigma 1}$$

$$[\textsc{bs-or}1]\quad \frac{e1, \sigma \Downarrow \texttt{true}, \sigma 1}{e1ORe2, \sigma \Downarrow \texttt{true}, \sigma 1}$$

$$[\textsc{bs-or}2]\quad \frac{\begin{array}{c} e1, \sigma \Downarrow \texttt{false}, \sigma 1 \\ e2, \sigma 1 \Downarrow \texttt{true}, \sigma 2 \end{array}}{e1ORe2, \sigma \Downarrow \texttt{true}, \sigma 2}$$

$$[\textsc{bs-or}3]\quad \frac{\begin{array}{c} e1, \sigma \Downarrow \texttt{false}, \sigma 1 \\ e2, \sigma 1 \Downarrow \texttt{false}, \sigma 2 \end{array}}{e1ORe2, \sigma \Downarrow \texttt{false}, \sigma 2}$$

$$[\textsc{bs-not}1]\quad \frac{e, \sigma \Downarrow \texttt{true}, \sigma 1}{NOTe, \sigma \Downarrow \texttt{false}, \sigma 1}$$

$$[\textsc{bs-not}2]\quad \frac{e, \sigma \Downarrow \texttt{false}, \sigma 1}{NOTe, \sigma \Downarrow \texttt{true}, \sigma 1}$$