Homework 2: Operational Semantics for WHILE

CS 252: Advanced Programming Languages
Prof. Thomas H. Austin
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 Small-step semantics

The small-step semantics for WHILE are given in Figure 3. For the sake of brevity, these rules use *evaluation contexts* (C), which specify which *redex* will be evaluated next. The evaluation rules then apply to the "hole" ($\bullet$) in this context.

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.

3 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:

$$
\begin{array}{llll}
C & \in & \mathit{Context} & ::= & C; e \mid C\ op\ e \mid v\ op\ C \mid x := C \mid \texttt{if}\ C\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \bullet \\
\sigma & \in & \mathit{Store} & = & \mathit{variable} \ \rightarrow\ v
\end{array}
$$

Evaluation Rules: $\boxed{e, \sigma \rightarrow e', \sigma'}$

[SS-VAR]
$$\frac{x \in \mathit{domain}(\sigma) \qquad \sigma(x) = v}{C[x], \sigma \rightarrow C[v], \sigma}$$

[SS-ASSIGN]
$$\frac{}{C[x := v], \sigma \rightarrow C[v], \sigma[x := v]}$$

[SS-OP]
$$\frac{v = v_1\ op\ v_2}{C[v_1\ op\ v_2], \sigma \rightarrow C[v], \sigma}$$

[SS-SEQ]
$$\frac{}{C[v; e], \sigma \rightarrow C[e], \sigma}$$

[SS-IFTRUE]
$$\frac{}{C[\texttt{if true then}\ e_1\ \texttt{else}\ e_2], \sigma \rightarrow C[e_1], \sigma}$$

[SS-IFFALSE]
$$\frac{}{C[\texttt{if false then}\ e_1\ \texttt{else}\ e_2], \sigma \rightarrow C[e_2], \sigma}$$

[SS-WHILE]
$$\frac{}{C[\texttt{while}\ (e_1)\ e_2], \sigma \rightarrow C[\texttt{if}\ e_1\ \texttt{then}\ e_2; \texttt{while}\ (e_1)\ e_2\ \texttt{else false}], \sigma}$$

Figure 2: Small-step semantics for WHILE

Runtime Syntax:

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

Evaluation Rules: $\boxed{e, \sigma \rightarrow e', \sigma'}$

[BS-VAR] $$\frac{x \in domain(\sigma) \qquad \sigma(x) = v}{x, \sigma \Downarrow v, \sigma}$$

[BS-VAL] $$\frac{}{v, \sigma \Downarrow v, \sigma}$$

[BS-ASSIGN] $$\frac{e_1, \sigma \Downarrow v_1, \sigma_1}{x := e_1, \sigma \Downarrow v_1, \sigma_1[x := v_1]}$$

[BS-OP] $$\frac{\begin{array}{c} e_1, \sigma \Downarrow v_1, \sigma_1 \\ e_2, \sigma_1 \Downarrow v_2, \sigma_2 \end{array}}{e_1 \ op \ e_2, \sigma \Downarrow v_1 \ op \ v_2, \sigma_2}$$

[BS-SEQ] $$\frac{\begin{array}{c} e_1, \sigma \Downarrow v_1, \sigma_1 \\ e_2, \sigma_1 \Downarrow v_2, \sigma_2 \end{array}}{e_1; e_2, \sigma \Downarrow v_2, \sigma_2}$$

[BS-IFTRUE] $$\frac{\begin{array}{c} e_1, \sigma \Downarrow \texttt{true}, \sigma \\ e_2, \sigma \Downarrow v, \sigma \end{array}}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \Downarrow v, \sigma}$$

[BS-IFFALSE] $$\frac{\begin{array}{c} e_1, \sigma \Downarrow \texttt{false}, \sigma \\ e_3, \sigma \Downarrow v, \sigma \end{array}}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \Downarrow v, \sigma}$$

[BS-WHILE] $$\frac{\begin{array}{c} \texttt{while } (e_1) \ e_2, \sigma_2 \Downarrow v_1, \sigma_1 \\ e_2, \sigma \Downarrow v_2, \sigma_2 \end{array}}{\texttt{while } (e_1) \ e_2, \sigma \Downarrow \texttt{if } e_1 \texttt{ then } v_1, \sigma_1 \texttt{ else false}, \sigma}$$

[BS-AND-TRUE] $$\frac{e_1, \sigma \Downarrow \texttt{true}, \sigma \qquad e_2, \sigma \Downarrow \texttt{true}, \sigma \qquad \texttt{or } e_1 \ e_2, \sigma \Downarrow \texttt{true}, \sigma}{\texttt{and } e_1 \ e_2, \sigma \Downarrow \texttt{true}, \sigma}$$

[BS-AND-FALSE] $$\frac{\begin{array}{c} e_1, \sigma \Downarrow \texttt{false}, \sigma \\ e_2, \sigma \Downarrow \texttt{false}, \sigma \end{array}}{\texttt{and } e_1 \ e_2, \sigma \Downarrow \texttt{false}, \sigma}$$

[BS-OR-TRUE] $$\frac{\begin{array}{c} e_1, \sigma \Downarrow \texttt{true}, \sigma \\ e_2, \sigma \Downarrow \texttt{true}, \sigma \end{array}}{\texttt{or } e_1 \ e_2, \sigma \Downarrow \texttt{true}, \sigma}$$

[BS-OR-FALSE] $$\frac{e_1, \sigma \Downarrow \texttt{false}, \sigma \qquad e_2, \sigma \Downarrow \texttt{false}, \sigma \qquad \texttt{and } e_1 \ e_2, \sigma \Downarrow \texttt{false}, \sigma}{\texttt{or } e_1 \ e_2, \sigma \Downarrow \texttt{false}, \sigma}$$

[BS-NOT-TRUE] $$\frac{e_1, \sigma \Downarrow \texttt{false}, \sigma}{\texttt{not } e_1, \sigma \Downarrow \texttt{true}, \sigma}$$

[BS-NOT-FALSE] $$\frac{e_1, \sigma \Downarrow \texttt{true}, \sigma}{\texttt{not } e_1, \sigma \Downarrow \texttt{false}, \sigma}$$

Figure 3: Big-step semantics for WHILE