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 4. 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.

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

Figure 1: The WHILE language

Runtime Syntax:

$$\sigma \in \mathit{Store} = \mathit{variable} \rightarrow v$$

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

$$[\text{SS-SEQCTX}] \quad \frac{e_1, \sigma \rightarrow e_1', \sigma'}{e_1; e_2, \sigma \rightarrow e_1'; e_2, \sigma'} \qquad [\text{SS-VAR}] \quad \frac{x \in \mathit{domain}(\sigma) \qquad \sigma(x) = v}{x, \sigma \rightarrow v, \sigma}$$

$$[\text{SS-SEQ}] \quad \frac{}{v; e, \sigma \rightarrow e, \sigma} \qquad [\text{SS-ASSIGNCTX}] \quad \frac{e_1, \sigma \rightarrow e_1', \sigma'}{x := e, \sigma \rightarrow x := e', \sigma'}$$

$$[\text{SS-OPCTX1}] \quad \frac{e_1, \sigma \rightarrow e_1', \sigma'}{e_1 \; op \; e_2, \sigma \rightarrow e_1' \; op \; e_2, \sigma'} \qquad [\text{SS-ASSIGN}] \quad \frac{}{x := v, \sigma \rightarrow v, \sigma[x := v]}$$

$$[\text{SS-OPCTX2}] \quad \frac{e_2, \sigma \rightarrow e_2', \sigma'}{v_1 \; op \; e_2, \sigma \rightarrow v_1 \; op \; e_2', \sigma'} \qquad [\text{SS-IFTRUE}] \quad \frac{}{\texttt{if true then } e_1 \texttt{ else } e_2, \sigma \rightarrow e_1, \sigma}$$

$$[\text{SS-OP}] \quad \frac{v = v_1 \; op \; v_2}{v_1 \; op \; v_2, \sigma \rightarrow v, \sigma} \qquad [\text{SS-IFFALSE}] \quad \frac{}{\texttt{if false then } e_1 \texttt{ else } e_2, \sigma \rightarrow e_2, \sigma}$$

$$[\text{SS-IFCTX}] \quad \frac{e_1, \sigma \rightarrow e_1', \sigma'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \rightarrow \texttt{if } e_1' \texttt{ then } e_2 \texttt{ else } e_3, \sigma'}$$

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

Figure 2: Small-step semantics for WHILE

4 Solution

The Big Step semantics can be defined as below:

Runtime Syntax:

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

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

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

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

[BS-ASSIGN] $$\frac{e, \sigma \Downarrow v, \sigma'}{x := e, \sigma \Downarrow x := v, \sigma'}$$

[BS-SEQ] $$\frac{e_1, \sigma \Downarrow v_1, \sigma_1 \quad e_2, \sigma_1 \Downarrow v_2, \sigma_2}{e_1; e_2, \sigma \rightarrow v_2, \sigma_2}$$

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

[BS-IFTRUE] $$\frac{e_1, \sigma \Downarrow \texttt{true}, \sigma_1 \quad e_2, \sigma_1 \Downarrow v_2, \sigma_2}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \Downarrow v_2, \sigma_2}$$

[BS-IFFALSE] $$\frac{e_1, \sigma \Downarrow \texttt{false}, \sigma_1 \quad e_3, \sigma_1 \Downarrow v_2, \sigma_2}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \Downarrow v_2, \sigma_2}$$

[BS-AND-TRUE] $$\frac{e_1, \sigma \Downarrow \texttt{true}, \sigma_1 \quad e_2, \sigma_1 \Downarrow v_2, \sigma_2}{e_1 \ and \ e_2, \sigma \Downarrow \texttt{true}, \sigma_2}$$

[BS-AND-FALSE] $$\frac{e_1, \sigma \Downarrow \texttt{false}, \sigma_1}{e_1 \ and \ e_2, \sigma \Downarrow \texttt{false}, \sigma_1}$$

[BS-OR-TRUE] $$\frac{e_1, \sigma \Downarrow \texttt{true}, \sigma_1}{e_1 \ or \ e_2, \sigma \Downarrow \texttt{true}, \sigma_1}$$

[BS-OR-FALSE] $$\frac{e_1, \sigma \Downarrow \texttt{false}, \sigma_1 \quad e_2, \sigma_1 \Downarrow v_2, \sigma_2}{e_1 \ or \ e_2, \sigma \Downarrow v_2, \sigma_2}$$

[BS-NOT-TRUE] $$\frac{e_1, \sigma \Downarrow \texttt{false}, \sigma_1}{not \ e_1, \sigma \Downarrow \texttt{true}, \sigma_1}$$

[BS-NOT-FALSE] $$\frac{e_1, \sigma \Downarrow \texttt{true}, \sigma_1}{not \ e_1, \sigma \Downarrow \texttt{false}, \sigma_1}$$

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

Figure 3: Big-step semantics for WHILE