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.

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 remove the contexts (C[...]) and to use evaluation order rules instead. Submit both your LATEX source and the generated PDF file.

Extend your semantics with features to handle boolean values. 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.

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

Figure 1: The WHILE language

Runtime Syntax:

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

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

$$
[\textsc{ss-var}] \quad \frac{x \in domain(\sigma) \qquad \sigma(x) = v}{C[x], \sigma \to C[v], \sigma}
$$

$$
[\textsc{ss-assign}] \quad \frac{}{C[x := v], \sigma \to C[v], \sigma[x := v]}
$$

$$
[\textsc{ss-op}] \quad \frac{v = v_1 \; op \; v_2}{C[v_1 \; op \; v_2], \sigma \to C[v], \sigma}
$$

$$
[\textsc{ss-seq}] \quad \frac{}{C[v; e], \sigma \to C[e], \sigma}
$$

$$
[\textsc{ss-iftrue}] \quad \frac{}{C[\texttt{if true then } e_1 \texttt{ else } e_2], \sigma \to C[e_1], \sigma}
$$

$$
[\textsc{ss-iffalse}] \quad \frac{}{C[\texttt{if false then } e_1 \texttt{ else } e_2], \sigma \to C[e_2], \sigma}
$$

$$
[\textsc{ss-while}] \quad \frac{}{C[\texttt{while } (e_1) \; e_2], \sigma \to C[\texttt{if } e_1 \texttt{ then } e_2; \texttt{while } (e_1) \; e_2 \texttt{ else false}], \sigma}
$$

Figure 2: Small-step semantics for WHILE