Homework 2: Operational Semantics for WHILE

CS 252: Advanced Programming Languages
Kushagra Bainsla
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 2. 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 Big-step Semantics

The big-step semantics for WHILE are given in Figure 3. In big-step (natural) semantics, a judgment $e, \sigma \Downarrow v, \sigma'$ means that expression e evaluated in store σ produces value v and (possibly updated) store σ'. Unlike small-step semantics, each rule describes the complete evaluation of an expression in a single step. The while rule is notable: its condition is first evaluated, and if `true`, the body is evaluated followed by a recursive big-step evaluation of the entire while expression again. The boolean extensions (`and`, `or`, `not`) use short-circuit evaluation: `and` does not evaluate its second argument if the first is `false`, and `or` does not evaluate its second argument if the first is `true`.

$e ::=$		*Expressions*
	x	variables/addresses
	v	values
	$x := e$	assignment
	$e; e$	sequential expressions
	$e\ op\ e$	binary operations
	`if` e `then` e `else` e	conditional expressions
	`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: $\boxed{e, \sigma \to e', \sigma'}$

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

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

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

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

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

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

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

Figure 2: Small-step semantics for WHILE

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

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

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

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

$$[\text{BS-SEQ}] \quad \frac{e_1, \sigma \Downarrow v_1, \sigma' \qquad e_2, \sigma' \Downarrow v_2, \sigma''}{e_1; e_2, \sigma \Downarrow v_2, \sigma''}$$

$$[\text{BS-OP}] \quad \frac{e_1, \sigma \Downarrow v_1, \sigma' \qquad e_2, \sigma' \Downarrow v_2, \sigma'' \quad v = v_1 \ op \ v_2}{e_1 \ op \ e_2, \sigma \Downarrow v, \sigma''}$$

$$[\text{BS-IFTRUE}] \quad \frac{e_1, \sigma \Downarrow \texttt{true}, \sigma' \qquad e_2, \sigma' \Downarrow v, \sigma''}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \Downarrow v, \sigma''}$$

$$[\text{BS-IFFALSE}] \quad \frac{e_1, \sigma \Downarrow \texttt{false}, \sigma' \qquad e_3, \sigma' \Downarrow v, \sigma''}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \Downarrow v, \sigma''}$$

$$[\text{BS-WHILE-FALSE}] \quad \frac{e_1, \sigma \Downarrow \texttt{false}, \sigma'}{\texttt{while } (e_1) \ e_2, \sigma \Downarrow \texttt{false}, \sigma'}$$

$$[\text{BS-WHILE-TRUE}] \quad \frac{e_1, \sigma \Downarrow \texttt{true}, \sigma' \qquad e_2, \sigma' \Downarrow v', \sigma'' \qquad \texttt{while } (e_1) \ e_2, \sigma'' \Downarrow v, \sigma'''}{\texttt{while } (e_1) \ e_2, \sigma \Downarrow v, \sigma'''}$$

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

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

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

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

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

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

Figure 3: Big-step semantics for WHILE