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 big-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.

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
	`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

Evaluation Rule: $(e_0, \sigma_0) \Downarrow (e_1, \sigma_1)$

[BS-VAR] $$\frac{v_0 = \sigma_0(x_0)}{x_0, \sigma_0 \Downarrow v_0, \sigma_0}$$

[BS-VAL] $$\frac{}{v_0, \sigma_0 \Downarrow v_0, \sigma_0}$$

[BS-ASSNGVAL] $$\frac{}{(x_0 := e_0, \sigma_0) \Downarrow (v_0, \sigma_0)}$$

[BS-ASSIGN] $$\frac{(e_0, \sigma_0) \Downarrow (v_0, \sigma_1)\ \ where\ \sigma_2 = \sigma_1(x.v_0)}{(x_0 := (e_0, \sigma_0)) \Downarrow (v_0, \sigma_2)}$$

[BS-OP] $$\frac{(e_0, \sigma_0) \Downarrow (v_0, \sigma_1)\ \ (e_1, \sigma_1) \Downarrow (v_1, \sigma_2)\ \ v_2 = v_0\ op\ v_1}{(e_0\ op\ e_1, \sigma_0) \Downarrow (v_2, \sigma_2)}$$

[BS-OPVAL] $$\frac{(e_1, \sigma_0) \Downarrow (v_1, \sigma_1)\ \ where\ v_2 = v_0\ op\ v_1}{(v_0\ op\ e_1, \sigma_0) \Downarrow (v_2, \sigma_1)}$$

[BS-SEQ] $$\frac{(e_0, \sigma_0) \Downarrow (v_0, \sigma_1)\ \ (e_1, \sigma_1) \Downarrow (v_1, \sigma_2)}{(e_0\ ;\ e_1, \sigma_0) \Downarrow (v_1\sigma_2)}$$

[BS-IFTRUE] $$\frac{(e_0, \sigma_0) \Downarrow True\ \ (e_1, \sigma_0) \Downarrow (v_0, \sigma_1)}{if(e_0\ e_1\ e_2, \sigma_0) \Downarrow (v_0, \sigma_1)}$$

[BS-IFFALSE] $$\frac{(e_0, \sigma_0) \Downarrow False\ \ (e_2, \sigma_0) \Downarrow (v_0, \sigma_1)}{if(e_0\ e_1\ e_2, \sigma_0) \Downarrow (v_0, \sigma_1)}$$

[BS-ANDTRUE] $$\frac{(e_0\sigma_0) \Downarrow (True, \sigma_1)\quad (e_1\sigma_1) \Downarrow (b, \sigma_2)\ \ where\ b\ is\ a\ boolean\ value}{(AND\ e_0\ e_1\ \sigma_0) \Downarrow\ (b, \sigma_2)}$$

[BS-ANDFALSE] $$\frac{(e_0\sigma_0) \Downarrow (False, \sigma_1)}{(AND\ e_0\ e_1\ \sigma_0) \Downarrow\ (False, \sigma_1)}$$

[BS-ORTRUE] $$\frac{(e_0\sigma_0) \Downarrow (True, \sigma_1)}{(OR\ e_0\ e_1\ \sigma_0) \Downarrow\ (True, \sigma_1)}$$

[BS-ORFALSE] $$\frac{(e_0\sigma_0) \Downarrow (False, \sigma_1)\quad (e_1\sigma_1) \Downarrow (b, \sigma_2)\ \ where\ b\ is\ a\ boolean\ value}{(OR\ e_0\ e_1\ \sigma_0) \Downarrow\ (b, \sigma_2)}$$

Figure 2: Big-step semantics for WHILE

$$[\text{BS-NOTTRUE}] \quad \frac{(e_0\sigma_0) \Downarrow (True, \sigma_1)}{(NOT\ e_0\ \sigma_0) \Downarrow\ (False, \sigma_1)}$$

$$[\text{BS-NOTFALSE}] \quad \frac{(e_0\sigma_0) \Downarrow (False, \sigma_1)}{(NOT\ e_0\ \sigma_0) \Downarrow\ (True, \sigma_1)}$$

$$[\text{BS-WHILEFALSE}] \quad \frac{(e_0, \sigma_0)\ \Downarrow\ (False, \sigma_1)}{(while(e_0)\ e_1, \sigma_0)\ \Downarrow\ (False\ \sigma_1)}$$

$$[\text{BS-WHILETRUE}] \quad \frac{(e_0, \sigma_0)\ \Downarrow\ (True, \sigma_1)\ (e_1, \sigma_1); \Downarrow\ (v_0, \sigma_2)\ (while(e_0)\ e_1, \sigma_0)\ \Downarrow\ (v_1\ , \sigma_3)}{(while(e_0)\ e_1, \sigma_0)\ \Downarrow\ (v_1\ \sigma_3)}$$

Figure 3: Big-step semantics for WHILE