Homework 2: Operational Semantics for the WHILE Language

Zayd Hammoudeh
(zayd.hammoudeh@sjsu.edu)

1 Introduction to the WHILE Language

The "WHILE" language is a basic language that was defined in class. Figure 1 defines the expressions, values, and operators in this language. This notation for expressions (e.g. e), values (v), variables/addresses (x), and store (σ) applies to all sections of this document.

$e ::=$		*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
	`not` e	not expressions
	`and` (e) (e)	and expressions (**Are the parentheses ok?**)
	`or` (e) (e)	or expressions (**Are the parentheses ok?**)
$v ::=$		*Values*
	i	integer values
	b	boolean values
$op ::=$	$+ \mid - \mid * \mid / \mid > \mid >= \mid < \mid <=$	*Binary operators*
σ		*Store*

Figure 1: The WHILE language

2 Base WHILE Language Small-Step Semantics Rules

The following figures enumerate the execution order, small-step semantics rules for the WHILE language expressions as defined in class.

Variable Evaluation Rules:

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

Figure 2: Variable Small-Step Semantics Evaluation Order Rules

Set/Assignment Evaluation Rules:

$$[\textsc{ss-assignContext}] \qquad \frac{e, \sigma \rightarrow e', \sigma'}{x := e, \sigma \rightarrow x := e', \sigma'}$$

$$[\textsc{ss-assignReduction}] \qquad \frac{}{x := v, \sigma \rightarrow v, \sigma[x := v]}$$

Figure 3: Set/Assignment Small-Step Semantics Evaluation Order Rules

Binary Operator (`op`) Evaluation Rules:

$$[\textsc{ss-opContext1}] \qquad \frac{e_1, \sigma \rightarrow e_1', \sigma'}{e_1 \ op \ e_2, \sigma \rightarrow e_1' \ op \ e_2, \sigma'}$$

$$[\textsc{ss-opContext2}] \qquad \frac{e, \sigma \rightarrow e', \sigma'}{v \ op \ e, \sigma \rightarrow v \ op \ e', \sigma'}$$

Is there a reason you used the infix op notation here instead of the notation from class "$\mathtt{v}_3 = \mathtt{apply}(\mathtt{op}, \mathtt{v}_1, \mathtt{v}_2)$"

$$[\textsc{ss-opReduction}] \qquad \frac{v_3 = v_1 \ op \ v_2}{v_1 \ op \ v_2, \sigma \rightarrow v_3, \sigma}$$

Figure 4: Binary Operator (`op`) Evaluation Order Rules

Sequence (`;`) Evaluation Rules:

$$[\textsc{ss-seqContext}] \qquad \frac{e_1, \sigma \rightarrow e_1', \sigma'}{e_1; e_2, \sigma \rightarrow e_1'; e_2, \sigma'}$$

$$[\textsc{ss-seqReduction}] \qquad \frac{}{v; e, \sigma \rightarrow e, \sigma}$$

Figure 5: Sequence (`;`) Evaluation Order Rules

Conditional Statement (`if`) Evaluation Rules:

$$[\textsc{ss-ifContext}] \qquad \frac{e_1, \sigma \to e_1', \sigma'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, \sigma \to \texttt{if } e_1' \texttt{ then } e_2 \texttt{ else } e_3, \sigma'}$$

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

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

Figure 6: Conditional (`if`) Small-Step Semantics Evaluation Order Rules

`while` **Evaluation Rules:**

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

Figure 7: `while` Small-Step Semantics Evaluation Order Rules

3 Boolean Expressions Small-Step Semantics Rules

In this section, I add three new expression types to the WHILE language namely: `not`, `and`, and `or`. The evaluation order rules for each are below.

Do I need the parentheses is in the "`and`" and "`or`" statements? Is an infix style more typically used?

`not` **Evaluation Rules:**

$$[\textsc{ss-notContext}] \qquad \frac{e, \sigma \rightarrow e', \sigma'}{\texttt{not}\ e, \sigma \rightarrow \texttt{not}\ e', \sigma'}$$

Not sure if I need this. If I do, then why? Why is this not like the "`op`" case?

$$[\textsc{ss-notReduction}] \qquad \frac{}{\texttt{not}\ v, \sigma \rightarrow \texttt{if}\ v\ \texttt{then false else true}, \sigma}$$

I believe the above rule makes these unnecessary. Would most define as above or like below (assuming they are even necessary)?

$$[\textsc{ss-notTrue}] \qquad \frac{}{\texttt{not true}, \sigma \rightarrow \texttt{false}, \sigma}$$

$$[\textsc{ss-notFalse}] \qquad \frac{}{\texttt{not false}, \sigma \rightarrow \texttt{true}, \sigma}$$

Figure 8: `not` Small-Step Semantics Evaluation Order Rules

`and` **Evaluation Rules:**

$$[\textsc{ss-andContext}] \qquad \frac{e_1, \sigma \rightarrow e_1', \sigma'}{\texttt{and}\ (e_1)\ (e_2), \sigma \rightarrow \texttt{and}\ (e_1')\ (e_2), \sigma'}$$

$$[\textsc{ss-andReduction}] \qquad \frac{}{\texttt{and}\ (v)\ (e), \sigma \rightarrow \texttt{if}\ v\ \texttt{then}\ e\ \texttt{else false}, \sigma}$$

Using the above, I think I do not need these. However, I believe the implementation from an execution perspective of these is slightly different since the above case is short circuit compare (which could affect the store) while the lower case is not. Correct me if I am wrong.

$$[\textsc{ss-andContext2}] \qquad \frac{e, \sigma \rightarrow e', \sigma'}{\texttt{and}\ (v)\ (e), \sigma \rightarrow \texttt{and}\ (v)\ (e'), \sigma'}$$

$$[\textsc{ss-andAllTrue}] \qquad \frac{}{\texttt{and}\ (\texttt{true})\ (\texttt{true}), \sigma \rightarrow \texttt{true}, \sigma}$$

$$[\textsc{ss-andFalse1}] \qquad \frac{}{\texttt{and}\ (\texttt{false})\ (v), \sigma \rightarrow \texttt{false}, \sigma}$$

$$[\textsc{ss-andFalse2}] \qquad \frac{}{\texttt{and}\ (v)\ (\texttt{false}), \sigma \rightarrow \texttt{false}, \sigma}$$

Figure 9: `and` Small-Step Semantics Evaluation Order Rules

`or` **Evaluation Rules: (Is defining "temporary variables" as I did allowed in small step semantics? I assumed it was because of how you handled the "`op`" case. I also assumed that defining these temp variables is required since they enforce the evaluation order (correct me if I am wrong).**

$$[\textsc{ss-orReduction}] \qquad \frac{e_1' = \texttt{not}\ e_1 \qquad e_2' = \texttt{not}\ e_2 \qquad e_3 = \texttt{and}\ (e_1')\ (e_2')}{\texttt{or}\ (e_1)\ (e_2), \sigma \rightarrow \texttt{not}\ e_3, \sigma}$$

Figure 10: `or` Evaluation Order Rule