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), 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 Rule:

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

Figure 2: Variable Small-Step Semantics Evaluation Order Rule

Set/Assignment Evaluation Rules:

$$[\text{SS-ASSIGNCONTEXT}] \qquad \frac{e, \sigma \to e', \sigma'}{x := e, \sigma \to x := e', \sigma'}$$

$$[\text{SS-ASSIGNREDUCTION}] \qquad \frac{}{x := v, \sigma \to v, \sigma[x := v]}$$

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

Binary Operator (`op`) Evaluation Rules:

$$[\text{SS-OPCONTEXT1}] \qquad \frac{e_1, \sigma \to e_1', \sigma'}{e_1 \ op \ e_2, \sigma \to e_1' \ op \ e_2, \sigma'}$$

$$[\text{SS-OPCONTEXT2}] \qquad \frac{e, \sigma \to e', \sigma'}{v \ op \ e, \sigma \to v \ op \ e', \sigma'}$$

Is there a reason you used the infix op notation here instead of the notation from class "$\mathtt{v_3} = \mathtt{apply}(\mathtt{op}, \mathtt{v_1}, \mathtt{v_2})$"

$$[\text{SS-OPREDUCTION}] \qquad \frac{v_3 = v_1 \ op \ v_2}{v_1 \ op \ v_2, \sigma \to v_3, \sigma}$$

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

Sequence (`;`) Evaluation Rules:

$$[\text{SS-SEQCONTEXT}] \qquad \frac{e_1, \sigma \to e_1', \sigma'}{e_1; e_2, \sigma \to e_1'; e_2, \sigma'}$$

$$[\text{SS-SEQREDUCTION}] \qquad \frac{}{v; e, \sigma \to e, \sigma}$$

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

Conditional Statement (`if`) Evaluation Rules:

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

[SS-IFTRUEREDUCTION]
$$\frac{}{\texttt{if true then } e_1 \texttt{ else } e_2, \sigma \to e_1, \sigma}$$

[SS-IFFALSEREDUCTION]
$$\frac{}{\texttt{if false then } e_1 \texttt{ else } e_2, \sigma \to e_2, \sigma}$$

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

`while` **Evaluation Rule:**

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

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

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.

Can I have the parentheses in the "`and`" and "`or`" statements? Is an infix style more typically used?

`not` **Evaluation Rules:**

$$[\text{SS-NOTCONTEXT}] \qquad \frac{e, \sigma \to e', \sigma'}{\texttt{not}\ e, \sigma \to \texttt{not}\ e', \sigma'}$$

Not sure if I need this. If I do, then why? Why is this not like the "`op`" case? Can this be used to enforce that the value types are boolean?

$$[\text{SS-NOTREDUCTION}] \qquad \frac{}{\texttt{not}\ v, \sigma \to \texttt{if}\ v\ \texttt{then false else true}, \sigma}$$

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

$$[\text{SS-NOTTRUE}] \qquad \frac{}{\texttt{not true}, \sigma \to \texttt{false}, \sigma}$$

$$[\text{SS-NOTFALSE}] \qquad \frac{}{\texttt{not false}, \sigma \to \texttt{true}, \sigma}$$

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

`and` **Evaluation Rules: (In the case of rule "SS-ANDREDUCTION", "e" could return an integer. I do not enforce any typing here while I do in the lower set of rules. Is that ok? Rather than just making it "e", I could make it "`AND e True`")**

$$[\text{SS-ANDCONTEXT}] \qquad \frac{e_1, \sigma \rightarrow e_1', \sigma'}{\texttt{and}\ (e_1)\ (e_2), \sigma \rightarrow \texttt{and}\ (e_1')\ (e_2), \sigma'}$$

$$[\text{SS-ANDREDUCTION}] \qquad \frac{}{\texttt{and}\ (v)\ (e), \sigma \rightarrow \texttt{if}\ v\ \texttt{then}\ e\ \texttt{else}\ \texttt{false}, \sigma}$$

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.

$$[\text{SS-ANDCONTEXT2}] \qquad \frac{e, \sigma \rightarrow e', \sigma'}{\texttt{and}\ (v)\ (e), \sigma \rightarrow \texttt{and}\ (v)\ (e'), \sigma'}$$

$$[\text{SS-ANDALLTRUE}] \qquad \frac{}{\texttt{and}\ (\texttt{true})\ (\texttt{true}), \sigma \rightarrow \texttt{true}, \sigma}$$

$$[\text{SS-ANDFALSE1}] \qquad \frac{}{\texttt{and}\ (\texttt{false})\ (v), \sigma \rightarrow \texttt{false}, \sigma}$$

$$[\text{SS-ANDFALSE2}] \qquad \frac{}{\texttt{and}\ (v)\ (\texttt{false}), \sigma \rightarrow \texttt{false}, \sigma}$$

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

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

$$[\text{SS-ORREDUCTION}] \qquad \frac{e_1' = \texttt{not}\ e_1 \qquad e_2' = \texttt{not}\ e_2 \qquad e_3 = \texttt{and}\ (e_1')\ (e_2')}{\texttt{or}\ (e_1)\ (e_2), \sigma \rightarrow \texttt{not}\ e_3, \sigma}$$

Figure 10: `or` Small-Step Semantics Evaluation Order Rule