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, operators, and store in this language; the 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
	`or` (e) (e)	or expressions
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

In the WHILE language, the base relation takes the format shown in figure 2.

$$e, \sigma \rightarrow e', \sigma'$$

Figure 2: WHILE Language Evaluation Relation

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

Variable Evaluation Rule:

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

Figure 3: 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 4: Set/Assignment Small-Step Semantics Evaluation Order Rules

Binary Operator (`op`) Evaluation Rules:

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

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

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

Figure 5: 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 6: 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 7: 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 8: `while` Small-Step Semantics Evaluation Order Rule

3 Boolean Expressions Small-Step Semantics Rules

In following subsections, I describe three additional expression types in the updated the WHILE language namely: `not`, `and`, and `or`.

3.1 `not` Expression

`not` in my modified version of the WHILE language behaves as a standard Boolean `not`. It takes a single Boolean value and returns its complement. If an expression is passed, the language simplifies that expression until it is in normal form at which point it applies the Boolean `not`.

`not` **Evaluation Rule:**

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

Figure 9: `not` Small-Step Semantics Evaluation Order Rule

3.2 `and` Expression

`and` is designed to mimic the Boolean `and` with the exception that it supports short circuit compare. Hence, if the first expression in the `and` evaluates to `false`, the second parameter is not evaluated at all.

`and` **Evaluation Rules:**

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

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

Figure 10: `and` Small-Step Semantics Evaluation Order Rules

3.3 `or` Expression

`or` is a composite of the expressions "`not`" and "`and`" described in sections 3.1 and 3.2 respectively.

`or` **Evaluation Rule:**

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

Figure 11: `or` Small-Step Semantics Evaluation Order Rule