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 discussed 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

The store (σ) is a container that holds key value pairs of variables to values. It is defined via the notation shown in figure 2.

$$\sigma \in Store \quad = \quad variable \ \rightarrow \ v$$

Figure 2: Store (σ) in the WHILE Language

2 Unextended WHILE Language Small-Step Semantics Rules

In the WHILE language, the evaluation relation takes the form shown in figure 3.

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

Figure 3: WHILE Language Evaluation Relation

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

2.1 Variable Expression

The variable expression is used to specify a key, which should correspond to a specific value in the store. Note that it is possible for the key (i.e. variable) to not exist in the store.

Variable Evaluation Rule:

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

Figure 4: Variable Small-Step Semantics Evaluation Order Rule

2.2 Set/Assign Expression

The set/assignment expression is the complement of the variable expression. In contrast to variable which extracts a value from the store, set/assign expression specifies a key (variable) and value, the pair of which is preserved in the store.

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 5: Set/Assignment Small-Step Semantics Evaluation Order Rules

2.3 Binary Operator Expression

By definition, binary operators take exactly two input parameters and return a result. Figure 6 defines the binary operator small-step evaluation order rules for the WHILE language. Also, note that the supported set of binary operators is specified in figure 1.

Binary Operator (`op`) Evaluation Rules:

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

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

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

Figure 6: Binary Operator (`op`) Evaluation Order Rules

2.4 Sequence Expression

The sequence expression is used when two or more distinct expressions need to be executed in a specific order; as such, it defines which of the set of specified expressions has precedence. The rules in figure 7 are for two expressions, but this is extensible to an arbitrary number of expressions by chaining.

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 7: Sequence (`;`) Evaluation Order Rules

2.5 Conditional (`if`) Expression

The conditional (`if`) in the WHILE language is similar to that of other languages in that it takes a Boolean value (e.g. `true`or `false`) and returns one of two possible results depending on that Boolean value. The rules for `if` are in figure 8.

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

2.6 While Expression

Similar to the `while` construct in other programming languages, the `while` expression in this language takes two expressions and will evaluate the second expression as long as the first expression evaluates to `true`.

`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 9: `while` Small-Step Semantics Evaluation Order Rule

3 Boolean Expression Small-Step Semantics Rules for an Extended WHILE Language

In following subsections, I describe three additional expression types in the updated/extended 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:

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

Figure 10: 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:

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

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

Figure 11: 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:

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

Figure 12: or Small-Step Semantics Evaluation Order Rule