# 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 enumerates the expressions, values, operators, and store in this language; the notation for expressions (e), values (v), variables/addresses (x), and store  $(\sigma)$  applies to all sections of this document.

```
Expressions
e ::=
                                                          variables/addresses
           x
                                                                        values
                                                                  assignment
           x := e
                                                       sequential expressions
           e; e
                                                           binary operations
           e op e
                                                      conditional expressions
           if e then e else e
           while (e) e
                                                            while expressions
                                                              not expressions
           \mathtt{not}\ e
           and (e) (e)
                                                             and expressions
           or (e) (e)
                                                               or expressions
v ::=
                                                                       Values
                                                                integer values
                                                               boolean values
           + | - | * | / | > | >= | < | <=
                                                            Binary operators
op ::=
                                                                         Store
\sigma
```

Figure 1: The WHILE language

The store  $(\sigma)$  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 = variable \rightarrow v
```

**Figure 2:** Store  $(\sigma)$  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.

$$e,\sigma o e',\sigma'$$

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. The small-step evaluation order semantic rules for this expression type is enumerated in figure 4.

Variable Evaluation Rule: 
$$\frac{x \in domain(\sigma) \qquad \sigma(x) = v}{x, \sigma \to v, \sigma}$$

Figure 4: Variable Small-Step Semantics Evaluation Order Rule

# 2.2 Set/Assignment 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. The small-step evaluation order semantic rules for this expression type are enumerated in figure 5.

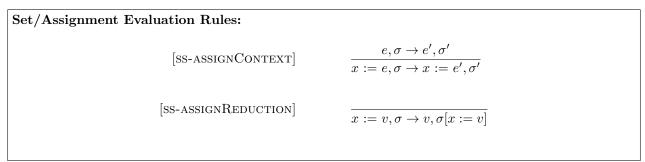


Figure 5: Set/Assignment Small-Step Semantics Evaluation Order Rules

# 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.

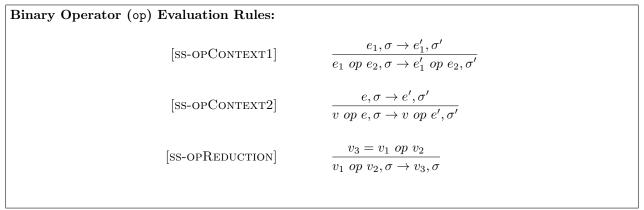


Figure 6: Binary Operator (op) Evaluation Order Rules

In the accompanying program that implements these rules, only integer values are supported for these binary operators.

# 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 (e.g.  $e_1$ ,  $e_2$ ), but this is extensible to an arbitrary number of expressions by chaining multiple sequences.

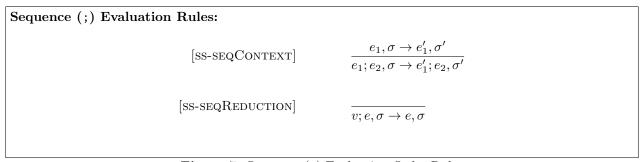


Figure 7: Sequence (;) Evaluation Order Rules

## 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.

# 

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. The reduction rule for this expression is in figure 9.

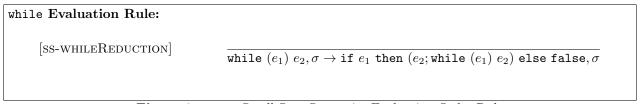


Figure 9: while Small-Step Semantics Evaluation Order Rule

# 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. The evaluation order, small step semantic rule for not is in figure 10.

# not Evaluation Rule: $[{\rm SS\text{-}NOTREDUCTION}] \qquad \qquad \overline{{\rm not}\ e,\sigma \to {\rm if}\ e\ {\rm then\ false\ else\ true},\sigma}$

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. The behavior of and is shown in figure 11.

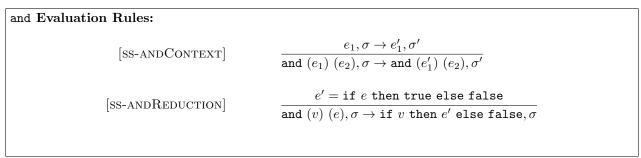


Figure 11: and Small-Step Semantics Evaluation Order Rules

# 3.3 or Expression

or is a composite of the expressions "not" and "and" described in sections 3.1 and 3.2 respectively. Its behavior is detailed in figure 12.

or Evaluation Rule: 
$$\frac{e_1' = \text{not } e_1 \quad e_2' = \text{not } e_2 \quad e_3 = \text{and } (e_1') \ (e_2') }{ \text{or } (e_1) \ (e_2), \sigma \to \text{not } e_3, \sigma }$$

Figure 12: or Small-Step Semantics Evaluation Order Rule