# 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  $(\sigma)$  applies to all sections of this document.

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

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 
ightarrow 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:

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

Figure 3: Variable Small-Step Semantics Evaluation Order Rule

# Set/Assignment Evaluation Rules:

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

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

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

# Binary Operator (op) Evaluation Rules:

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

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

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

Figure 5: Binary Operator (op) Evaluation Order Rules

# Sequence (;) Evaluation Rules:

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

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

Figure 6: Sequence (;) Evaluation Order Rules

# 

Figure 7: Conditional (if) Small-Step Semantics Evaluation Order Rules

# while Evaluation Rule: $[\text{SS-WHILEREDUCTION}] \qquad \qquad \overline{\text{while } (e_1) \ e_2, \sigma \to \text{if } e_1 \text{ then } (e_2; \text{while } (e_1) \ e_2) \text{ else false}, \sigma}$

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.

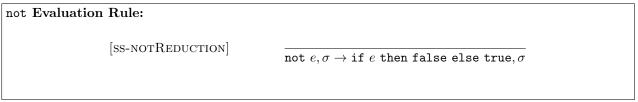


Figure 9: not Small-Step Semantics Evaluation Order Rule

# 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: $\frac{e_1, \sigma \to e_1', \sigma'}{\text{and } (e_1) \ (e_2), \sigma \to \text{and } (e_1') \ (e_2), \sigma'}$ $\frac{e' = \text{if } e \text{ then true else false}}{\text{and } (v) \ (e), \sigma \to \text{if } v \text{ then } e' \text{ else false}, \sigma}$

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

Figure 11: or Small-Step Semantics Evaluation Order Rule