## Homework 2: Operational Semantics for WHILE

CS 252: Advanced Programming Languages Parth Jayantilal Jain San José State University

## 1 Introduction

For this assignment, you will implement the semantics for a small imperative language, named WHILE.

The language for WHILE is given in Figure 1. Unlike the Bool\* language we discussed previously, WHILE supports *mutable references*. The state of these references is maintained in a *store*, a mapping of references to values. ("Store" can be thought of as a synonym for heap.) Once we have mutable references, other language constructs become more useful, such as sequencing operations  $(e_1; e_2)$ .

## 2 Small-step semantics

The small-step semantics for WHILE are given in Figure 3. For the sake of brevity, these rules use *evaluation* contexts (C), which specify which redex will be evaluated next. The evaluation rules then apply to the "hole"  $(\bullet)$  in this context.

Most of these rules are fairly straightforward, but there are a couple of points to note with the [SS-WHILE] rule. First of all, this is the only rule that makes a more complex expression when it has finished.

Secondly, note the final value of this expression once the while loop completes. It will *always* be false when it completes. We could have created a special value, such as null, or we could have made the while loop a statement that returns no value. Both choices, however, would complicate our language needlessly.

## 3 YOUR ASSIGNMENT

Part 1: Rewrite the operational semantic rules for WHILE in LATEX to remove the contexts (C[...]) and to use evaluation order rules instead. Submit both your LATEX source and the generated PDF file.

Extend your semantics with features to handle boolean values. Specifically, add support for:

- $\bullet$  and
- or
- not

The exact behavior of these new features is up to you, but should seem reasonable to most programmers.

Part 2: Once you have your semantics defined, download WhileInterp.hs and implement the evaluate function, as well as any additional functions you need. Your implementation must be consistent with your operational semantics, *including your extensions for* and, or, *and* not. Also, you may not change any type signatures provided in the file.

Finally, implement the interpreter to match your semantics.

Zip all files together into hw2.zip and submit to Canvas.

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

Figure 1: The WHILE language

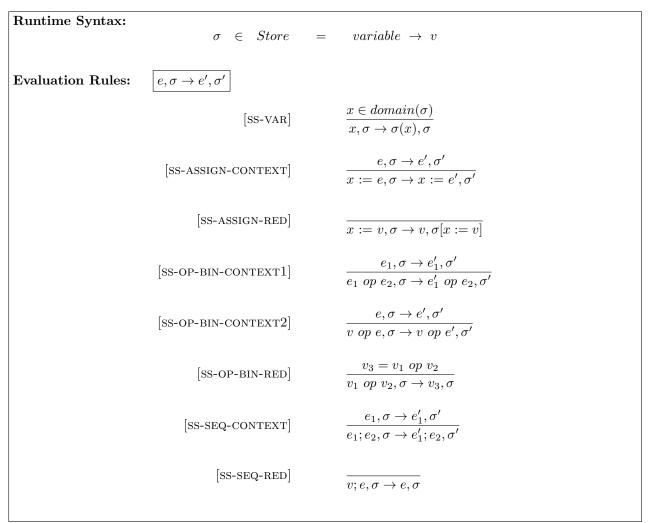


Figure 2: Small-step semantics for WHILE

[SS-IFCONTEXT]	$e_1,\sigma o e_1',\sigma'$ if $e_1$ then $e_2$ else $e_3,\sigma o$ if $e_1'$ then $e_2$ else $e_3,\sigma'$
[SS-IFTRUE]	$\overline{ ext{if true then }e_1 ext{ else }e_2,\sigma{ ightarrow}e_1,\sigma}$
[SS-IFFALSE]	$\overline{ ext{if false then }e_1 ext{ else }e_2,\sigma o e_2,\sigma}$
[SS-WHILE]	$\overline{\text{while } (e_1) \; e_2, \sigma \to \text{if } e_1 \; \text{then } e_2; \text{while } (e_1) \; e_2 \; \text{else false}, \sigma}$
[SS-AND-CONTEXT]	$rac{e_1,\sigma ightarrow e_1',\sigma'}{ ext{and }e_1\ e_2,\sigma ightarrow  ext{and }e_1'\ e_2,\sigma'}$
[SS-AND-FALSE]	and false $e,\sigma  o \mathtt{false},\sigma$
[SS-AND-TRUE]	and true $e, \sigma \to e, \sigma$
[SS-OR-CONTEXT]	$rac{e_1,\sigma ightarrow e_1',\sigma'}{ ext{or }e_1\;e_2,\sigma ightarrow  ext{or }e_1'\;e_2,\sigma'}$
[SS-OR-FALSE]	$\overline{ ext{or false}\;e,\sigma o e,\sigma}$
[SS-OR-TRUE]	$\overline{ ext{or true } e, \sigma  o  ext{true}, \sigma}$
[SS-NOT-CONTEXT]	$\frac{e,\sigma \to e',\sigma'}{not\ e,\sigma \to not\ e',\sigma'}$
[SS-NOT-FALSE]	$\overline{\mathtt{not}\;\mathtt{false},\sigma  o \mathtt{true},\sigma}$
[SS-NOT-TRUE]	$\overline{\mathtt{not}\;\mathtt{true},\sigma o\mathtt{false},\sigma}$

Figure 3: Small-step semantics for WHILE (Continued)