HW 7 **Prof. Caldwell**
Due: 7 April 2016 **COSC 4780**

1 Reading

Problem 1.1. Read chapter 2 of Schmidt pp. 31-40.

2 Reasoning about `while` statements.

Recall the semantics of the while command.

Definition 2.1.

$$\begin{array}{l}[\![\textbf{while } E \textbf{ do } C \textbf{ od} : comm]\!](s) = w(s) \\ \quad \text{where } w(s) = \textbf{if}([\![E : boolexp]\!](s), w([\![C : comm]\!](s)), s)\end{array}$$

Schmidt defines the meaning of w by a family of functions as follows:

$$\begin{array}{l} w_0(s) = \bot \\ w_1(s) = \mathit{if}([\![E : boolexp]\!](s), w_0([\![C : comm]\!](s)), s) \\ w_2(s) = \mathit{if}([\![E : boolexp]\!](s), w_1([\![C : comm]\!](s)), s) \\ w_3(s) = \mathit{if}([\![E : boolexp]\!](s), w_2([\![C : comm]\!](s)), s) \\ \cdots \\ w_{i+1}(s) = \mathit{if}([\![E : boolexp]\!](s), w_i([\![C : comm]\!](s)), s) \\ \cdots \end{array}$$

The following theorem is stated implicitly in Schmidt (pp.15) about the recursive function w used to defined the semantics of the `while` command.

Theorem 2.1.

$$\forall s, s' : Store.\quad w(s) = s' \quad \Leftrightarrow \quad \exists k : \mathbb{N}.\ w_k(s) = s'$$

By the least element principle you may assume that if such a k exists then there is a least such k.

As an application of this theorem we stated and proved the following theorem in class.

Theorem 2.2. $\forall s : Store.\ \forall C : comm \quad [\![\textbf{while } 0 = 0 \textbf{ do } C \textbf{ od}]\!](s) = \bot$.

Proof: Choose an arbitrary store s. We argue by contradiction. Assume that $w(s) = s'$ for some $s' \in \mathit{Store}$. Then, by Theorem **??**, there is some least k such that $w_k(s) = s'$. But then, since k is the least such integer, by the definition of w_k we know that $w_k(s) = w_1(s') = s'$. By definition of w_1,

$$w_1(s') = if([\![0 = 0 : \mathit{boolexp}]\!](s'), w_0([\![C : \mathit{comm}]\!](s'), s')$$

Since for every store s, $[\![0 = 0 : \mathit{boolexp}]\!](s) = \mathit{true}$ we know that $w_1(s') = w_0([\![C : \mathit{comm}]\!](s')) = \bot$. Thus $w_k(s) = \bot$. This contradicting the assumption that $w(s) = s'$ for some store s'.
□

Problem 2.1. Do problem 6.2a and 6.2c on page 25.

To prove 6.2c choose an arbitrary Store s and then evaluate the semantic equations one step using the semi-colon rule for sequencing commands. Then consider the cases of whether there is a store s' such that

$[\![\textbf{while } E \textbf{ do } C_1 \textbf{ od} : \mathit{comm}]\!](s) = s'$ or whether $[\![\textbf{while } E \textbf{ do } C_1 \textbf{ od} : \mathit{comm}]\!](s) = \bot$

If the meaning of the while is a proper store use the theorem stated above. Hint: If there is a least $k \in \mathbb{N}$ such that $w_k(s) = s'$ then what do you know about $[\![E : \mathit{boolexp}]\!](s')$??

3 Static Analysis

We can analyze a program to see if it satisfies a particular property. An analysis of a program is *static* if it is analyzed without evaluating it; by solely examining the program text or the abstract syntax tree. A *dynamic analysis* is an analysis that proceeds by observing an execution (possibly many) of a program.

In class we noted that the semantics can be used to reason about programs. Already in the core language we can do some simple heuristic static analysis to try to catch looping programs. From Thm. **??** we know that

$$[\![\texttt{while } 0 = 0 \texttt{ do } C \texttt{ od} : \mathit{comm}]\!](s) = \bot$$

Can we generalize this observation to try to design a piece of code than analyzes programs in the core language to detect possible loops? Of course, we

will not be able to detect all looping behavior, this would be the equivalent of solving the halting problem. But given a while statement of the form

$$\texttt{while } E \texttt{ do } C \texttt{ od} : comm$$

When do we know it will loop forever? Certainly, if $[\![E : boolexp]\!](s) = true$ for all s. But determining whether E evaluates to the constant $true$ in all stores could be difficult. Lets consider something easier. If $[\![E : boolexp]\!](s) = true$ for some store s and the command C does not update any location referenced by E, then the program must loop if started in store s. And also, if $[\![E : boolexp]\!](s) = false$, then this entire statement is equivalent to `skip`. So, if we could, for while statements, check to see of the locations updated by the body C are disjoint from the locations referenced in the expression E, we would have identified a possible looping statement.

Definition 3.1 (active) We will say a location is *active* in a command C if it occurs on the left side of an assignment statement in C.

Definition 3.2 (Referenced) We say a location is *referenced* in an expression if it occurs in the expression.

Problem 3.1. Write functions `exp_refs` and `active_locs` that recursively collect lists of location references in expressions and a lists of active locations in a command. Write a function `loops_analysis` that reports possible looping behavior in a program by checking all while statements to see if the set of location referenced in the condition overlap with the active locations in the body. If so, report it `Good`. If not, report the violating while statement as `Possible` $[C_1, \cdots, C_k]$. Where the commands in the list are the violating While statements.

You can use the following type to encode return values form the analyze function.

```
type report = Good | Possible of command list;;
```

I found a function to union reports together useful; I included it in the source code supplied with this homework.

Here are the expected types of your functions:

```
exp_refs : expression -> loc list
active_locs : command -> loc list
loops_analysis : command -> report
```

You should add some tests cases that show that your code really recursivley descends though more complex commands and expressions.