CS 6371	**Advanced Programming Languages**	Assignment 3:
Spring 2020	Due: Feb 11, 2020 by 1:05pm	Operational Semantics

1 Introduction

This written assignment is to be turned in via eLearning or in hardcopy at the beginning of class on February 11, 2020 (no later than 1:05pm). If in hardcopy, please use **standard letter** ($8.5'' \times 11''$) **paper** with no torn edges (so it can survive a photocopier). If hand-written, please write legibly; if typed, please clearly identify any non-standard notational conventions you've adopted for typesetting symbols, derivation trees, and other mathematics. Remember to put your name and email address at the top of each page.

2 Problem Set

(1) (8 pts) Let $\sigma : v \rightharpoonup \mathbb{Z}$ be a store in which $\sigma(\texttt{i}) = 1$. Prove that when started in memory state σ the program `while i*i<=i do i:=i+1` eventually terminates and results in a memory state σ' in which $\sigma'(\texttt{i}) = 2$. (The derivation that proves this is a bit long. You may divide it into several pieces if needed, to make it easier to read.)

(2) (5 pts) Define $W =$ (`while true do x:=x+2`) and $\sigma = \{(\texttt{x}, 0)\}$. List the first 6 configurations in the small-step evaluation of $\langle W, \sigma \rangle$ (not including configuration $\langle W, \sigma \rangle$ itself). You do not need to show a derivation of each small-step; just write the configuration that results after each small-step. (Note that showing the first 6 small-step configurations is *not* the same as running the loop body 6 times; you must *use the small-step semantics* to figure out what the first 6 configurations are.)

(3) (10 pts) Assume that a is a SIMPL arithmetic expression in which there are exactly k "`+`" symbols, k "`-`" symbols, and k "`*`" symbols, and in which all integer constants are 2. Assume also that $\sigma^{\rightarrow} = \{2\}$ (where notation $\sigma^{\rightarrow}$ denotes the image of σ). Prove by structural induction that if judgment $\langle a, \sigma \rangle \Downarrow n$ holds, then $|n| \leq 2^{3k+1}$.

(4) Consider the following extension to SIMPL's syntax, which introduces *side-effects* to arithmetic expressions:

$$a ::= n \mid v \mid a_1 \texttt{+} a_2 \mid a_1 \texttt{-} a_2 \mid a_1 \texttt{*} a_2 \mid (c, a)$$

The intended semantics of the side-effect expression (c, a) is that first command c is executed, then expression a is evaluated and the result is returned as the result of the whole expression. For example, the command `x:=((x:=1), x)+((x:=x+1), x)` should first assign 1 to `x`, then assign 2 to `x`, and finally assign 3 to `x`.

(a) (11 pts) Rewrite the large-step operational semantics of SIMPL to support these new side-effect expressions. That is, write large-step semantics rules that formally define the behavior of expression (c, a). If you need to modify any of the existing large-step rules, write those modified rules as well.

Hint: You will need to modify judgments $\langle a, \sigma \rangle \Downarrow n$ and $\langle b, \sigma \rangle \Downarrow p$ so that they also return a store. For example, you might write $\langle a, \sigma \rangle \Downarrow n, \sigma'$ to denote that arithmetic expression a returns integer n and a new store σ'.

(b) (3 pts) Extend the small-step operational semantics of SIMPL to support side-effect expressions. That is, write small-step semantics rules that formally define the behavior of expression (c, a).

Hint: This is much easier than part a. You should not need to modify any of the existing small-step rules.