DCC888 – Operational Semantics [1]

Name: _____ ID: _____

1. Consider the grammar below, which generates boolean expressions:

$$\begin{array}{rcl} \langle E \rangle & ::= & \langle E \rangle \ \mathit{and} \ \langle E \rangle \\ & | & \langle E \rangle \ \mathit{or} \ \langle E \rangle \\ & | & \mathit{true} \\ & | & \mathit{false} \end{array}$$

   Prove that any term that this grammar produces has more operands than operators. Operands are *true* and *false*, and operators are *and* and *or*.

2. We can define the multi-step evaluation relation $\rightarrow^*$ as the reflexive, transitive closure of one-step evaluation. The multi-step evaluation relation must meet the following properties:

   - If $t \rightarrow t'$, then $t \rightarrow^* t'$.
   - $t \rightarrow^* t$ for any term t.
   - If $t \rightarrow^* t'$ and $t' \rightarrow^* t$", then $t \rightarrow^* t$".

   In this exercise, you must redefine the multi-step evaluation relation as a set of inference rules.

[1] Some of these exercises have been taken from the book *Types and Programming Languages* by Benjamin Pierce.

3. There exists another style of operational semantics that is very popular. It is called *big-step operational semantics*, in contrast with the *small-step style* that we have seen. The big step evaluation rules for the language of boolean and arithmetic expressions are given below:

$$[\text{B-Value}] \qquad v \Downarrow v$$

$$[\text{B-IfTrue}] \qquad \frac{t_1 \Downarrow \mathit{true} \qquad t_2 \Downarrow v_2}{\mathit{if}\ t_1\ \mathit{then}\ t_2\ \mathit{else}\ t_3 \Downarrow v_2}$$

$$[\text{B-IfFalse}] \qquad \frac{t_1 \Downarrow \mathit{false} \qquad t_3 \Downarrow v_3}{\mathit{if}\ t_1\ \mathit{then}\ t_2\ \mathit{else}\ t_3 \Downarrow v_3}$$

$$[\text{B-Succ}] \qquad \frac{t_1 \Downarrow v_1}{\mathit{succ}\ t_1 \Downarrow \mathit{succ}\ v_1}$$

$$[\text{B-PredZero}] \qquad \frac{t_1 \Downarrow 0}{\mathit{pred}\ t_1 \Downarrow 0}$$

$$[\text{B-PredSucc}] \qquad \frac{t_1 \Downarrow \mathit{succ}\ v_1}{\mathit{pred}\ t_1 \Downarrow v_1}$$

$$[\text{B-IsZeroZero}] \qquad \frac{t_1 \Downarrow 0}{\mathit{iszero}\ t_1 \Downarrow \mathit{true}}$$

$$[\text{B-IsZeroSucc}] \qquad \frac{t_1 \Downarrow \mathit{succ}\ v_1}{\mathit{iszero}\ t_1 \Downarrow \mathit{false}}$$

In this exercise, you must show that the big-step and the small-step evaluation rules coincide. In other words, you must prove that $t \rightarrow^* v$ if, and only if, $t \Downarrow v$.

4. Lets us add two new semantic rules to the simple language which we saw in the previous exercise. These rules are meant to handle boolean negations:

$$[\text{B-NegTrue}] \qquad \frac{t \Downarrow \mathit{true}}{\mathit{not}\ t \Downarrow \mathit{false}}$$

$$[\text{B-NegFalse}] \qquad \frac{t \Downarrow \mathit{false}}{\mathit{not}\ t \Downarrow \mathit{true}}$$

Given these two rules, prove the following equivalence relation in our language: " *if t then true else false* $\equiv$ *if not t then false else true*". You must use structural induction on the derivation rules.

5. The next question is about the language below, which emulates a simple assembly instruction set:

(Program)	::=	*Instruction**
(Instruction)	::=	
– (Assignment)	|	$\texttt{mov}(v_1, v_2)$
– (Addition)	|	$\texttt{add}(v_1, v_2, v_3)$
– (Memory store)	|	$\texttt{stm}(v_0, v_1)$
– (Memory load)	|	$\texttt{ldm}(v_1, v_0)$
– (Branch if zero)	|	$\texttt{bzr}(v, l)$
– (Unconditional jump)	|	$\texttt{jmp}(l)$

The state of a program, in this language, is described by a three elements tuple $(P, \texttt{pc}, \Sigma)$. P is an array of assembly instructions, representing the program itself.

Notice that P will never change during the execution of the program. Thus, technically it is not "state", but we include it in our state tuple for readability. The register `pc` is a counter that indexes instructions in P. Finally, Σ is a function that associates variable names with integer values. Some instructions change `pc`, others change Σ. The semantics of some of these instructions is given below. We let $f[a \mapsto b]$ denote $\lambda x.\mathit{if}\ x = a\ \mathit{then}\ b\ \mathit{else}\ f(x)$.

$$[\textsc{AddSem}]\quad \frac{P[\texttt{pc}] = \texttt{add}(v_1, v_2, v_3) \qquad \Sigma[v_2] = n_2 \qquad \Sigma[v_3] = n_3 \qquad \Sigma' = \Sigma[v_1 \mapsto (n_2 + n_3)] \qquad \langle P, \texttt{pc} + 1, \Sigma' \rangle \to \Sigma''}{\langle P, \texttt{pc}, \Sigma \rangle \to \Sigma''}$$

$$[\textsc{StmSem}]\quad \frac{P[\texttt{pc}] = \texttt{stmem}(v_0, v_1) \qquad \Sigma[v_0] = x \qquad \Sigma[v_1] = n \qquad \Sigma' = \Sigma[x \mapsto n] \qquad \langle P, \texttt{pc} + 1, \Sigma' \rangle \to \Sigma''}{\langle P, \texttt{pc}, \Sigma \rangle \to \Sigma''}$$

$$[\textsc{BzrSem}]\quad \frac{P[\texttt{pc}] = \texttt{bzr}(v, l) \qquad \Sigma[v] \neq 0 \qquad \langle P, \texttt{pc} + 1, \Sigma \rangle \to \Sigma'}{\langle P, \texttt{pc}, \Sigma \rangle \to \Sigma'}$$

$$[\textsc{BnzSem}]\quad \frac{P[\texttt{pc}] = \texttt{bzr}(v, l) \qquad \Sigma[v] = 0 \qquad \langle P, l, \Sigma \rangle \to \Sigma'}{\langle P, \texttt{pc}, \Sigma \rangle \to \Sigma'}$$

(a) Write the semantics of $\texttt{mov}(v_1, v_2)$, which copies the contents of v_2 into v_1. Hint: see AddSem in our example rules.

(b) Write the semantics of $\texttt{ldm}(v_1, v_0)$, which is equivalent to the C command $v_1 = *v_0$. In other words, this instruction reads the contents of v_0, uses this value as an address in Σ, and copies the value stored at that address into v_1. Hint: see StmSem above.

(c) Write the semantic rule for the instruction $\texttt{jmp}(l)$, which changes the program counter `pc` to point to label l. Hint: take a look into BzrSem and BnzSem above.