CS411 Probs 1

A. Demers

1 Feb 2001 – due 8 Feb 2001

1. Let c be the **IMP** program

   $$X \leftarrow 1;\ \textbf{while}\ X \leq 1\ \textbf{do}\ X \leftarrow X + 1$$

   Using the large-step rules from Lecture Notes 1, show a derivation of

   $$\langle c, \sigma \rangle \ \rightarrow\ \sigma[2/X]$$

   You may use "obvious" facts about states, for example the fact that

   $$\sigma[m/X][n/X] \ = \ \sigma[n/X]$$

   Argue that the derivation does not depend on the initial state σ.

   Note we have recently added names to the large-step rules in the on-line lecture notes.

2. For the same **IMP** program c as in the previous question, but using the small-step rules from Lecture Notes 2, show all the derivations involved in the computation

   $$\langle c, \sigma \rangle \ \rightarrow_1\ \langle c_1, \sigma_1 \rangle \ \rightarrow_1\ \ldots\ \rightarrow_1 \langle , \sigma[2/X] \rangle$$

   Again, we have added names for the small-step rules in the on-line lecture notes.

3. (See Winskel Exercise 2.11) Suppose we want to extend **IMP** to allow expression evaluation to update the state and potentially fail to terminate. A simple way to do this is to add a construct

   $a\ ::=\ (\ c_0\ \textbf{resultis}\ a_0\ )$

   The intended meaning is: to evaluate such a command in state σ you should first execute command c_0 in state σ, resulting in a possibly-updated state σ'; you should then evaluate arithmetic expression a_0 in state σ'.

   Show how to extend the small-step rules for **IMP** to incorporate this new construct.

4. Now consider extending the large-step rules to handle the **resultis** construct. In this case you have a lot more to do. In particular, if arithmetic expression evaluation can affect the state, then you will need an arithmetic expresion evaluation relation of the form

   $$\langle a, \sigma \rangle \;\; \rightarrow \;\; \langle n, \sigma' \rangle$$

   Since evaluating a Boolean expression may require evaluating one or more arithmetic subexpressions (for example **if** $a \geq b$ **then** ...) you will need a Boolean expression relation with a similar form

   $$\langle b, \sigma \rangle \;\; \rightarrow \;\; \langle t, \sigma' \rangle$$

   and all the expression rules must be updated to carry around the updated states.

   Work out the details.

5. Up to now we have treated states as arbitrary functions on locations; that is

   $$\Sigma \;\; = \;\; \mathbf{Loc} \rightarrow \mathbf{N}$$

   Consider the set of *finite* states – informally, states in which at most a finite number of locations contain nonzero values.

   Define this concept formally – that is, give an inductive definition of a set $\mathbf{F} \subset \Sigma$ that corresponds to the finite states described informally above.

   Prove the following by induction on derivations:

   $$(\sigma \in \mathbf{F} \;\wedge \langle c, \sigma \rangle \;\; \rightarrow \;\; \sigma') \qquad \Longrightarrow \qquad \sigma' \in \mathbf{F}$$

   This is the (presumably) obvious fact that a (finite) terminating program execution can update only finitely many locations in the store.