CS 440: Programming Languages
Assignment: Big-step semantics

IMP Rules
The following are the big-step semantic rules of the simple imperative language (IMP) as described in class.

LITERAL  \[ \langle i, \sigma \rangle \Downarrow e_i \]  \text{when } i \in \mathbb{Z}  

VAR \[ \langle u, \sigma \rangle \Downarrow e_v \]  \text{if } u := v \in \sigma 

ARITH \[ \langle e_1, \sigma \rangle \Downarrow e_{v_1} \] \[ \langle e_2, \sigma \rangle \Downarrow e_{v_2} \] \[ \langle e_1 \oplus e_2, \sigma \rangle \Downarrow e_{v_1 \oplus v_2} \]  

Figure 1: Arithmetic expressions

LITERAL \[ \langle b, \sigma \rangle \Downarrow b \]  \text{if } b \in \{true, false\}  

VAR \[ \langle u, \sigma \rangle \Downarrow e_v \]  \text{if } u := v \in \sigma 

REL \[ \langle e_1, \sigma \rangle \Downarrow e_{v_1} \] \[ \langle e_2, \sigma \rangle \Downarrow e_{v_2} \] \[ \langle e_1 \sim e_2, \sigma \rangle \Downarrow b_{v_1 \sim v_2} \]  

Figure 2: Boolean expressions

SKIP \[ \langle \text{skip}, \sigma \rangle \Downarrow \sigma \]  

ASSIGN \[ \langle e, \sigma \rangle \Downarrow e_v \] \[ \langle x := e, \sigma \rangle \Downarrow \sigma[x := e] \]  

SEQ \[ \langle S_1, \sigma \rangle \Downarrow \sigma' \] \[ \langle S_2, \sigma \rangle \Downarrow \sigma'' \] \[ \langle S_1; S_2, \sigma \rangle \Downarrow \sigma'' \]  

IF-T \[ \langle b, \sigma \rangle \Downarrow b \text{ true} \] \[ \langle e, \sigma \rangle \Downarrow e_v \] \[ \langle S_1, \sigma \rangle \Downarrow \sigma' \] \[ \langle \text{if } b \text{ then } S_1 \text{ else } S_2, \sigma \rangle \Downarrow \sigma' \]  

IF-F \[ \langle b, \sigma \rangle \Downarrow b \text{ false} \] \[ \langle e, \sigma \rangle \Downarrow e_v \] \[ \langle S_1, \sigma \rangle \Downarrow \sigma' \] \[ \langle \text{if } b \text{ then } S_1 \text{ else } S_2, \sigma \rangle \Downarrow \sigma' \]  

WHILE-F \[ \langle b, \sigma \rangle \Downarrow b \text{ false} \] \[ \langle e, \sigma \rangle \Downarrow e_v \] \[ \langle S, \sigma \rangle \Downarrow \sigma' \] \[ \langle \text{while } b \text{ do } S, \sigma \rangle \Downarrow \sigma'' \]  

WHILE-T \[ \langle b, \sigma \rangle \Downarrow b \text{ true} \] \[ \langle e, \sigma \rangle \Downarrow e_v \] \[ \langle S, \sigma \rangle \Downarrow \sigma' \] \[ \langle \text{while } b \text{ do } S, \sigma \rangle \Downarrow \sigma'' \]  

Figure 3: Statements
Logistics and Submission

Please submit your solutions as a PDF (typed or neatly handwritten!) on Blackboard by the due date.

1 Extending the language

1. (5 points) We wish to add Boolean negation to IMP, via the \( \neg \) operator. Write down inference rules to describe the big-step semantics of this operator.

2. (10 points) We wish to add for loops to IMP, which will have the form “\( \text{for } v \text{ in } a_0 \text{ to } a_1 \text{ do } S \)”, which will run statement \( S \) with the variable \( v \) taking on values \( a_0, a_0 + 1, ..., a_1 \). E.g., “\( \text{for } x \text{ in } 1 \text{ to } 5 \text{ do } S \)” will run \( S \) with \( x \) taking on values 1, 2, 3, 4, 5, in that order.

Write down inference rules to describe the big-step semantics of the for statement. Note that the loop variable is allowed to clash with pre-existing variables, and it may remain in the environment after the loop completes.

2 Proofs

For each of the following assertions, draw a proof tree:

3. (5 points) Draw a proof tree for the following assertion:
   \( < t := a + b; a := b; b := t, \{a := 5, b := 10\} > \{t := 15, a := 10, b := 15\} \)

4. (5 points) Draw a proof tree for the following assertion:
   \( < \text{if } x > y \text{ then } m := x \times 10 \text{ else } m := y \times 10, \{x := 10, y := 20\} > \{x := 10, y := 20, m := 200\} \)

5. Consider the program \( P = \text{“for } x \text{ in } m \text{ to } n \text{ do } \text{sum} := \text{sum} + x” \) (which makes use of the for statement defined in the previous section). Note that \( m \) and \( n \) are not variables, but represent arbitrary integer constants.
   (a) (5 points) Describe the environment \( \sigma' \) such that \( < P, \sigma_0 > \{\} < \sigma' \), where \( \sigma_0 \) is an environment which maps all variables to 0. Hint: you may make use of the summation operator (\( \Sigma \)).
   (b) (5 points) Write down a program \( Q \) using the while statement, which is semantically equivalent to \( P \). 
   (c) (10 points) Prove that \( P \) and \( Q \) are equivalent when run in starting state \( \sigma_0 \).