Homework 10

Please hand your solution in during class or email them to the instructor with CC to ly603@nyu.edu.

The deadline for Homework 10 is April 22.

Problem 1 Operational Semantics of IMP (7 Points)

(a) Use the operational semantics of IMP commands to infer the successor state of the state $\{x \mapsto 2, y \mapsto 3\}$ and the following command

$$\texttt{if } x > 0 \texttt{ then } y := x * y;\ x := x - 1 \texttt{ else skip}$$

(3 Points)

(b) We say that two commands c_1 and c_2 are *operationally equivalent* if

$$\forall q, q' \in Q.\ q \xrightarrow{c_1} q' \iff q \xrightarrow{c_2} q'$$

Are the following pairs of IMP commands operationally equivalent? Give a proof (using the operational semantics) or a counterexample.

(i) $\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2$ and $\texttt{if } \neg b \texttt{ then } c_2 \texttt{ else } c_1$,

(ii) $\texttt{if } b \texttt{ then } c \texttt{ else } c$ and c.

What changes, if you consider the corresponding Java statements?

(i) $\texttt{if}(e)\, c_1 \texttt{ else } c_2$ and $\texttt{if}(!e)\, c_2 \texttt{ else } c_1$,

(ii) $\texttt{if}(e)\, c \texttt{ else } c$ and c.

(4 Points)

Problem 2 For-Loops in IMP (8 Points)

We extend the IMP language with for-loops

$$\texttt{for } x \texttt{ to } e \texttt{ do } c$$

where x is a location, e is an arithmetic expression, and c is a command. We specify the semantics of such loops by saying that for every x, e, and c, the above command should produce the same result as:

$$\texttt{if } x \leq e \texttt{ then } c;\ x := x + 1;\ \texttt{for } x \texttt{ to } e \texttt{ do } c \texttt{ else skip}$$

(a) Provide an IMP command using for-loops (but no while-loops) that diverges for every possible starting state. **(3 Points)**

(b) Extend the operational semantics of IMP with appropriate rules for for-loops. **(5 Points)**

Problem 3 Loops with breaks (10 Points)

Java provides the `break` statement that when executed within a loop causes the execution of the loop to be stopped immediately. Execution is then continued with the first statement after the corresponding loop. We can model `break` statements by extending the flow component of program states

$$\mathit{Flow} ::= \mathit{Norm} \mid \mathit{Ret} \mid \mathit{Exc}\langle \mathit{Address} \rangle \mid \mathit{Break}$$

Use this extension to define the operational semantics of `break` statements and `while` loops with breaks.