11 Optimising Compilers (AM)

(a) Give a semantic notion of a variable being live at a program point, explaining why this is problematic to calculate. Now give a simpler-to-calculate notion of liveness and explain how it relates to the semantic notion. Formulate dataflow equations whose solution(s) give the liveness at each program point. You need only consider liveness of simple non-address-taken variables. [4 marks]

(b) Suppose we have a basic block of \( p \) simple statements. Give a formula relating the liveness on entry to the block to those of its \( q \) neighbouring blocks in the control flow graph. This formula naturally uses \( O(p) + O(q) \) operations – justify this statement. It is claimed that this formula can be re-arranged to require only \( O(q) \) time to calculate by only using one ‘\( \cup \)’ and one ‘\( \setminus \)’ operator. Determine whether this is true. [Hint: you may wish to consider examples, and to start by solving the case \( p = 2 \). Partial credit will be given for a good set of concrete examples arguing for or against.] [5 marks]

(c) To solve the dataflow equations, an initial approximation to liveness at the start of each basic block is required. What is it, and indicate why this leads to a preferable solution. [2 marks]

(d) Solving dataflow equations is usually expressed iteratively, where each iteration is of the form “for every basic block re-calculate the set of live variables from the current sets of live variables of its neighbours”. We want to determine whether some basic-block orderings in “for every basic block” result in fewer overall iterations than others. Suppose the program has \( k \) basic blocks, but no cycle in the control flow graph; give an optimal ordering which only requires one dataflow iteration to calculate liveness (a second would only calculate the same value of the first). Also give such a program and an ordering which maximises the number of iterations required, giving the number of iterations in terms of \( k \). [5 marks]

(e) Consider the program with four labelled blocks (with \( B1 \) as entry node):

\[
\begin{align*}
B1: & \quad x = \text{read}(); \quad y = \text{read}(); \quad z = \text{read}(); \quad \text{goto B2}; \\
B2: & \quad z = z+1; \quad x = x-1; \quad \text{if} \ (x>0) \ \text{goto B3}; \quad \text{else goto B4}; \\
B3: & \quad z = z+1; \quad y = y-1; \quad \text{if} \ (y>0) \ \text{goto B2}; \quad \text{else goto B4}; \\
B4: & \quad \text{print}(z);
\end{align*}
\]

Show (i) there is no basic block ordering for which a single iteration gives the correct liveness at each label, but (ii) there is an ordering for which two iterations suffice (in the sense that a third would agree with the second). Give your ordering both explicitly as a permutation of \( \{B1,B2,B3,B4\} \) and also as a general principle along the lines of your answer to part (d). [4 marks]