Optimising Compilers

Consider a flowgraph, containing 3-address instructions, which represents a source-level routine. Let \( e \) be an expression (\( e \) may be considered to be a right-hand side 3-address instruction, i.e. either \( x \) or \( x \mathbf{op} y \) where \( x \) and \( y \) are variables).

We say that \( e \) is very busy at a node \( n \) if all paths from \( n \) compute the expression \( e \) at least once and each such computation yields the same value as evaluating \( e \) at \( n \) would (i.e. no modification of its variables occurs between \( n \) and the first occurrence of \( e \) on any path from \( n \)).

Let \( VB(n) \) be the set of very busy expressions at \( n \).

(a) Give data flow equations for \( VB(n) \). [4 marks]

(b) Give the relationship, if any, to the set \( \text{Avail}(n) \) of expressions available at \( n \) including the direction (forwards/backwards) of the analyses. Indicate whether either inclusion \( VB(n) \subseteq \text{Avail}(n) \) or \( \text{Avail}(n) \subseteq VB(n) \) holds. [4 marks]

(c) Sketch an algorithm to compute \( VB(n) \), briefly commenting on any initialisation. [4 marks]

Suppose now that we compile a program in a call-by-need functional language into 3-address code using closures (i.e. \( \lambda() . e' \)) to represent laziness. Given a functional definition \( f(x,y,z) = e \) we have notions of \( f \) being strict in, or needing, its second parameter \( y \).

Point out similarities and differences between these notions and that of \( y \) (or \( y() \)) being very busy at some, to be determined, point in the 3-address code form of \( e \). [8 marks]

Hint: you may find it helpful to consider separately

(a) a case where \( e \) uses only the conditional function and strict primitive functions such as +

(b) a case such as \( f(x,y) = g(x, y + 1) \)