2008 Paper 8 Question 8

Optimising Compilers

(*a*) Sometimes evaluating expressions may be partially or wholly redundant in that they have been previously evaluated on some or all of the program paths leading to them.

(*i*) Outline the theory of available expressions, including dataflow equations and how to compute their solution. Also give a brief explanation of how to use this solution to remove common-subexpressions. How does the idea of either form of redundant computation relate to the notion of common-subexpression? [7 marks]

(*ii*) Give an example of a redundant computation that is not removed by the technique you give in part (*i*). [2 marks]

(*b*) Consider an intra-procedural dataflow analysis for security. Variables may hold high-security (e.g. a PIN) or low-security (e.g. a counter) values. Program constants are low-security, and on function entry only variables in the set H are high-security. Security flows through direct dataflow: the result of an assignment is assumed to be high-security if a variable on the right-hand side may hold a high-security value.

(*i*) Design a dataflow analysis that calculates, for each node n in a flowgraph, the set of variables that *may* hold a high-security value at n. Have you defined a forward analysis or backward analysis? How is your dataflow analysis implemented, noting particularly initialisation of any iteration? [7 marks]

(*ii*) Give an informal argument as to why your dataflow analysis is safe or an example of why it is not—in either case discussing reasons or interesting cases. For this purpose treat an analysis as being safe if it is impossible to write a function body that (1) implements the identity function and (2) has the property that the output variable is analysed as low-security on exit even though the input variable is high-security (a member of H) on entry. [4 marks]