11 Optimising Compilers (AM)

(a) Explain the ideas behind available expression analysis. Your explanation should include data-flow equations, an informal argument as to why these correctly capture a semantic notion of availability, any issues with decidability and an algorithm to solve the data-flow equations. It is sufficient to consider, as candidate available expressions, those expressions of the form \( v \oplus v' \) where \( \oplus \) is a binary operation and \( v \) and \( v' \) are variables or constants. [6 marks]

(b) Show how the result of available expression analysis can be used to perform common sub-expression elimination. You need not give an algorithm, but explain the steps in the optimisation carefully. [4 marks]

(c) Assume that your intermediate code is represented by three-address instructions stored within basic blocks, and with a fresh temporary used whenever a temporary variable is used to hold intermediate results of a larger expression. Explain how your algorithm deals with optimising the program fragment

\[
\begin{align*}
u &= f(a+b*c, a+b*c); \\
v[a+b*c] &= u;
\end{align*}
\]

where \( a \) is a global variable which may be updated by \( f \), and \( b \) and \( c \) are local variables. [5 marks]

(d) Explain carefully how your common sub-expression elimination algorithm reacts to program fragment:

\[
\text{do } \{ x += b*c; \ldots \} \text{ while } (\ldots);
\]

and also to program fragment:

\[
\text{z = b*c; do } \{ x += b*c; \ldots \} \text{ while } (\ldots);
\]

commenting on any differences and on any similarity to lifting a loop-invariant expression out from a loop. In both cases assume neither \( b \) nor \( c \) is modified anywhere in the loop. [5 marks]