Optimising Compilers

(a) Summarise the basic principles behind strictness analysis including: what language paradigm it can be applied to, the representation of compile-time values expressing strictness, how these may be calculated and how the results of such calculations can be used to optimise programs. [8 marks]

(b) A program contains the following user function definitions. Give corresponding strictness functions assuming that if-then-else takes an integer as its first argument.

(i) fun f(x) = 42 [1 mark]
(ii) fun g(x) = g(x+1) [1 mark]
(iii) fun h(y,z) = if f(7) then y else z [2 marks]
(iv) fun k(x,y,z) = pif(x,y,z) where pif(e,e′,e′′) is a primitive which evaluates its three arguments in parallel, returning e′ if e evaluates to a non-zero integer, returning e′′ if e evaluates to zero and also returning e′ if e′ and e′′ evaluate to the same integer even if e is still being evaluated. [4 marks]

(c) “Any Boolean expression be containing variables \{x_1, \ldots, x_k\} but not containing negation can be expressed as the strictness function for a user-defined function fun u(x_1, \ldots, x_k) = e.” Argue that this statement is true, showing how to construct some such e from a given be. [4 marks]

[Hint: you may assume be has been written in DNF form

\((v_{11} \land \cdots \land v_{1m_1}) \lor \cdots \lor (v_{n1} \land \cdots \land v_{nm_n})\)

where \(v_{ij}\) are members of \(\{x_1, \ldots, x_k\}\).]