Optimising Compilers

(a) Summarise briefly the principles of strictness analysis, including descriptions of:

(i) the space of values used for analysis-time representation of a $k$-argument, 1-result function in the source language;

(ii) how a built-in function is given an abstract meaning;

(iii) how a recursive user-defined function is given an abstract meaning (it is acceptable to do this part by example);

(iv) the machine-level benefit of the associated optimisation. [8 marks]

(b) A problem amenable to similar treatment is that of escape analysis. Here we have a call-by-value language with cons and the question to be answered is “whether a value containing a cons-node passed as argument to a function may be returned (‘escape’) as part of the function’s result”.

(i) Choose (and state clearly) an appropriate set of abstract values and abstractions of functions to formalise the problem of escape analysis for a simple first-order language with integers and simple integer lists (but not lists of lists). Also give abstract interpretations of if-then-else, +, cons, hd and tl.

[Hint: to manage this system without using static types, you might best assume that nil is treated as 0, and that any type-error (dynamically detected) such as cons(1,nil)+3, tl(3) and even (because of the ‘no-lists-of-lists’ rule) cons(cons(1,nil),nil) gives result 0.] [8 marks]

(ii) Give without proof abstract meanings (resulting from your system) of the following functions:

\[
\begin{align*}
  f(x,y,z) &= \text{cons}(\text{hd}(\text{tl}(x)), \text{if } \text{hd}(x) \text{ then } y \text{ else } \text{tl}(z)) \\
  g(x,y) &= \text{if } x=0 \text{ then } 0 \text{ else } \text{cons}(\text{hd } x, \text{g(tl } x, y)) \\
  h(x,y) &= \text{if } x=0 \text{ then } x \text{ else } \text{cons}(\text{hd } x, \text{h(tl } x, y)) \\
  k(x,y) &= \text{if } x=0 \text{ then } y \text{ else } \text{cons}(\text{hd } x, \text{k(tl } x, y))
\end{align*}
\] [4 marks]