2007 Paper 9 Question 16

Optimising Compilers

(*a*) Summarise very briefly (one short paragraph each) the overall idea behind program analysis using

(*i*) abstract interpretation;

(*ii*) set-constraint-based (CFA-like) analysis;

(*iii*) rule-based analysis.

[6 marks]

(*b*) Consider the following language of integer expressions e and (integer) list expressions E where n represents integer constants, x and X respectively range over integer and list variables, $\oplus$ represents integer operations (e.g. $+$, $\leq$ etc.), and **if** and **IF** test their first argument for zero/non-zero as in C:

$$
\begin{aligned}
e &::= x \mid n \mid e_1 \oplus e_2 \mid \mathbf{hd}\ E \mid \mathbf{if}(e_0, e_1, e_2) \\
E &::= X \mid [\,] \mid e :: E \mid \mathbf{tl}\ E \mid \mathbf{IF}(e, E_1, E_2)
\end{aligned}
$$

In *escape analysis and optimisation*, given a call to f such as

$$g(x, y) = f(x :: x :: [\,], y :: [\,])$$

we want to know whether or not the result of f can include any of the cons-cells reachable from its arguments. A formal parameter of f that might be incorporated into its result is known as *escaping*. This is useful because if (say) formal parameter 1 to f cannot escape then cons-cells allocated for actual parameter 1 can be allocated (more cheaply) on the stack instead of in the heap.

This problem may be formulated as an analysis that takes an expression, e or E, constituting the body of f. The parameters of f are the free variables, x_i and X_i, of its body.

Express this analysis using *two* of the techniques from part (*a*). In both cases state how to use the analysis result for e or E to test "parameter X_i definitely does not escape from E or e". [Hint: in some analyses it is easier to treat the variables x_i and X_i just as strings, and in others as variables ranging over $\{0, 1\}$.]

[5 marks each]

(*c*) Indicate what changes would be necessary for *one* of your analyses were the syntax also to allow a recursive call to f.

[4 marks]