p7q4
AM

Optimising Compilation `optcomp7.tex` 2001

An expression is *very busy* at a program point n if, no matter what path is taken from n, the expression is always used before any of the variables occurring in it are redefined. A transformation using *Very Busy Expression* (VBE) analysis is to evaluate the expression at the block and store its value for later use.

Give dataflow equations for, and a pseudo-code algorithm to calculate, VBE for a program in flowgraph form. State whether your dataflow equations are *forwards* or *backwards*. Sketch the above transformation which exploits VBE in more detail. [11 marks]

Show how to calculate the *call graph* of a program, and explain the safety property your call graph should have. (I.e. relate the call-graph you define to possible execution behaviour.) Detail how you handle procedure-valued variables, and state whether it is possible to improve on the technique you have chosen for such variables. [6 marks]

Argue how feasible it is to calculate the call-graph for a Java program, considering carefully the case of inheritance and use of the `final` keyword. [3 marks]

Solution Notes

See [Hankin, Nielson and Neilson] p44, important is the use of intersection and the fact that this is a backwards analysis.

Nodes are procs, arcs are calls. Safety is every call in every possible execution must correspond to an arc in the call-graph. Simple treatment of proc. vars is unsafe. One idea is to take the view that every call to a variable fn may call any possible address-taken function. CFA can be used to improve on this.

Yes, it's possible, but most calls are indirect and thus hard to predict because in inheritance. 'final' can be used to show that a class is not overridden and thus we more often know which method to call. Can also use type info (can only call a method with the same type) even though indirect calls.