Re: algorithm for computing the interference graph in SSA form?

From comp.compilers

From: mwolfe@pgroup.com (Michael Wolfe)
Newsgroups: comp.compilers
Date: 8 Jan 1998 00:53:04 -0500
Organization: The Portland Group, Inc. Wilsonville, Oregon U.S.A.
References: 97-12-141 98-01-026
Keywords: optimize, analysis

Paul Haahr wrote:
>
> *It's probably just that I'm not clever enough, but all I've been able*
> *to come up with is something that looks like an iterative data flow*
> *analysis, which can be somewhat improved by paying attention to loops*

Cliff Click wrote:
> *(1) Solve the classic LIVE problem any way you like. Phis need special*
> *handling.*

Sorry, Cliff, this is "something that looks like an iterative data flow analysis", just what Paul came up with himself.

SSA neither helps nor hurts live variable analysis, and Cliff's comments can be used to implement classical live variable analysis using SSA. SSA also neither helps nor hurts interference graph construction, though you may have a different number of variables (live ranges) in your interference graph.

My interpretation of SSA is that it is an efficient and effective solution to reaching definitions [others have more aggressive interpretations].

Years ago, Eric Stoltz, while a student of mine at the Oregon Graduate Institute, worked out the mechanisms for using an SSA-like structure to solve live variables, and in particular, to build the interference graph without actually solving the live-variables problem. The solution was independent of SSA, that is, the program need not be in SSA form. The method inserted lambda-terms at the end of a basic block where the block had 2 or more successors and where there were different reached uses along the paths. Compare this to SSA, where phi-terms are inserted at the beginning of a basic block where the block has 2 or more predecessors and where there were different reaching definitions along the paths, and you see the relation. Lambdas for a variable were inserted at the post-dominance frontier of the blocks containing uses of that variable.

Eric also looked at using the same structure for other data-flow analysis problems, in his PhD thesis, but we never published the results for two reasons. First, unlike SSA, we didn't find any problems that got a better solution from this method; SSA not only solves reaching definitions, it is more effective, that is, you have more information than reaching definitions gives you (you get the paths along which the definitions reach). Second, again unlike SSA, we didn't get any performance advantage. We tried really hard to optimize interference graph construction using the lambda-terms, seeing that we didn't need to solve a global iterative live-variable problem, and in fact didn't need to solve the live-variable problem globally at all. However, a simple bit-vector live-variable analysis and basic-block traversal interference graph constructor still outperformed our best efforts (by a constant factor, between 1.2 and 1.9). Our best guess is that the cost of inserting and traversing the lambda terms ate up any gains from a more efficient algorithm, and remember that bit-vector algorithms have a factor of 32 (or 64, word size) advantage.

For reference, see Eric's thesis:
ftp://cse.ogi.edu/pub/tech-reports/1995/95-TH-001.ps.gz

Bottom line, I'd go with iterative live-variable analysis (not that my opinion matters).

-Michael Wolfe
--

Return to the comp.compilers page.
Search the comp.compilers archives again.