CS432N/CS728: Compiler Design
I semester 2004-05

Assignment: Byte Code Optimization

---

1 The problem

Given that you have a bytecode interpreter, the next step is to introduce an optimizations as options to your interpreter. Implement the following optimizations for the bytecode language of the last assignment.

1. Constant propagation
2. Copy Propagation
3. Local and Global Common Subexpression Elimination
4. Loop Invariant Code Motion
5. Induction variables
6. Dead Code Elimination

Study Chapter 21 of Muchnick's book to analyse how to organize the calls to various optimizations and decide on

- the order of performing optimizations,
- a criterion to ensure that you stop the iteration process at some suitable point. The criterion should not be anything fixed. Clearly the number of iterations of the sequence of optimizations depends upon the input code. It could be a function of either the number of optimizations detected at the end of each iteration or the number of changes in code produced with each iteration.

2 The structure of the solution

In order to do that structure your code in the following **modules**.

Driver This module consists of the main loop for performing the optimizations. It also (after doing an elementary scan and parse of the input program along with a symbol table– you may copy it from your interpreter) constructs an encoding of the control-flow graph of the input program, and makes it available for the other modules. The other modules such as data-flow analysis and the actual optimization modules are nested in this module.

Data flow analysis This should be a separate signature and structure so that all the data-flow problems of the input program can be performed at one central place. The major data structures here are the functions which manipulate the bit-vectors you use the collect and collate information.

Optimizations Each optimization is a function which takes the control flow graph as input (and also has access to the input code) and actually performs the optimization. Each such function therefore outputs a fresh control flow graph and corresponding code. The whole suite of optimizations are the methods available to the Driver module.

The external interface has to be such that:

- a user specifies which optimizations are to be performed (including none) either interactively or as command-line options.
- The input to the optimizer is a program written in the bytecode language of the last assignment
- The output is again optimized bytecode written in the same language, so that the optimized code can simply be '**UNIX-piped**' to the interpreter you have already written.