Assignment #3: Dead Code Elimination

Due: Before class, March 3rd

Introduction

In this assignment, you will write a dead code elimination pass. This pass will use the reaching definitions pass from the last assignment plus elements of the BVD, CFG, and CFA MachSUIF libraries.

Before starting the assignment you should:

- Read Section 18.10 of Muchnick, Advanced Compiler Design and Implementation
- copy the files in `/afs/cs.cmu.edu/academic/class/15745-s03/public/assignments/2` to your working directory. E.g.

  ```
  cp -a /afs/cs.cmu.edu/academic/class/15745-s03/public/assignments/3
  ~/localnci/assignments/
  ```

  This will create one sub-directory, `dce`. In `dce` You will find the driver code as well as a skeleton for the dce pass. You should only have to modify `dce.h` and `dce.cpp`.

Requirements

In a nutshell, we expect that, after running your dead code elimination pass, only the minimum set of necessary instructions will be present in the compiled program. To define this requirement more precisely let's consider what defines a necessary instruction in the context of MachSUIF.

Machine SUIF passes operate at the level of procedures and should be able to handle code containing hard and virtual registers from any supported architecture. As MachSUIF operates at the level of procedures, any code that generates results that could be observed from outside of a procedure should be considered necessary; as there is currently, in MachSUIF, no way to assert it is not.

Also, some registers are used to implement the calling convention/frame layout of the ABI used in a given architecture. Such registers are named the non allocatable registers in MachSUIF lingo, because they serve a special purpose. Instructions that define such registers are always necessary instructions because otherwise the code would not conform to the ABI associated with the underlining architecture. For details on obtaining such information in MachSUIF, check `machine/reg_info.h`.

MachSUIF also support 'built-in' instructions, those instructions should be considered black boxes and, therefore, are always required. Similarly, to make sure your MachSUIF pass cooperates with other passes, instructions that contain annotations should also be considered necessary, otherwise annotations that are important to other passes could be lost. Also, null instructions, labels, and 'dot' instructions are important even though they do not compute any result. To identify instructions in the above classes check `machine/instr.h` and `machine/context.h`.

All other instructions could potentially be unnecessary. Of course instructions that necessary instructions require to operate properly are also necessary. That is, in compiler lingo, if a necessary instruction is control or data dependent on a second instruction, the second instruction must be considered necessary. To compute control dependences check the MachSUIF Control Flow Analysis library, located at `cfa/*` , data dependence information should be obtained through reaching definitions; which was the main topic of assignment #2[1].

[1] You may use the class binary for reaching definitions instead of your own solution to assignment #2

Finally, observe that some control flow related instructions may be found unnecessary. Those cannot be deleted directly otherwise you may break the control flow graph based representation. There is no need to write control flow graph manipulation/optimization routines as those already exist in `cfg/*` . You should be able to remove the unnecessary control flow instructions indirectly by manipulating the control flow graph using such routines.

As in previous assignments, the binary solution (`do_dce`) implements all the requirements and can be used for testing/comparison purposes.

Hand In

Please create a text file `group.txt` containing your user id and your partner's. To hand in your solution just copy the files `dce.{h,cpp}` and `group.txt` to the appropriate hand in directory. Also, only one partner should hand in a solution.

As before, you can hand in as many times as you wish. Only the last submitted version will be graded.

Good luck!

Acknowledgments

Thanks to Mike Smith and Glenn Holloway who originally created this assignment; and Seth Goldstein who created the original 15-745 version.