Homework 8 – Due Friday, November 11, 2010

Please refer to the general information handout for the full homework policy and options.

Reminders

• Your solutions are due before the lecture. Late homework will not be accepted.

• Collaboration is permitted, but you must write the solutions by yourself without assistance, and be ready to explain them orally to a member of the course staff if asked. You must also identify your collaborators. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

• To facilitate grading, please write down your solution to each problem on a separate sheet of paper. Make sure to include all identifying information and your collaborators on each sheet.

• For problems that require you to provide an algorithm, you must give a precise description of the algorithm, together with a proof of correctness and an analysis of its running time. You may use algorithms from class as subroutines. You may also use any facts that we proved in class or from the book.

Problems to be handed in

1. A block of code in a program is dead if the block is never executed on any input (that is, there is no possible input on which the line of code would be executed). Large programs often contain many blocks of dead code, and it would be convenient if compilers could reliably identify “dead” code.

Let $\text{dead-code} = \{(p, i) : p$ is a C program in ASCII text format and $i$ is the number of a dead line of code in $p\}$. We wish to prove that $\text{dead-code}$ is undecidable.

One way to do this would be to come up with a diagonalization argument as we did in class for the halting problem. However, there is an easier way: we can use reductions!

(a) Which of the following implies that $\text{dead-code}$ is undecidable? Justify your answer. No points will be given without an explanation.

• a reduction from the halting problem to the dead code problem (that is, an algorithm that solves the halting problem assuming that it has access to a subroutine which solves the dead-code problem).

• a reduction from the dead code problem to the halting problem (that is, an algorithm that solves the dead code problem assuming that it has access to a subroutine which solves the halting problem).

(b) Prove that the dead code problem is undecidable by giving an appropriate reduction. [The running time of your reduction doesn’t matter, but the one I have in mind is linear time.] As usual, you have to prove that your reduction is correct (assuming it has access to an appropriate subroutine).