CPEG 421/621 - Homework 1
Issue Date: Tuesday, 21 February, 2012
Due Date: Tuesday, 6 March, 2012

Instructions

Please write the answer to each problem on one sheet of paper. Be as concise and clear as you can. Although we accept hand written answers, we encourage you to hand in print-out solutions or email your solutions to `aron+ta@udel.edu` and `szuckerm@eecis.udel.edu`. If you do hand in hand-written solutions, please make sure the writing is clearly readable. To avoid misplacement of various components of your assignment, make sure that all answer sheets are stapled together. You may discuss the problems with your classmates, but all solutions must be derived independently.

Problem 1

Program flow analysis is a process of estimating properties of program control flow at each program point. The information provided by a program analysis is useful in general compiler optimization and code generation, as well as in software/hardware co-design. To understand the concept of program flow analysis consider the program written in a C like language shown in Figure 1.

a. Generate a three-address intermediate code for the program.

b. Partition your three-address intermediate code into basic blocks.

c. Draw the CFG for the program in Figure 1.

d. Define the term back-edge.

e. List the back-edges in the CFG, and their corresponding natural loops.

f. Show why the graph representing the domination relation of a CFG must form a tree.

g. Draw the dominator tree of the CFG from 1-c.

```
1  a[] = ...;
2  s = 0;
3  for(v=0 ; v<n ; v++)
4  {
5        m = 0;
6        for(i=v ; i<n ; i++)
7        {
8             m = m+a[i]*5;
9             if(m != 0)
10            s = s+(a[i]+d*5);
11            else
12            s = s-(a[i]+d*5);
13       }
14 }
15 return (s);
```

Figure 1: A simple program to illustrate flow analysis.

Problem 2

Consider the program in Figure 2:

a. Compute the live-in variables and the live-out variables for each basic block.

b. Perform constant propagation on the code.

c. Using DAG representation, perform redundant subexpression elimination.

```
1  a[] = ...;
2  b[] = ...;
3  n = 4207849484;
4  for (i=0 ; i<n ; i++)
5  {
6       v = 2*a[i]*b[i];
7       if (a[i]*b[i] < n)
8       {
9            v = v + 2*a[i] + b[i]*2*a[i];
10      }
11      else
12      {
13           v = v - 2*a[i];
14      }
15      a[i] = v;
16 }
17 return &a[0];
```

Figure 2: A simple program to illustrate liveness, common subexpression elimination, and constant propagation.

Extra Credit: What is the relationship between algebraic equations and set theoretic ones, w.r.t. dataflow analysis? Give some intuition - not necessarily a formal explanation.