CS 6340 – Spring 2013 – Assignment 3 **Name** Sam Britt, Shriram Swaminathan,
Assigned: January 16, 2013
Due: January 23, 2013 **Name** and Sivaramachanran Ganesan

At the beginning of class on the due date, submit your neatly presented solution with this page stapled to the front (70 points).

NOTE: All work on this problem set is to be done with your partner and without solutions from other past or current students. Any violations will be dealt with according to the Georgia Tech Academic Honor Code and according to the College of Computing process for resolving academic honor code violations. All work must be done using some document creation tool. In addition, graphs must be drawn with a graph-drawing tool—no hand-drawn graphs will be accepted. We'll discuss this requirement more in class.

1. Given the following program and the statement-based control-flow graph for that program (which you created in Problem Set 1):

```
  procedure sqrt(real x):real
     real x1,x2,x3,eps,errval;

     begin
1.     x3 = 1
2.     errval = 0.0
3.     eps = .001
4.     if (x <= 0.0)
5.        output("illegal operand");
6.        return errval;
7.     else
8.        if (x < 1)
9.           x1 = x;
10.          x2 = 1;
11.       else
12.          x1 = eps;
13.          x2 = x;
14.       endif
15.       while ( (x2-x1) >= 2.0*eps )
16.          x3 = (x1+x2)/2.0
17.          if ( (x3*x3-x)*(x1*x1-x) < 0)
18.             x2 = x3;
19.          else
20.             x1 = x3;
21.          endif;
22.       endwhile;
```

Sam Britt
Shriram Swaminathan
Sivaramachanran Ganesan

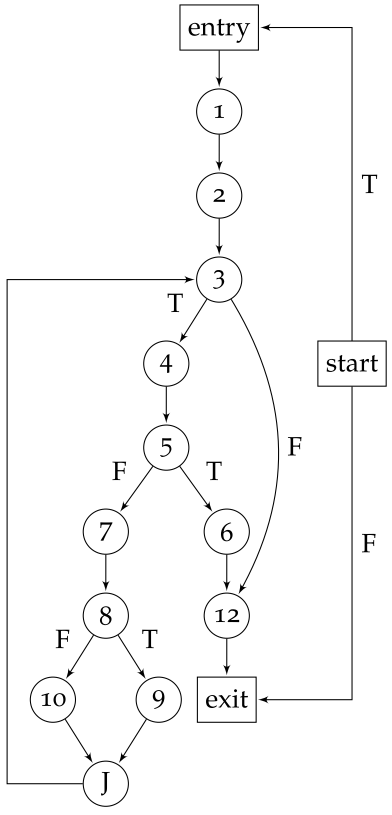
CS 6340
Assignment 3
Jan. 26, 2013

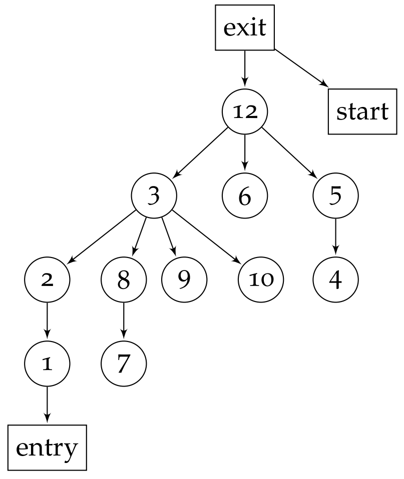
1. (a) Reaching definitions are shown in Table 1. Although the data is presented in order of increasing block number, the table was actually created in the following depth-first order: 1, 2, 3, 4, 8, 12, 13, 9, 10, 15, 23, 16, 17, 20, 18, 5, 6.

   (b) Reachable uses are shown in Table 2. Although the data is presented in order of increasing block number, the table was actually created in the following reversed depth-first order: 6, 5, 18, 20, 17, 16, 23, 15, 10, 9, 13, 12, 8, 4, 3, 2, 1.

   (c) Definition-use pairs are shown in Table 3, grouped according to use.

2. (a) First, we create the augmented CFG and the postdominator tree:





Augmented CFG. Corresponding postdominator tree.

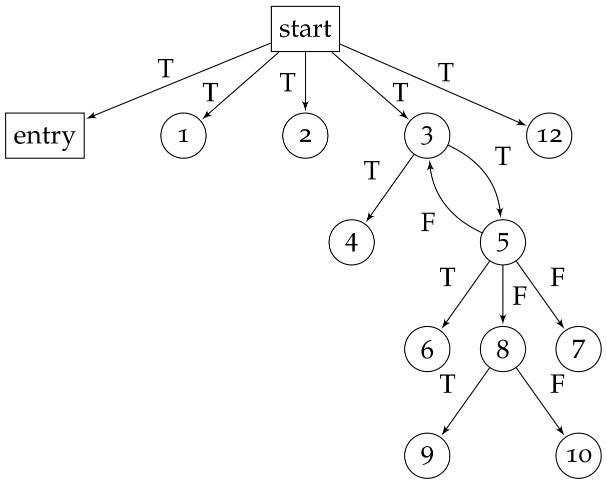
We then find the set S, where each element in S is an edge $(A, B) \in G$, where G is the above CFG and B does not postdominate A. We find that

$$S = \{(\text{start, entry}), (3, 4), (5, 6), (5, 7), (8, 9), (8, 10)\}.$$

For each edge $(A, B) \in S$, we find L to be the common ancestor of A and B. Finally, the nodes that are control-dependent on A are those the path from L to B on the postdominator tree, including B, and including L only if $L = A$. These results are summarized in the following table:

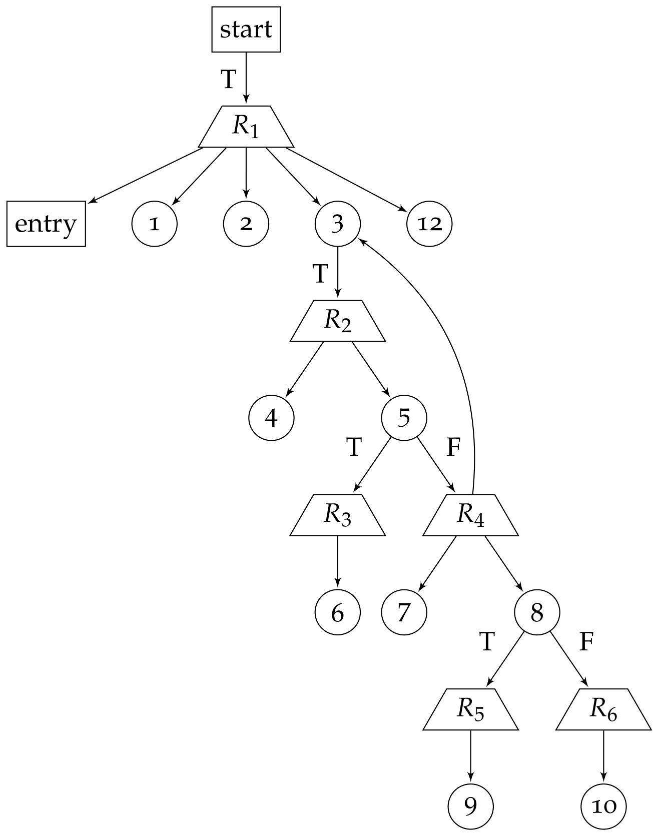
$(A, B) \in S$	Condition (T/F)	L	Nodes dependent on A
(start, entry)	T	exit	{entry, 1, 2, 3, 12}
(3, 4)	T	12	{4, 5}
(5, 6)	T	12	{6}
(5, 7)	F	12	{3, 7, 8}
(8, 9)	T	3	{9}
(8, 10)	F	3	{10}

The control dependence graph is constructed directly from the above table.



Control dependence graph, without regions.

(b) Below, regions (indicated as trapezoids) are added to the control dependence graph. This was performed in one pass through the control dependence graph: for each node n in the graph, one region was created to group the nodes dependent on n evaluating to `true`, if any, and another region to group the nodes dependent on n evaluating to `false`, if any. Each region was made the parent of its corresponding node group, and the region in turn was made a child of n.



Control dependence graph, with regions added.

3. (a) Our algorithm assumes that each block contains a single statement, so that we can be sure each block contains a single definition. Let S be an array that contains, for each block (statement) in the old program, the tuple (ℓ, d_ℓ, k_ℓ), where

   ℓ is the statement number (and block number),

   d_ℓ is the variable that statement ℓ defines (possibly none, represented by the special symbol $\perp$), and

   k_ℓ is the `KILL` set for the block at statement ℓ (possibly empty).

   For our algorithm to work, we need to store the latest value of S between iterations of the program.

   (b) We assume data about the changes to the program are given as a list C containing (ℓ, d'_ℓ) tuples, where ℓ is the line number that changed, and d'_ℓ is the variable that ℓ defines in the new program. The value of d'_ℓ can be $\perp$ if there is no definition at ℓ, and it can also be the same as d_ℓ, the variable defined by ℓ in the old program, if the line was altered but still defines the same variable. The following algorithm will update the relevant `KILL` sets in S.

```
UPDATE-KILL $(S = [(\ell_i, d_{\ell_i}, k_{\ell_i}), \ldots], C = [(\ell_j, d'_{\ell_j}), \ldots])$
1   for each $(\ell_i, d'_{\ell_i})$ in $C$
2       Find $(\ell_i, d_{\ell_i} k_{\ell_i})$ in $S$
3       if $d_{\ell_i} \neq d'_{\ell_i}$
4           for each $(\ell_j, d_{\ell_j}, k_{\ell_j})$ in $S$
5               if $d_{\ell_j} = d'_{\ell_i} \neq \perp, \ell_j \neq \ell_i$
6                   Let $k_{\ell_j} \leftarrow k_{\ell_j} \bigcup \ell_i$
7                   Let $k_{\ell_i} \leftarrow k_{\ell_i} \bigcup \ell_j$
8               if $\ell_i \in k_{\ell_j}$
9                   Let $k_{\ell_j} \leftarrow k_{\ell_j} - \ell_i$
10                  Let $k_{\ell_i} \leftarrow k_{\ell_i} - \ell_j$
11          Let $d_{\ell_i} \leftarrow d'_{\ell_i}$
```

This procedure iterates through every line of change ℓ_i in C. In line 3, it first checks to see if the definition at the line has really been changed, as these changes are the only ones that will affect reaching definitions. It then iterates through S. If a different block defines the same variable as ℓ_i (and they actually give a definition; that is, neither are $\perp$), then both `KILL` sets need to be updated to include each other (lines 5–7). Alternatively, if a block is found with ℓ_i in its `KILL` set, the both blocks need to be removed from each other's `KILL` set (lines 8–10)—we know that the definitions are different from the check in line 3, and statements that define different variables can't kill each other. Finally, the definition in S is updated to the new definition from C (line 11).

After running this procedure, S will contain the `KILL` sets for each block, and the `GEN` set for block ℓ will just be $\{\ell\}$ if $d_\ell \neq \perp$, else $\emptyset$. So, since we have the `GEN` and `KILL` sets for every block in the new program, we can compute the reaching definitions for the new version of the program using the usual algorithm.

(c) *

* work through the algo. by hand for the test case

Table 1: Reaching definitions, problem 1(a). Definitions are denoted by their block number (source code line number); definitions made in the entry block (that is, program in put parameters) are denoted by E. We define D to be the set of all definitions. The notation $\{i\text{–}k\}$ means $\{i, i+1, \ldots, k\}$. Changes from iteration 1 to iteration 2 are highlighted in red.

Block	Gen	Kill	Init		Iteration 1		Iteration 2	
			In	Out	In	Out	In	Out
entry	$\{E\}$	∅	∅	$\{E\}$	∅	$\{E\}$	∅	$\{E\}$
1	{1}	{16}	∅	{1}	$\{E\}$	{E, 1}	$\{E\}$	{E, 1}
2	{2}	∅	∅	{2}	{E, 1}	{E, 1, 2}	{E, 1}	{E, 1, 2}
3	{3}	∅	∅	{3}	{E, 1, 2}	{E, 1–3}	{E, 1, 2}	{E, 1–3}
4	∅	∅	∅	∅	{E, 1–3}	{E, 1–3}	{E, 1–3}	{E, 1–3}
5	∅	∅	∅	∅	{E, 1–3}	{E, 1–3}	{E, 1–3}	{E, 1–3}
6	∅	∅	∅	∅	{E, 1–3}	{E, 1–3}	{E, 1–3}	{E, 1–3}
8	∅	∅	∅	∅	{E, 1–3}	{E, 1–3}	{E, 1–3}	{E, 1–3}
9	{9}	{12, 20}	∅	{9}	{E, 1–3}	{E, 1–3, 9}	{E, 1–3}	{E, 1–3, 9}
10	{10}	{13, 18}	∅	{10}	{E, 1–3, 9}	{E, 1–3, 9, 10}	{E, 1–3, 9}	{E, 1–3, 9, 10}
12	{12}	{9, 20}	∅	{12}	{E, 1–3}	{E, 1–3, 12}	{E, 1–3}	{E, 1–3, 12}
13	{13}	{10, 18}	∅	{13}	{E, 1–3, 12}	{E, 1–3, 12, 13}	{E, 1–3, 12}	{E, 1–3, 12, 13}
15	∅	∅	∅	∅	{E, 1–3, 9, 10, 12, 13, 18, 20}	{E, 1–3, 9, 10, 12, 13, 18, 20}	{E, 1–3, 9, 10, 12, 13, 16, 18, 20}	{E, 1–3, 9, 10, 12, 13, 16, 18, 20}
16	{16}	{1}	∅	{16}	{E, 1–3, 9, 10, 12, 13, 18, 20}	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}	{E, 1–3, 9, 10, 12, 13, 16, 18, 20}	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}
17	∅	∅	∅	∅	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}
18	{18}	{10, 13}	∅	{18}	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}	{E, 2, 3, 9, 12, 16, 18, 20}	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}	{E, 2, 3, 9, 12, 16, 18, 20}
20	{20}	{9, 12}	∅	{20}	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}	{E, 2, 3, 10, 13, 16, 18, 20}	{E, 2, 3, 9, 10, 12, 13, 16, 18, 20}	{E, 2, 3, 10, 13, 16, 18, 20}
23	∅	∅	∅	∅	{E, 1–3, 9, 10, 12, 13, 18, 20}	{E, 1–3, 9, 10, 12, 13, 18, 20}	{E, 1–3, 9, 10, 12, 13, 16, 18, 20}	{E, 1–3, 9, 10, 12, 13, 16, 18, 20}
exit	∅	D	∅	∅	{E, 1–3, 9, 10, 12, 13, 18, 20}	∅	{E, 1–3, 9, 10, 12, 13, 16, 18, 20}	∅

Table 2: Reachable uses, problem 1(b). If a source code line contains only one use, the use is denoted by the line number. Otherwise, the notation $\ell.i$ means the ith use in line ℓ, and $\{\ell.i–k\}$ means $\{\ell.i, \ell.(i+1), \ldots \ell.k\}$. Changes from iteration 1 to iteration 2 are highlighted in red.

Block	Gen	Kill	Init		Iteration 1		Iteration 2	
			In	Out	In	Out	In	Out
entry	∅	{4, 8, 9, 13, 17.3, 17.6}	∅	{4, 8, 9, 13, 17.3, 17.6}	{4, 8, 9, 13, 17.3, 17.6}	∅	{4, 8, 9, 13, 17.3, 17.6}	∅
1	∅	{17.1–2, 18, 20, 23}	∅	∅	{4, 8, 9, 13, 17.3, 17.6, 23}	{4, 8, 9, 13, 17.3, 17.6}	{4, 8, 9, 13, 17.3, 17.6, 23}	{4, 8, 9, 13, 17.3, 17.6}
2	∅	{6}	∅	∅	{4, 6, 8, 9, 13, 17.3, 17.6, 23}	{4, 8, 9, 13, 17.3, 17.6, 23}	{4, 6, 8, 9, 13, 17.3, 17.6, 23}	{4, 8, 9, 13, 17.3, 17.6, 23}
3	∅	{12, 15.3}	∅	∅	{4, 6, 8, 9, 12, 13, 15.3, 17.3, 17.6, 23}	{4, 6, 8, 9, 13, 17.3, 17.6, 23}	{4, 6, 8, 9, 12, 13, 15.3, 17.3, 17.6, 23}	{4, 6, 8, 9, 13, 17.3, 17.6, 23}
4	{4}	∅	∅	{4}	{6, 8, 9, 12, 13, 15.3, 17.3, 17.6, 23}	{4, 6, 8, 9, 12, 13, 15.3, 17.3, 17.6, 23}	{6, 8, 9, 12, 13, 15.3, 17.3, 17.6, 23}	{4, 6, 8, 9, 12, 13, 15.3, 17.3, 17.6, 23}
5	∅	∅	∅	∅	{6}	{6}	{6}	{6}
6	{6}	∅	∅	{6}	∅	{6}	∅	{6}
8	{8}	∅	∅	{8}	{9, 12, 13, 15.3, 17.3, 17.6, 23}	{8, 9, 12, 13, 15.3, 17.3, 17.6, 23}	{9, 12, 13, 15.3, 17.3, 17.6, 23}	{8, 9, 12, 13, 15.3, 17.3, 17.6, 23}
9	{9}	{15.2, 16.1, 17.4–5}	∅	{9}	{15.2–3, 16, 17.3–6, 23}	{9, 15.3, 17.3, 17.6, 23}	{15.2–3, 16, 17.3–6, 23}	{9, 15.3, 17.3, 17.6, 23}
10	∅	{15.1, 16.2}	∅	∅	{15.1–3, 16.1–2, 17.3–6, 23}	{15.2–3, 16.1, 17.3–6, 23}	{15.1–3, 16.1–2, 17.3–6, 23}	{15.2–3, 16.1, 17.3–6, 23}
12	{12}	{15.2, 16.1, 17.4–5}	∅	{12}	{13, 15.2–3, 16, 17.3–6, 23}	{12, 13, 15.3, 17.3, 17.6, 23}	{13, 15.2–3, 16, 17.3–6, 23}	{12, 13, 15.3, 17.3, 17.6, 23}
13	{13}	{15.1, 16.2}	∅	{13}	{15.1–3, 16.1–2, 17.3–6, 23}	{13, 15.2–3, 16.1, 17.3–6, 23}	{15.1–3, 16.1–2, 17.3–6, 23}	{13, 15.2–3, 16.1, 17.3–6, 23}
15	{15.1–3}	∅	∅	{15.1–3}	{15.1–3, 16.1–2, 17.3–6, 23}	{15.1–3, 16.1–2, 17.3–6, 23}	{15.1–3, 16.1–2, 17.3–6, 23}	{15.1–3, 16.1–2, 17.3–6, 23}
16	{16.1–2}	{17.1–2, 18, 20, 23}	∅	{16}	{15.1–3, 17.1–6, 18, 20}	{15.1–3, 16.1–2, 17.3–6}	{15.1–3, 16.1–2, 17.1–6, 18, 20, 23}	{15.1–3, 16.1–2, 17.3–6}
17	{17.1–6}	∅	∅	{17.1–6}	{15.1–3, 18, 20}	{15.1–3, 17.1–6, 18, 20}	{15.1–3, 16.1–2, 17.3–6, 18, 20, 23}	{15.1–3, 16.1–2, 17.1–6, 18, 20, 23}
18	{18}	{15.1, 16.2}	∅	{18}	{15.1–3}	{15.1, 15.3, 18}	{15.1–3, 16.1–2, 17.3–6, 23}	{15.2–3, 16.1, 17.3–6, 18, 23}
20	{20}	{15.2, 16.1, 17.4–5}	∅	{20}	{15.1–3}	{15.1, 15.3, 20}	{15.1–3, 16.1–2, 17.3–6, 23}	{15.1, 15.3, 16.2, 17.3, 17.6, 20, 23}
23	{23}	∅	∅	{23}	∅	{23}	∅	{23}

Table 3: Definition-use pairs, problem 1(c). If a source code line contains only one use, the use is denoted by the line number. Otherwise, the notation $\ell.i$ means the ith use in line ℓ. Definitions in the entry block (that is, program parameters) are denoted by E.

Use	Variable	Definitions
4	`x`	$\{E\}$
6	`errval`	{2}
8	`x`	$\{E\}$
9	`x`	$\{E\}$
12	`eps`	{3}
13	`x`	$\{E\}$
15.1	`x2`	{10, 13, 18}
15.2	`x1`	{9, 12, 20}
15.3	`eps`	{3}
16.1	`x1`	{9, 12, 20}
16.2	`x2`	{10, 13, 18}
17.1	`x3`	{16}
17.2	`x3`	{16}
17.3	`x`	$\{E\}$
17.4	`x1`	{9, 12, 20}
17.5	`x1`	{9, 12, 20}
17.6	`x`	$\{E\}$
18	`x3`	{16}
20	`x3`	{16}
23	`x3`	{1, 16}