CS 618 Program Analysis: Practice Questions

Uday P. Khedker

(http://www.cse.iitb.ac.in/˜uday)

Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

May 10, 2011

About These Questions

These questions are a copyrighted material (©2009 Uday Khedker) and have been designed for the course CS 618: Program Analysis offered at the Department of Computer Science and Engineering, IIT Bombay. They are made available only for academic use as material accompanying the book *Data Flow Analysis: Theory and Practice* by Khedker, Sanyal, and Karkare. Eventually they are expected to be included in the next edition of the book. Additional material accompanying the book can be found at the book page http://www.cse.iitb.ac.in/ uday/dfaBook-web.

Topic	Number of Questions	Page Number
Bit vector data flow frameworks	15	2
Theoretical abstractions in data flow analysis	6	8
General data flow frameworks	27	9
Interprocedural data flow analysis	14	16

   We would be happy to receive suggestions for corrections and improvements in questions. We also welcome new questions or ideas for new questions.

1 Bit Vector Data Flow Frameworks

1. Consider the available expressions analysis framework for the following control flow graph.
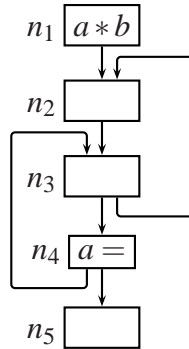
(a) What should be *BI*? Which program point should it be associated with? What should be the initialization for all internal nodes?

(b) Perform available expression analysis using round robin iterative algorithm by traversing the graph in the forward direction. Show the values for each node in each iteration. How many iterations are needed for this analysis?

(c) Repeat the analysis by traversing the graph in the backward direction. In backward traversal, compute Out_n before computing In_n. How many iterations do you need now?

(d) Show the trace of the worklist iterative algorithm for the given control flow graph. Assume that the work list follows FIFO (First in First Out) policy.

Step No.	Program Point Selected	Remaining Work list	Data Flow Value	Program Point(s) Added	Resulting Work list

(e) Compare the work performed by the two algorithms for the given control flow graph in terms of the total number of nodes processed until convergence is established.

Let one unit of work be defined as processing one node (i.e. computing *In* and *Out* for the node). Include the initialization of In_i and Out_i as one visit to node i. Compare the work done by

  (i) Round robin analysis with forward traversal
  (ii) Round robin analysis with backward traversal
 (iii) Work list based method

Which algorithm involves less work? Why?

(f) Does your work list contain nodes that need not be included in it? Can you suggest how the worklist algorithm can be made more efficient in terms of nodes processed?

(g) What is the depth of this graph? What is its width for available expressions analysis with forward traversal in a round robin method? What is its width for if the direction of traversal is changed to backward? In each case, identify the width defining information flow path.

2. Consider the following dump by gcc for a C program.

```
# BLOCK 2                          # BLOCK 5
# PRED: ENTRY (fallthru)           # PRED: 3 (false)
e = a * b;                         <L2>:;
d = b * c;                         d = 3;
# SUCC: 3 (fallthru)               # SUCC: 6 (fallthru)


# BLOCK 3                          # BLOCK 6
# PRED: 2 (fallthru) 6 (true)      # PRED: 4 (fallthru) 5 (fallthru)
<L0>:;                             <L3>:;
e = c * d;                         c = d * e;
if (c < d) goto <L1>;              c = a * b;
          else goto <L2>;          if (c < d) goto <L0>;
# SUCC: 4 (true) 5 (false)                   else goto <L3>;
                                   # SUCC: 3 (true) 7 (false)

# BLOCK 4
# PRED: 3 (true)                   # BLOCK 7
<L1>:;                             # PRED: 6 (false)
c = 2;                             <L4>:;
goto <bb 6> (<L3>);                D.1547 = d * e;
# SUCC: 6 (fallthru)               return D.1547;
                                   # SUCC: EXIT
```

(a) Draw the control flow graph. Which control construct may have been used in the input C program?

(b) Perform available expressions analysis. Clearly show Gen_n and $Kill_n$ and the initial values of In_n and Out_n, and the values of In_n and Out_n in each iteration of analysis.

(c) Show common subexpression elimination using the above data flow values.

3. Perform reaching definitions analysis on the following CFG using round-robin algorithm. List *Gen* and *Kill* for each noode and show the complete trace of all *In*/*Out* values in each iteration. Do you find any scope of copy propagation?

n_1 $\boxed{a = 3}$

n_2 $\boxed{b = 5}$ $\boxed{d = a + 1}$ n_3

n_4 $\boxed{c = b}$

n_5 $\boxed{d = 7}$

n_6 $\boxed{print\ a,b,c,d}$
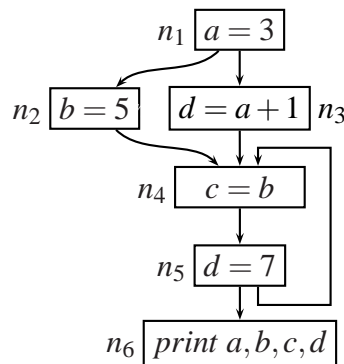
4. The following function computes the mth fibonacci number for a given $m \geq 1$. Construct its control flow graph and perform reaching definitions analysis. Do you find any scope for copy propagation?

```
int fib(unsigned int m)
{    int f0, f1, f2, i;

     f0 = 0;
     f1 = 1;

     if (m <= 1)
         f2 = m;
     else
     {   for (i=2; i<=m; i++)
         {   f2 = f0 + f1;
             f0 = f1;
             f1 = f2;
         }
     }
     return f2;
}
```

5. Construct an instance of live variables analysis which requires four iterations to converge regardless of whether the control flow graph is traversed in forward direction (i.e. reverese depth first order), or in the backward direction (i.e. depth first order).

   • You need to construct a control flow graph and show the data flow values in each iteration for both directions of traversal.

   • Convergence in four iterations means that the data flow values computed in the first three iterations are different but the data flow values computed in the fourth iterations are identical to the data flow values computed in the third iteration.

   • Assume that in a forward traversal, In_n is computed before Out_n whereas in a backward traversal, Out_n is computed before In_n.

6. Construct a C program for which available expressions analysis for which the minimum fixed point assignment is different from the maximum fixed point assignment and both computations require four iterations to converge (three for convergence and one for detecting convergence). You are not allowed to use goto statements in your C programs. Use at most two expressions.

   Assume that the control flow graph is traversed in forward direction (i.e. reverse depth first order). Show the data flow values in each iteration.

7. Construct an instance of live variables analysis such that

   • the maximum fixed point assignment for the instance is different from its minimum fixed point assignment, and

4

- the computation of maximum fixed point assignment converges in 3+1 iterations (3 for computation and 1 for discovering convergence), and

- the computation of minimum fixed point assignment also converges in 3+1 iterations.

Assume that analysis is performed by making a backward traversal over the control flow graph.

8. Construct a C program for which available expressions analysis requires four iterations to converge. You are not allowed to use `goto` statements in your C programs. Assume that the control flow graph is traversed in forward direction (i.e. reverese depth first order). Show the data flow values in each iteration.

9. Recall that it is possible to perform total and partial availability analyses together by using the component lattice shown below. Perform this analysis for the following program using the round robin iterative method. Assume that the control flow graph is traversed in reverse postorder (i.e. forward direction). Is the result of your analysis any different from the results obtained by independent analyses? Why?
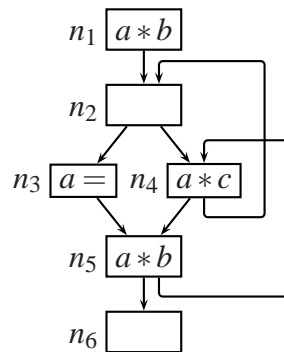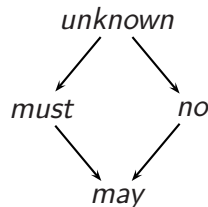
10. *Backward Slicing* finds the smallest set of statements relevant to a given computation at a program point of interest (known as the *Slicing Criterion*). The first step in slicing is to construct a set of variables that are directly relevant to the computation at the slicing criterion.

For each edge $i \rightarrow j$ in the CFG, a variable v is relevant at i (denoted $v \in \textit{Relevant}(i)$) if:

$$(v \in \textit{Relevant}(j) \wedge v \notin \textit{Def}(i)) \vee (v \in \textit{Ref}(i) \wedge \textit{Def}(i) \cap \textit{Relevant}(j) \neq \emptyset)$$
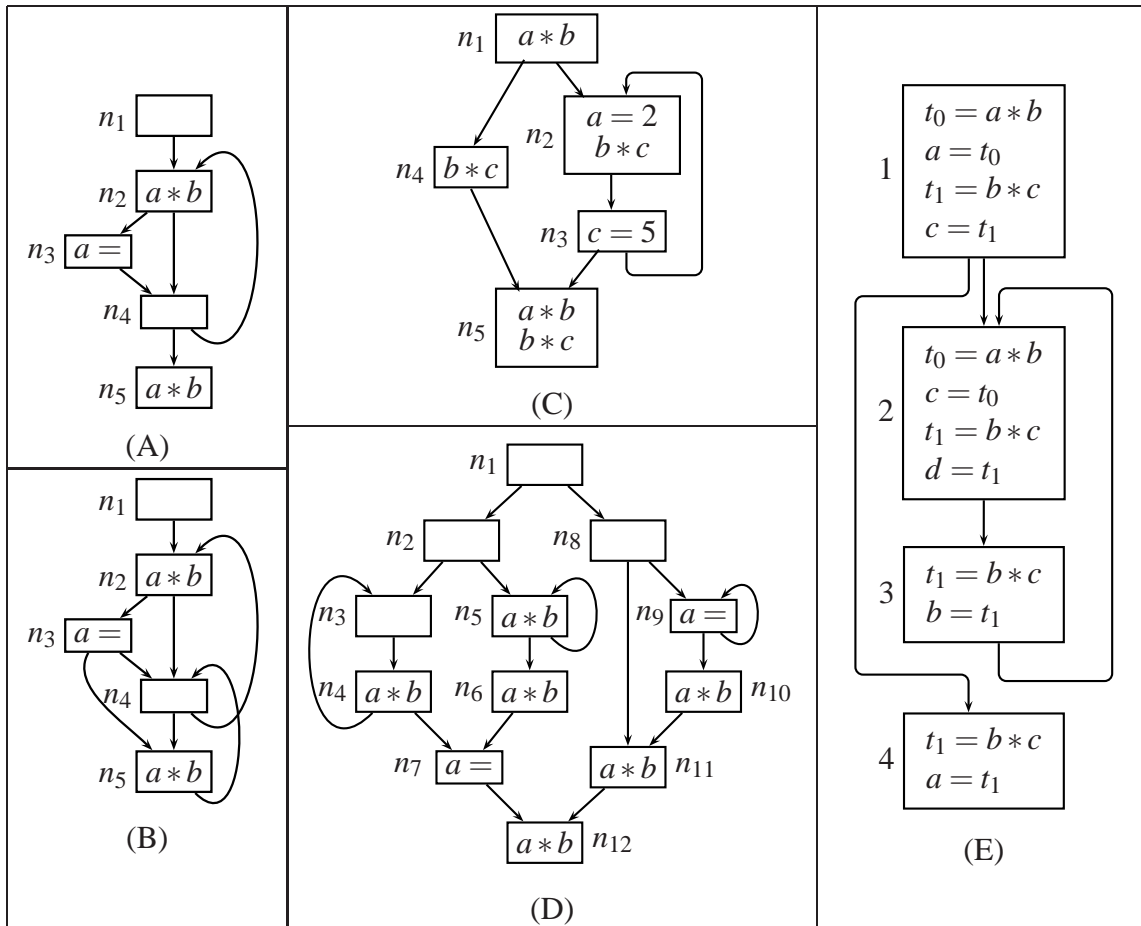
Consider the following example:

Program	Directly relevant variables
| ```
L1: b = 5;
L2: c = 10;
    if (<cond>)
L3:    a = a + c;
    else
L4:    a = a + 1;
L5: return (a*b);
    // slicing criterion
``` | • $b \in \textit{Relevant}(L4)$ because $b \in \textit{Relevant}(L5)$ and $b \notin \textit{Def}(L4)$ <br><br> • $b \in \textit{Relevant}(L2)$ because $b \in \textit{Relevant}(L4)$ and $b \notin \textit{Def}(L2)$ <br><br> • $a \in \textit{Relevant}(L4)$, because $a \in \textit{Ref}(L4)$ and $a \in \textit{Def}(L4) \cap \textit{Relevant}(L5)$ <br><br> • $a,c \in \textit{Relevant}(L3)$, because $a,c \in \textit{Ref}(L3)$ and $a \in \textit{Def}(L4) \cap \textit{Relevant}(L5)$ |

Define the data-flow equations for computing $Relevant(i)$. Please provide:

(a) Definitions of Gen, $Kill$, In, Out.

(b) Interpretation of BI for the problem with appropriate assumptions.

(c) Lattice, confluence operation, and default initialization.

(d) Comment on separability of the problem with illustration.

11. Perform partial redundancy elimination for the following control flow graphs. Use bit vector notation for convenience.

(a) List the values of Gen_n, $Kill_n$, $AntGen_n$.

(b) Compute $PavIn_n/PavOut_n$, and $AvIn_n/AvOut_n$ for each node n.

(c) Show the values of In_n and Out_n for each node n in each iteration.

(d) Show the values of $Redundant_n$ and $Insert_n$.

(e) Show the hoisting paths and explain the resulting transformation intuitively. If no hoisting is possible, explain why it is not possible.

(f) What is the width of the CFG for PRE? Identify the width determining path.

6

(g) Computer $AntIn_n$/$AntOut_n$. Do you observe any relationship between these values and the In_n/Out_n values for PRE?

12. Construct a C program for which PRE requires 5 iterations of round robin iterative analysis (4 to converge and 1 to detect convergence). Assume that the control flow graph is traversed in postorder (i.e. backward direction). You have to meet the following design constraints.

- The program should not contain goto statements or nested loops.

- Design the program to contain a single expression for analysis.

- The structure of the program should not resemble the structures that we have seen in the class.

Simple programs will receive more credit.

(a) Write the program and draw the corresponding control flow graph.

(b) Show the final values of available expressions and partially available expressions analyses.

(c) Perform work list based PRE analysis for your program. Show the trace of analysis in the following format. Assume that the work list follows FIFO (First in First Out) policy.

Step No.	Program Point Selected	Remaining Work list	Data Flow Value	Program Point(s) Added	Resulting Work list

(d) Show an ifp leading to 5 iterations of round robin analysis. Indicate the step numbers in your trace of the work list algorithm that cover this ifp.

13. Construct a C program for which MFP and LFP assignments for PRE are different.

14. Let N be the set of nodes of a control flow graph. A node $x \in N$ dominates a node $y \in N$, denoted $x \, dom \, y$, if and only if x appears on every path from *Start* to y. The dominance relation *dom* is reflexive, transitive, and antisymmetric. An example of dominator information has been shown below.

Node	Dominators
n_1	$\{n_1\}$
n_2	$\{n_1, n_2\}$
n_3	$\{n_1, n_2, n_3\}$
n_4	$\{n_1, n_2, n_4\}$
n_5	$\{n_1, n_2, n_5\}$

Define a data flow analysis for computing dominators of each node. The result on analysis should be to compute, for a given node n, a set $domIn_n$ which is the set of dominators of n excluding n and $domOut_n$ which includes n also.

In particular, you need to specify *BI*, the program point with which *BI* should be associated, the flow functions in terms of Gen_n and $Kill_n$, the confluence operation, and tie them into data flow equations. Is your framework a bit vector framework?

15. Let the set of dominators of node y be denoted by $dom(y)$. An immediate dominator of a node y is a node $z \in dom(y)$ such that $\forall w \in (dom(y) - \{y\}), \ w \in dom(z)$. Intuitively, an immediate dominator of y is the dominator that is "closest" to y but is not y.

    Define a data flow analysis based method to compute immediate dominators.

2   Theoretical Abstractions in Data Flow Analysis

1. Is the following poset a lattice? If not, explain whether it is a meet semilattice or a join semilattice. If yes, explain whether it is a complete lattice or a bounded lattice.

2. Assume that L is a complete lattice in which all strict chains are bounded. Given a monotonic function $f : L \mapsto L$, show that

$$\exists k \geq 0 \text{ such that } f^{k+1}(\bot) = f^k(\bot) \text{ and } \forall j < k, \ f^{j+1}(\bot) \neq f^j(\bot)$$

   Prove that $f^k(\bot)$ is the least fixed point f.

3. Given $f : L \mapsto L$, prove that the following two definitions of monotonicity are equivalent.

$$\forall x, y \in L, \quad x \sqsubseteq y \ \Rightarrow \ f(x) \sqsubseteq f(y)$$
$$\forall x, y \in L, \quad f(x \sqcap y) \ \sqsubseteq \ f(x) \sqcap f(y)$$

4. Let f_ρ and $f_{i \to j}$ denote the flow functions associated with path ρ and edge $i \to j$, respectively. Let $Paths(i)$ denote the paths from *Entry* to i. Given the definitions of MFP and MoP,

$$MoP(i) \ = \ \bigsqcap_{\rho \in Paths(i)} f_\rho(BI)$$

$$MFP(i) \ = \ \bigsqcap_{p \in Pred(i)} f_{p \to i}(MFP(p))$$

   show that $\forall i, MFP(i) = MoP(i)$ for distributive frameworks.

5. Construct an instance of a framework by describing a lattice L, a $\sqsubseteq$ relation, and a $\sqcap$ operation and by constructing a control flow graph with the associated flow functions. Assume the following data flow equations:

$$
\begin{aligned}
In_n &= \begin{cases} BI & n \text{ is } \textit{Start} \text{ block} \\ \underset{p \in pred(n)}{\sqcap} Out_p & \text{otherwise} \end{cases} \\
Out_n &= f_n(In_n)
\end{aligned}
$$

The constructed instance should have the following characteristics:

- L must be a complete lattice.

- Every flow function f_n must be distributive.

- Select a BI, an initialization, and flow functions such that the round robin iterative algorithm should *not* terminate for this instance.

Does your instance have a fixed point assignment? If yes, explain why the round robin iterative algorithm does not terminate. If it does not have a fixed point assignment, explain why.

6. Recall that a data flow framework $\langle L, \sqcap, F \rangle$ is k-bounded if

$$
\forall f \in F, \forall x \in L, \; f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap \ldots = x \sqcap f(x) \sqcap f^2(x) \sqcap \ldots \sqcap f^{k-1}(x)
$$

Is the boundedness parameter k related to the the component lattice? The overall lattice? Justify your answer in the context of

(a) Constant propagation

(b) Available Expressions Analysis

(c) Combined Total and Partially Available Expressions Analysis

3 General Data Flow Frameworks

1. Perform possibly uninitialized variables analysis for the following CFG using round-robin algorithm and show the values of *In*/*Out* in each iteration.

2. Perform faint variables analysis for the following CFG round-robin algorithm and show the values of *In*/*Out* in each iteration.

3. (a) Construct a CFG containing only one back edge such that

   (i) Faint variables analysis requires four iterations of backward traversal for convergence (i.e. values in 2nd and 3rd iterations are different but the values in 3rd and 4th iteration are identical).

   (ii) Does the result of your analysis lead to dead code elimination?

   (iii) Perform live variables analysis on the same graph. How many iterations does it take? What is the relationship between dead variables as discovered by live variables analysis and faint variables?

   (b) Construct another example of faint variables analysis which has 3 back edges but requires only two iterations to converge.

4. Construct an instance of possibly uninitialized variables analysis such that

   • the maximum fixed point assignment for the instance is different from its minimum fixed point assignment, and

   • the computation of maximum fixed point assignment converges in 3+1 iterations (3 for computation and 1 for discovering convergence), and

   • the computation of minimum fixed point assignment also converges in 3+1 iterations.

   Assume that analysis is performed by making a forward traversal over the control flow graph.

5. Consider the constant propagation framework for the following control flow graph.

10

(a) Compute the maximum fixed point for constant propagation for the give control flow graph. How many iterations does it need?

(b) Observe that f_{n_2} does not have a fixed point. Still the data flow equations compute a fixed point. How?

6. Perform constant propagation for the following program using the round robin iterative method. Assume that the control flow graph is traversed in the forward direction.

7. (a) Perform constant propagation for the following two programs using round-robin algorithm. Show the complete trace (in tabular form) of all *In/Out* values in each iteration. Clearly specify the number iterations in each case.

(b) Repeat constant propagation analysis for the same CFG by changing the statements in selected nodes as described below: n_3 contains $a = b$, n_5 contains $b = c$, n_6 contains $c = d$, n_7 contains $d = 2$. How does the number of iterations changes?

8. Recall that copy constant propagation is a simplified version of constant propagation in expressions are not evaluated; whenever expression evaluation is required, the result of the expression is assumed to be $\bot$. Perform copy constant propagation for the following CFG.

9. Create an example of constant propagation for which the *MoP*, the *MFP*, and the *LFP*, are all different from each other.

10. Signs Analysis involves determining signs of the possible values of variables at each program point as positive non-zero $(+)$, negative $(-)$, and zero (0). An example of a program and the desired data flow information is:

Define a data flow analysis to compute signs information for each variable. Clearly define the lattice, the confluence operation, the *BI*, the initialization for internal nodes, and data flow

equations. For the purpose the flow functions, assume that the arithmetic expressions are restricted to both binary and unary versions of operators $+$ and $-$. Define the flow functions in terms of $ConstGen_n$, $ConstKill_n$, $DepGen_n(X)$, and $DepKill_n(X)$.

Is your framework separable? Distributive?

11. Consider the following program fragments for points-to analysis.

```
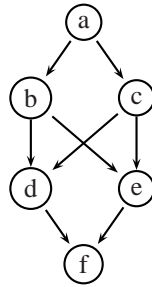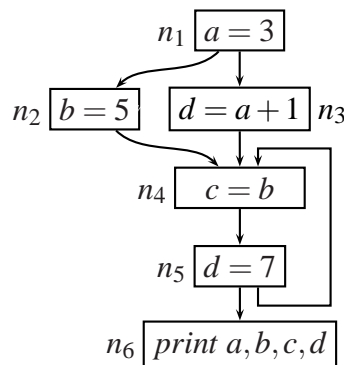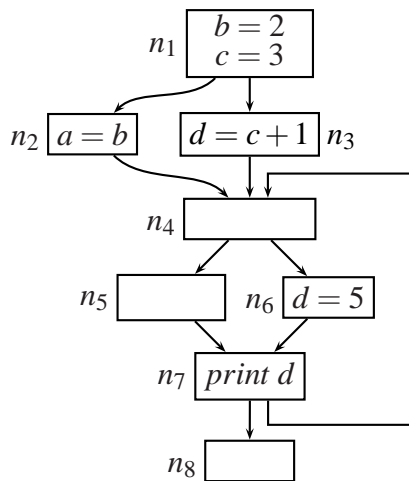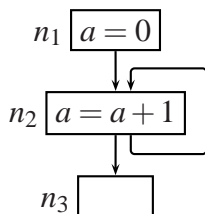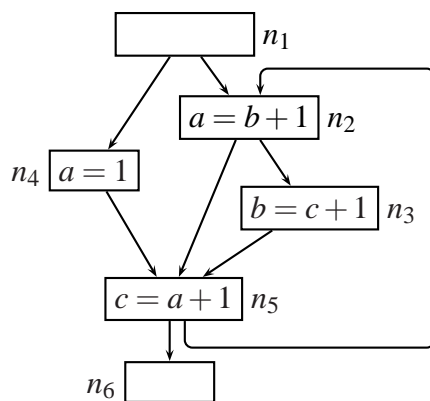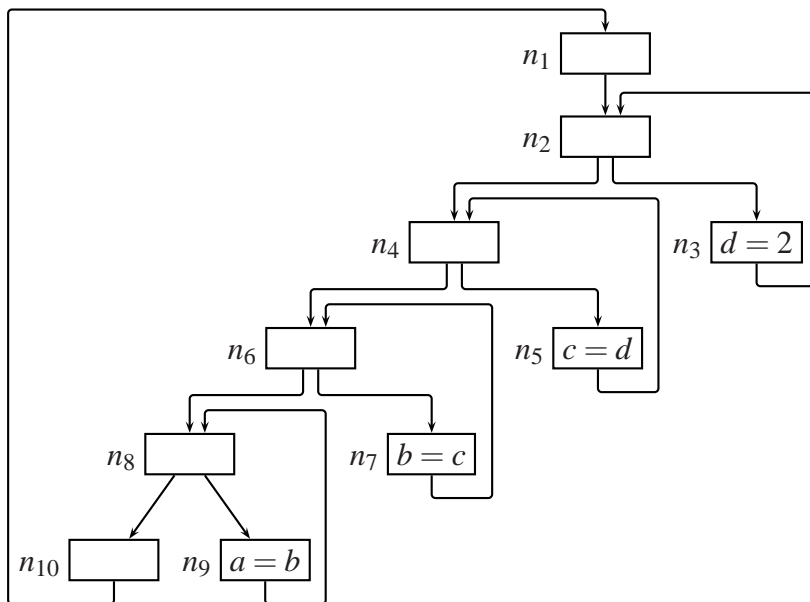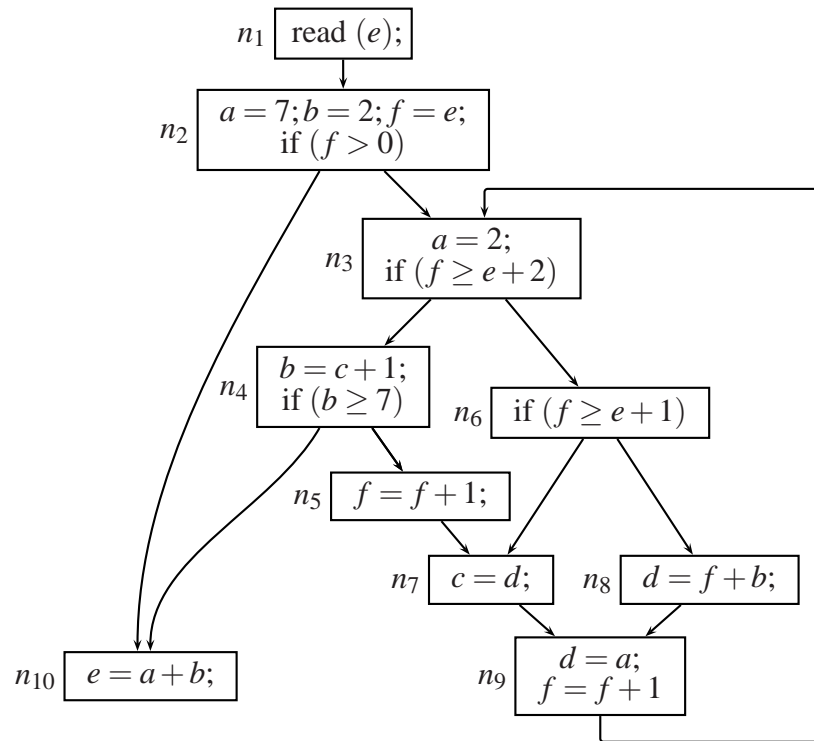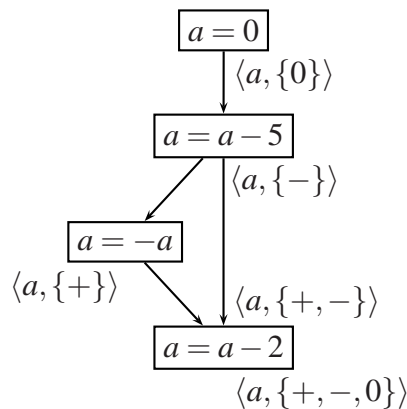if (...)                      if (...)
      p = &x;                       p = &y;
else                          else
      p = &y;                       p = &x;
x = &a;                       y = &b;
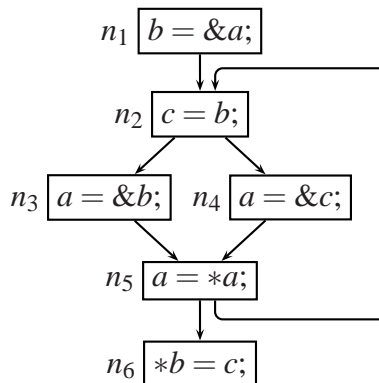y = &b;                       x = &a;
*p = &c;                      *y = &a;
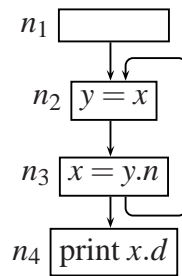*y = &a;                      *p = &c;
```

   (a) Show the result of flow insensitive May points-to analysis.

   (b) Show the result of flow sensitive May points-to analysis.

   (c) Show the result of flow sensitive Must points-to analysis.

12. Perform independent May and Must points-to analyses on the following control flow graph. When performing May points-to analysis, assume conservative Must points-to information (no pointer points to any location). When performing Must points-to analysis, assume conservative May points-to information (a pointer points to every location).

13. Is May points-to analysis framework k-bounded? If yes, what is the value of k?

14. Is Must points-to analysis framework k-bounded? If yes, what is the value of k?

15. Create an example to show the non-distributivity of May points-to analysis framework.

16. Create an example to show the non-distributivity of Must points-to analysis framework.

17. Perform liveness analysis of heap references for the following program. Clearly specify the number iterations.

18. Construct a control flow graph for this program and perform explicit liveness analysis for heap references in the following program. You have to show the In_n and Out_n in each iteration.

```
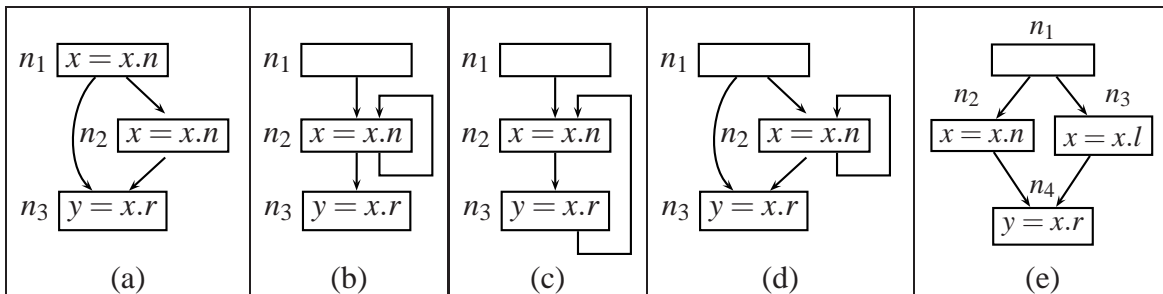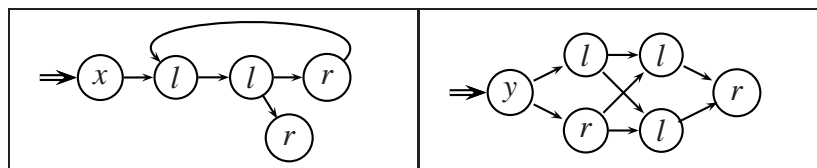bool find(int n, Tree * t)
{  found = false;
   while (t != NULL)
   {  if (n == t->n)
      {  found = true; break; }
      else if (n < t->n)
         t = t->l;
      else t = t->r;
   }
}
```

19. Consider the following programs for heap reference analysis.

   (a) Show the final explicit liveness access graphs for each node.

   (b) Liveness graphs of programs (b) and (c) are identical at node n_1. Why is this semantically correct inspite of the fact that both the programs have different structures?

20. Consider the following access graphs representing explicit liveness at some program point (perhaps in different programs).

14

Unfortunately the student who constructed these access graphs forgot to attach a statement number as a subscript to the node labels and has misplaced the programs which gave rise to these graphs. Please help her by constructing (independent) CFGs for which these access graphs represent explicit liveness at some program point in the CFGs. Please also show the liveness access graphs at all other program points in the CFGs.

21. Perform heap liveness analysis for the following program:

22. Is the access path $x \dashrightarrow n$ live at the entry of node n_4? Does explicit liveness include it? If there is a discrepancy in your observation and the result of explicit liveness analysis, please suggest how this discrepancy can be removed. If there is no discrepancy, explain why there is no discrepancy.

23. Create an example of explicit liveness analysis in HRA such that it requires five iterations. The values must change in the first four iterations and should remain constant in the fifth iteration. Draw the CFG and show the values in each iteration.

24. Construct an example to show that explicit liveness analysis is non-distributive.

25. Is it possible to get the following two liveness access graphs reaching the same program point p along two different control flow paths? Please explain your answer and describe the conclusion that you draw from your explanation.

26. Perform explicit liveness analysis for the following control flow graph. What nullification statements, if any, can be inserted and at what points?

n_1 | w = x

n_2 | while (x.data < max)

n_3 | y = x.lptr n_5 | x = x.rptr

n_4 | x = y.lptr n_6 | print x.data

27. Heap reference analysis computes explicit liveness at each program point and then aliasing information is used to compute implicitly live access paths by discovering link aliases of the explicitly live access paths. There are two ways of doing this:

- Implicitly live access paths are computed *during* explicit liveness analysis and are propagated backwards in the program along with the explicitly live access paths.

- Implicitly live access paths are computed *after* explicit liveness analysis and are not propagated backwards.

Perform heap reference analysis for the following program to show that for correctness it is necessary to compute implicitly live access paths during explicit liveness analysis. Also show how it results in imprecise liveness information. (Since this program does not have loops, there is no need to construct access graphs. Give your answer in terms of sets of access paths for liveness and pairs of access paths for aliasing.)

n_1 | x = y n_2

n_3 | x.n = z

n_4 | Use y.n.n.d

4   Interprocedural Data Flow Analaysis

1. Consider the following program.

```
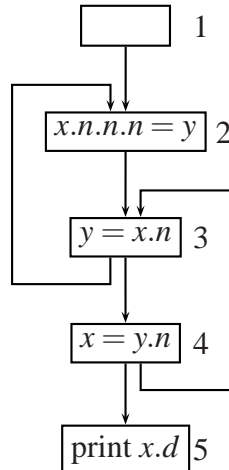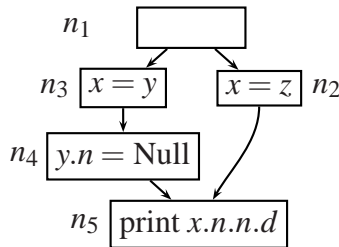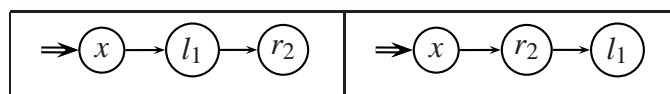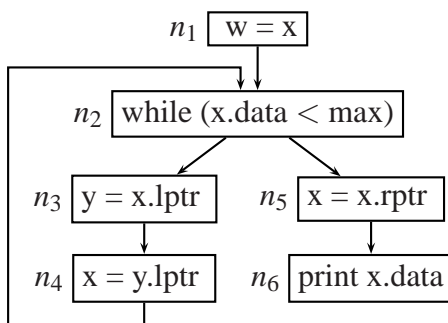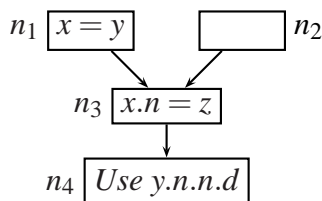int a, b, c;        void p()
int main()          {  if (...)
{  c = a*b             {  a = a*b;
   p();                   p();
}                      }
                    }
```

(a) Draw control flow graphs of the two procedures and perform available expressions analysis using functional approach. Clearly show $\Phi_{main}(n)$ and $\Phi_p(n)$ for each block n in procedures main and p. You have to provide complete trace of computation of these flow functions in a tabular form.

(b) Construct a supergraph for the program and perform interprocedural available expressions analysis using the call strings approach assuming that a call site needs to appear in a call string at most thrice. Use the work list approach and show the trace of your computation in the following format:

Step No.	Selected Node	Qualified Data Flow Value		Remaining Work List	
		IN_n	OUT_n	Intraprocedural Nodes	Call/Return Nodes

(c) Use the modified call strings approach and perform interprocedural available expressions analysis. Clearly indicate representation and regeneration of call strings. Recall that the algorithm requires that

- intraprocedural nodes are processed before any call/return node in the worklist, and
- call nodes are processed before any return node in the work list.

Do you have to construct fewer call strings?

2. Perform interprocedural available expressions analysis for the following program using the functional approach. Draw the control flow graphs and construct summary flow functions by iterating over them. Show the trace of construction in a tabular form.

```
1.     void main(void)
2.     {
3.         c = a*b;
4.         p();
5.         printf ("%d\n", c);
6.     }
```

```
7.     void p (void)
8.     {   if (...)
9.         {   a = a*b;
10.            p();
11.        }
12.        else if (...)
13.        {   c = a * b;
14.            p();
15.            printf ("%d\n", c);
16.        }
17.        else
18.            ; /* ignore */
19.    }
```

What does your analysis conclude about the availability of $a*b$ at line 18? At line 6? If it is not available, please show an interprocedurally valid path along with the expression is killed.

3. Construct an instance of interprocedural available expressions analysis to show that the result of context sensitive interprocedural analysis is more precise than the result of context insensitive interprocedural analysis.

4. Construct an example of liveness analysis to show that context sensitive interprocedural data flow analysis is more precise compared to context insensitive interprocedural data flow analysis. You do not have to perform analysis but only show possible interprocedurally invalid paths in a supergraph which, if not avoided, lead to imprecision for your instance of interprocedural liveness analysis.

    Your program must have a single variable which must be a global variable. Use only two procedures, one of which must be *main* and is not called by any procedure.

5. Construct a supergraph for the program shown below. Perform interprocedural available expressions analysis using call-strings approach. Use the work list approach and give a table of the trace of the algorithm.

Main		**P**		**Q**		**T**	
1.	start	1.	start	1.	start	1.	start
2.	call P	2.	$a*b$	2.	if() then	2.	call Q
3.	$c*d$	3.	if() then	2.	$a = 5$	3.	end
4.	call T	4.	call Q	3.	else		
5.	end	5.	else	4.	$b = 6$		
		6.	call T	6.	end		
		7.	$c = 5$				
		8.	end				

6. Recall that the 3 occurrences bound on call strings for interprocedural data flow analysis of bit vector frameworks is a general program independent bound and fewer than 3 occurrences of any call sites in every call string may be sufficient for some programs. Using the reasoning for 3 occurrences bound, explain why a single occurrence of any call site is sufficient for computing a precise solution for available expressions analysis of the following program. (You have to explain this without performing data flow analysis.)

7. Is $a * b$ available immediately after c_1 in the following supergraph? If it is not available, identify and interprocedural valid information flow path that causes it to be un-available. How is it covered by the functional method? By the call strings method?

8. Use the modified call strings method to perform interprocedural available expressions analysis of the following program.

Is expression $a + b$ available at R_2? At R_3? If it is not available, please show an interprocedurally valid path along with the expression is killed.

9. Explain using the staircase diagram why 3 occurrences of a call site are sufficient in any call string for performing interprocedural data flow analysis of bit vector data flow frameworks.

10. Consider the following program

(a) Perform call strings based interprocedural copy constant propagation using the modified call strings method for the following supergraph. Show the trace of work-list algorithm.

(b) Perform copy constant propagation using the functional approach. Show the constructed summary flow functions.

11. The *approximate call strings approach* maintains k-length suffixes of call strings instead of full call strings.

When length of a call string σ reaching a call nodes C_i in procedure P is less then m, c_i is appended to σ. If length of σ is equal to m, then the first call site is removed and c_i is appended to σ, thus maintaining length m. Assume that $\langle c_j \cdot \sigma', x_j \rangle$ and $\langle c_k \cdot \sigma', x_k \rangle$ reach C_i and $|\sigma'| = m - 1$. Then $\langle \sigma' \cdot c_i, x_j \sqcap x_k \rangle$ is propagated further.

At return node R_i, the last call site c_i is removed from the incoming call string and all call sites containing a call to procedure P are attached at the first position, thereby reconstructing the call strings whose first call site was removed at C_i. Assume that $\langle \sigma' \cdot c_i, x_i \rangle$ reach R_i, then $\langle c_j \cdot \sigma', x_i \rangle$ and $\langle c_k \cdot \sigma', x_i \rangle$ are propagated further.

(a) Construct a supergraph for a non-recursive program such that when this method (with $m = 2$) is used for performing interprocedural live variables analysis, the data flow information so obtained is imprecise. Use at most two variables in your program.

(b) Repeat the above exercise for a recursive program where approximation that causes imprecision is influenced by recursion.

12. (a) Perform interprocedural constant propagation for the following program using the functional approach. Draw the control flow graphs and construct summary flow functions by iterating over them. Show the trace of construction in a tabular form.

```
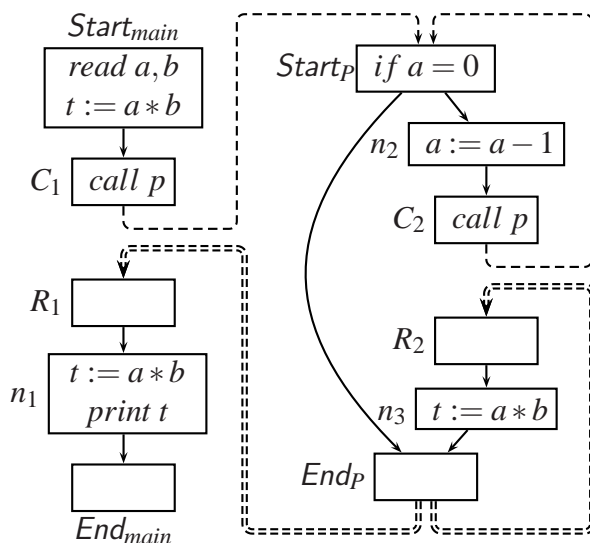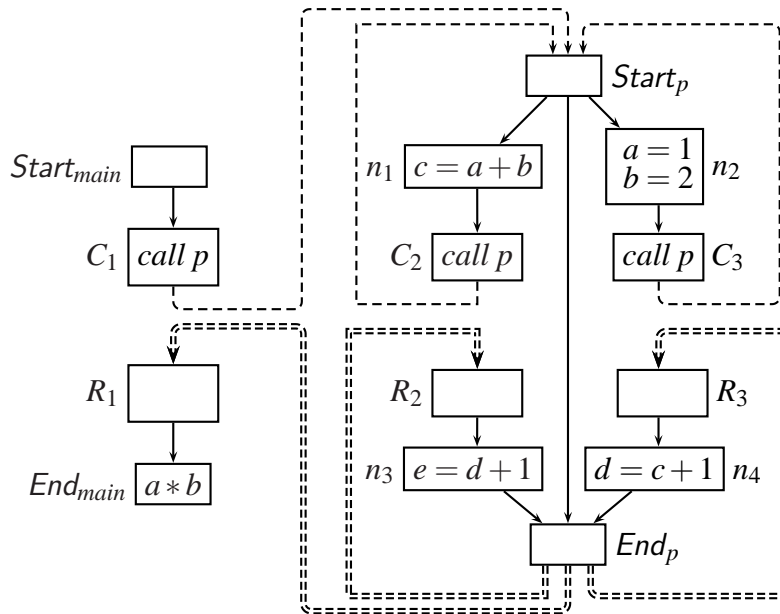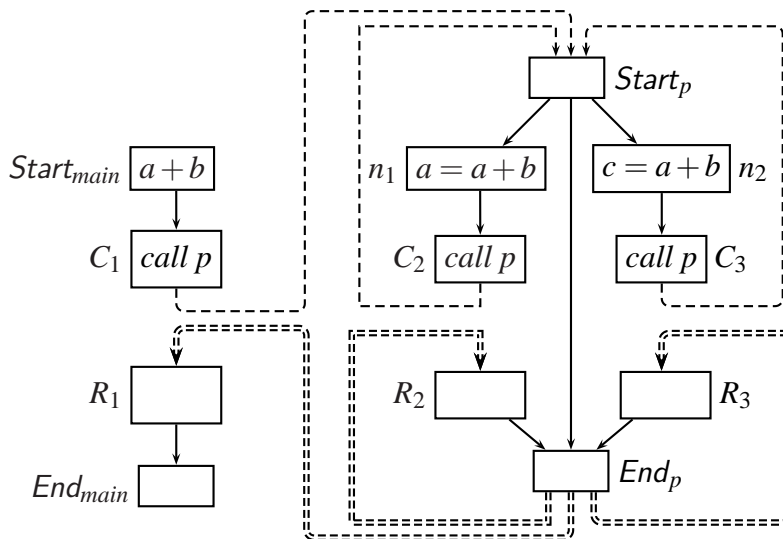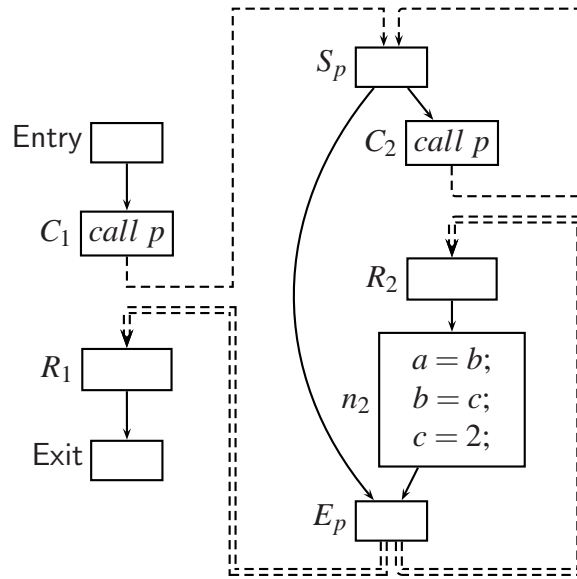1.    void main(void)
2.    {
3.        a = 5;
4.        p();
5.        printf ("%d\n", a);
6.    }
```

```
7.    void p (void)
8.    {   if (...)
9.        {   a = a+7;
10.           p();
11.           a = a-7;
12.       }
13.   }
```

Does variable a have a constant value at line 5? What does your analysis conclude about it?

(b) Draw the supergraph for the above program and explain the difficulty in performing interprocedural constant propagation over the program using the classical Sharir-Pnueli call strings method. Does the problem go away if we use the modified call strings method? Why?

(c) Use approximate call strings method with $m = 2$ for interprocedural constant propagation analysis of the above program. Does the problem you described in answer to part (b) above go away now? Why?

Does your method discover a to be constant line 5? Why?

13. Perform may points-to analysis for the following program using modified call strings method. Make conservative assumptions about must points-to information. How many call strings do you need to construct? How many call strings would be required by the Sharir-Pnueli method?

```
main()
{   x = &y;
    z = &x;
    y = &z;
    p(); /* C1 */
}
```

```
p()
{   if (...)
    {   p(); /* C2 */
        x = *x;
    }
}
```

14. In the proof of correctness of the modified call strings method, we have used the concept of a *fixed point closure bound* which is defined as the smallest $n \geq 0$ such that $\forall x \in L, \ f^{n+1}(x) = f^n(x)$.

Consider the following program for may points-to analysis. Assume that there are only three pointer variables a, b, and c. Also assume that must points-to analysis is not to be performed.

```
main()
{
    a = &b;
    b = &c;
    c = &b;
    p();
}
```

```
p()
{
    if (...)
    {
        a = *a;
        p();
    }
}
```

Does the flow function for the cyclic call sequence of the program have a fixed point closure bound? Perform interprocedural may points-to analysis using the modified call strings method. Does the method terminate? Explain why.