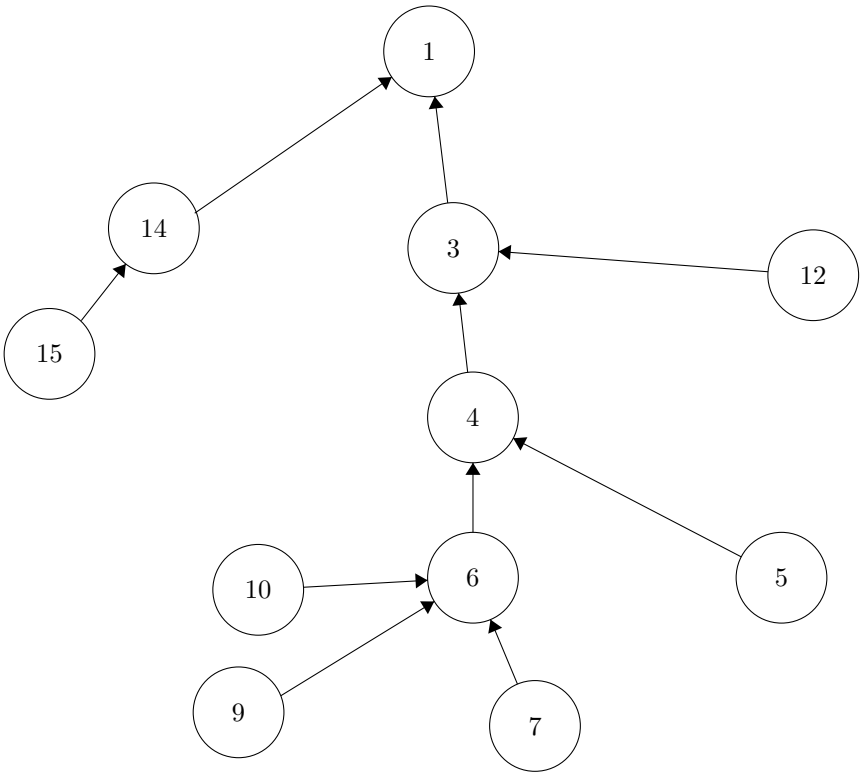
1 Dynamic Control Dependence

Dynamic control dependence graph:



trace	control depend.	depend. detected
1	$(1_1, EXIT)$	
3	$(1_1, EXIT)(3_1, 14)$	$3_1 \rightarrow 1_1$
4	$(1_1, EXIT)(3_1, 14)(4_1, 14)$	$4_1 \rightarrow 3_1$
6	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)$	$6_1 \rightarrow 4_1$
7	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)$	$7_1 \rightarrow 6_1$
12	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)$	$12_1 \rightarrow 6_1$
3	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)$	$3_1 \rightarrow 4_1$
4	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)$	$4_2 \rightarrow 3_2$
6	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)$	$6_2 \rightarrow 3_2$
9	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(6_2, 3)$	$9_1 \rightarrow 4_2$
10	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(6_2, 3)$	$10_1 \rightarrow 6_2$
3	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(3_3, 14)$	$10_1 \rightarrow 6_2$
4	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(3_3, 14)(4_3, 14)$	$4_3 \rightarrow 3_3$
6	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(3_3, 14)(4_3, 14)(6_3, 3)$	$6_3 \rightarrow 6_3$
9	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(3_3, 14)(4_3, 14)(6_3, 3)$	$9_2 \rightarrow 6_3$
10	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(3_3, 14)(4_3, 14)(6_3, 3)$	$10_2 \rightarrow 6_3$
3	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(3_3, 14)(4_3, 14)(3_4, 14)$	$3_4 \rightarrow 4_3$
4	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(3_3, 14)(4_3, 14)(3_4, 14)(4_4, 14)$	$4_4 \rightarrow 3_4$
5	$(1_1, EXIT)(3_1, 14)(4_1, 14)(6_1, 3)(3_2, 14)(4_2, 14)(3_3, 14)(4_3, 14)(3_4, 14)(4_4, 14)$	$5_1 \rightarrow 4_4$
14	$(1_1, EXIT)(14, EXIT)$	$14 \rightarrow 1$
15	$(1_1, EXIT)(14, EXIT)(15, EXIT)$	$15 \rightarrow 14$
17		

2 Dynamic Data Dependence

This is an open ended question, but at a minimium the student must show how to handle the statements in the example program.

statement	actions
read(B, n)	id_map[id] = input; for (i = 0 toi row n) Pr(B+i)=id++
x (op) y	Pr(&x) = Pr(&y)∪ stack.top().third
A[i] = B[j]	$\Pr(A + i) = \Pr(B + j)$
L: if(y) or while(y)	stack.push(L, ipdom(L0), Pr(&y)∪ stack.cup().third)
L: an ipdom	while(stack.top().second == L) pop()

trace	Pr	stack
5. i = j = 0;	Pr(&i)=Pr(&j)={}	
6. read(B, 10)	Pr(B) = 0 ... Pr(B+3) = 3	
7. F= &foo();	Pr(&F) = {}	
8. while(j ¡ 10)		[8,16,{}]
9. if (B[j] == 'b')		[8,16,{}][9,16,{0}]
11. j++	Pr(&j) = 0	[8,16,{}][9,16,{0}]
12. if (j > 0)		[8,16,{}][9,16,{0}][12,14{0,1}]
13. i++		[8,16,{}][9,16,{0}][12,14{0,1}]
14. (*F)()		[8,16,{}][9,16,{0}]
...		
(eventually the loop breaks)		
16. A[i] = B[j]	Pr(A+3) = Pr(B+3)=3	...
17. (*F)()	WARNING! Pr(F+2) = 3	

The attacker can control the address A+3, which allows him to change one

byte in the function address, potentially performing a control flow hijack attack.

3 Forward Computation of Dynamic Slices

trace	x	y	power	z	CDS
3. input(x, y)	3	3			
4. if (y > 0))	3	3			{4, 8, {4,8,{3}}
7. power = -y	3	3	3,6,7		{4, 8, {6,8{3}}}
8. z = -1	3	3	3	8	
9. while(power != 0)	3	3	3	8	{9, 13, {3,6,7,9}}
10. z = z * x	3	3	3,9,7,10	3,6,7,8,9,10	{9, 13, {3,6,7,9}}
11. power = power - 1	3	3	3,6,7,9,10,11	3,7,8,9,10	{9,13,{3,6,79}}
9. while (power != 0)	3	3	3,6,7,9,10,11	3,7,8,9,10	{9, 13,{3,6,7,9,11}}
10. z = z * x	3	3	3,6,7,9,10	3,6,7,8,9,10,11	{9, 13, {3,6,7,9,11}}
(loop repeats)					
13. if (y < 0)	3	3,6,7,9,10	3,6,7,8,9,10,11	8	{13,15,{3,13}}
15. output(z)	3	3	3,6,7,9,10	3,6,7,8,9,10,11	

slice(z) = {3,6,7,8,9,10,11}

4 Static Analysis

Abstract domain and transfer function:

Let S(x) denote the sign of the variable in our abstraction. The domain is $\{+, -, 0, \top, \bot\}$ with $\top$ meaning undefined, $\bot$ meaning uncertain, and the other symbols being obvious.

The transfer function is:

$x = c$:

$$S(X) = \begin{cases} +, & c > 0 \\ 0, & c == 0 \\ -, & c < 0 \end{cases}$$

$x = y$:

$S(X) = S(Y)$

$x = y + z$:

$$S(x) = \begin{cases} +, & (S(y) = S(z) = +) \vee (S(y) = 0 \wedge S(z) = +) \vee (S(y) = + \wedge S(z) = 0) \\ 0, & S(y) = S(z) = 0 \\ -, & (S(y) = S(z) = -) \vee (S(y) = 0 \wedge S(z) = -) \vee (S(y) = - \wedge S(z) = 0) \\ \bot, & otherwise \end{cases}$$

$x = y - z$:

$$S(x) = \begin{cases} +, & (S(y) = 0 \wedge S(z) = -) \vee (S(y) = + \wedge (S(z) = 0 \vee (S(z) = -)) \\ 0, & S(y) = S(z) = 0 \\ -, & (S(y) = 0 \wedge S(z) = -) \vee (S(y) = - \wedge (S(z) = 0 \vee (S(z) = +)) \\ \bot, & otherwise \end{cases}$$

Argue that it terminates:

Assume a fixed point algorithm where everything begins $\top$. Because of the way I defined the transfer function the only possible transitions are:

$$\top \to + \tag{1}$$
$$\top \to - \tag{2}$$
$$\top \to 0 \tag{3}$$
$$\top \to \bot \tag{4}$$
$$+ \to \bot \tag{5}$$
$$- \to \bot \tag{6}$$
$$0 \to \bot \tag{7}$$
$$\tag{8}$$

Since each variable can only make a finite number of transitions (never going back) each iteration makes some progress towards the fixed point and, by induction, the analysis terminates. If for example a transition $+ \to +$ were possible the analysis would not be guaranteed to terminate.

Extend your analysis to compute aggregate values directly:

Basically you need to meet all the variables in the instruction.

$x = y + z + a + b... :$

$$S(x) = \begin{cases} 0, & \forall\{y, z, a, b...\} : SIGN(i) = 0 \\ +, & \forall\{y, z, a, b...\} : SIGN(i) = + \vee SIGN(i) = 0 \\ -, & \forall\{y, z, a, b...\} : SIGN(i) = - \vee SIGN(i) = 0 \\ \bot & otherwise \end{cases}$$

$x = y - z - a - b... :$

$$S(x) = \begin{cases} 0, & \forall\{y, z, a, b...\} : SIGN(i) = 0 \\ +, & (\forall\{y, z, a, b...\} : SIGN(i) = + \vee SIGN(i) = 0) \wedge \exists i | SIGN(i) = + \\ -, & ((\forall\{y, z, a, b...\} : SIGN(i) = - \vee SIGN(i) = 0) \wedge \exists i | SIGN(i) = -) \\ -, & \text{for exactly one i: } SIGN(i) = 1 \wedge \text{the others: } SIGN(i) = 0 \\ \bot & otherwise \end{cases}$$

5 Delta Debugging

input	partition	result	action
.....ging-is-fun	Δ_1	F	
delta-debug.....	Δ_2	T	reduce test set to Δ_2
delta-.....	Δ_1	F	
.....debug	Δ_2	F	increase granularity
del.....	Δ_1	F	
...ta-.....	Δ_2	F	
.....deb..	Δ_3	F	
.....ug	Δ_4	F	try complements
...ta-debug	∇_1	F	
del...debug	∇_1	T	reduce to ∇_2 with n = 3
del.....	Δ_1	F	
...debug	Δ_2	F	increase granularity n = 6
de.....	Δ_1	F	
..l.....	Δ_2	F	
...d....	Δ_3	F	
....e...	Δ_4	F	
.....b..	Δ_5	F	
.....u.	Δ_6	F	try complements
..ldebug	∇_1	F	
de.debug	∇_2	T	reduce to ∇_2 with n = 5
de.....	Δ_1	F	
..de...	Δ_2	F	
....b..	Δ_3	F	
.....u.	Δ_4	F	
.....g	Δ_5	F	try complements
..debug	∇_1	F	
de..bug	∇_2	F	
dede.ug	∇_3	T	reduce to ∇_3 with n = 4
de....	Δ_1	F	
..de..	Δ_2	F	
....u.	Δ_3	F	
.....g	Δ_4	F	try complements
..deug	∇_1	F	
de..ug	∇_2	F	
dede.g	∇_3	T	reduce to ∇_3 with n = 3
de...	Δ_1	F	
..de.	Δ_2	F	
....g	Δ_3	F	try complements
..deg	∇_1	F	
de..g	∇_2	F	
dede.	∇_3	T	reduce to ∇_3 with n = 2
de..	Δ_1	F	
..de	Δ_2	F	try complements
..de	∇_2	F	
de..	∇_1	F	increase granularity n = 3
de..	Δ_1	F	
..d.	Δ_2	F	
...e	Δ_3	F	try complements
..de	∇_1	F	
de.e	∇_2	T	reduce to ∇_2 n = 2
de.	$\Delta_1 = \nabla_2$	F	
..e	$\Delta_2 = \nabla_1$	F	increase granularity
d..	Δ_1	F	
.e.	Δ_2	F	
..e	Δ_3	F	try complements
.ee	∇_1	reduce to ∇_1 with n = 2	

(I had to split it into 2 tables for latex)

e.	$\Delta_1 = \nabla_2$	F	
.e	$\Delta_2 = \nabla_1$	F	cannot reduce further. done.

Therefore ee is the 1-minimal input.