CS510 Midterm 2017 Spring

name: _____

1. What are the differences between flow-sensitive and flow-insensitive analysis (10p). Please give an example for each kind (10p).

   Flow-sensitive analysis considers control flow. The analysis results are indexed by program point. The result for a specific point depends on those preceding the point along control flow. Dependence analysis is flow-sensitive

   Flow insensitive anlaysis does not consider control low. As such, the analysis results are not indexed by the program points. For example, type inference is flow insensitive. The type of a variable has nothing to do with its control flow

2. What are the differences between context-sensitive and context-insensitive analysis (10p). Please use an example to illustrate the limitations of context-insensitive analysis (10p).

   Context-sensitive analysis considers function invocation contexts. It is equivalent to inlining the code of a callee to the callers. Hence, the code body of the callee get duplicated in the different call sites, each having its own analysis state.

   Context-insensitive analysis does not consider function invocation contexts. Hence, there is only one sanalysis state for a function.

   Consider the following example,

```
foo (int * p, int * q)
{
   *p=10;
   t=*q+1;
}
main (){
  foo (&x, &y);
  foo (&x, &x);
}
```

   Context sensitive analysis can tell that p and q are alias in the context of the second call of foo, where as they are not alias in the context of the first call. The results are indexed by contexts.

3. Assume an intra-procedural alias analysis as follows.

$$\frac{x = \&y}{x \to y} \qquad \frac{x = {}^{L}malloc(y)}{x \to L}$$

$$\frac{x = y \quad y \to t}{x \to t} \qquad \frac{(*x) = y \quad y \to t \quad x \to p}{p \to t}$$

$$\frac{x = y + z \quad y \to t}{x \to t} \qquad \frac{x = (*y) \quad y \to t \quad t \to q}{x \to q}$$

Please apply the analysis to the following program and present the results (15p). What are the pros and cons of the algorithm (5p)

```
1:  x=y
2:  p=malloc (10)
3:  (*p)=&x
3:  x=&z
4:  q=malloc(10)
5:  r=q+1
6:  (*r)= &p
10: b=(*q);
12: a=(*b)
```

p->2
2->x
x->z, y->z
q->4
r->4
4->p
b->p
a->2

Don't miss y->z because the analysis is flow-insensitive

Pro: simple and scalable
Con: flow-insenstive and hence imprecise in many places

4. Program dependences and slicing

```
1  input(x,y);
2  s=0;
3  p=1;
4  t=x%2;
5  if (t==0) {
6     while (x>=0) {
7       s=s+x;
8         p=p*(x-1);
9       if (p>y) break;
10      x-=2;
11    }
12 }
13 Output(s);
14 Output(p);
```

Construct the CFG of the above program (10p). Construct dependence graphs of the program (10p). Please separate data and control dependence graphs for readability.

Most of you did fine on this, so no need for solution. The only point you need to pay attention is that line 6 is control dependent on line 9

5. Please define a dynamic analysis to detect memory leak. Please use the following language.

Program P ::= s

Statement s ::= s1; s2 | x=^{L}y | x =^{L} y op z | x=^{L} c |
x=^{L} &y | (*x)=^{L} y | x=^{L} *y | x=^{L} malloc(y) | free^{L} (x)
if (x^{L}) s1 else s2 |
while (x^{L}) s

Operation op :: = + | - | * | / | > | < | …
Value c :: = 0 | 1 | 2 … | true | false
Address a :: = 0 | 1 | 2…
Variable x, x1, x2, x3
Label L, L1, L2,…

Assume pointer arithmetic is not supported. Define your analysis domain (5p). Write down the semantics (15p). Define a static analysis to do this (extra 10p).

Note: you can assume infinite memory so that you never need to reuse de-allocated space. You don't have to write down the rules for all statements if the rules are not related to the analysis.

Memory Leak Detection

- $< s, \delta, \gamma, H, A, F > \rightarrow < s', \delta', \gamma', H', A', F' >$
  - γ : heap top
  - H: Address-> Address X Int (mapping from variable to the heap region that it points to)
  - A: P(Address), the set of allocated addresses
  - F: P(Address), the set of freed addresses

Semantics Rules

$$\frac{\delta' = \delta[\alpha[x] \mapsto c]}{< x = c; s, \delta, \gamma, H, A, F > \rightarrow < s, \delta', \gamma, H, A, F >}$$

$$\frac{\delta' = \delta\left[\alpha[x] \mapsto \delta[\alpha[y]]\right] \quad H' = H[\alpha[x] \mapsto H[\alpha[y]]]}{< x = y; s, \delta, \gamma, H, A, F > \rightarrow < s, \delta', \gamma, H', A, F >}$$

$$\frac{a = \delta[\alpha[y]] \quad b = \delta[\alpha[z]] \quad c = a + b \quad H[\alpha[z]] = < a, i > \quad \delta' = \delta[\alpha[x] \mapsto c] \quad H' = H[\alpha[x] \mapsto < a, i >]}{< x = y + z; s, \delta, \gamma, H, A, F > \rightarrow < s, \delta', \gamma, H', A, F >}$$

Semantics Rules

$$\frac{a = \delta[\alpha[y]] \quad b = \delta[\alpha[z]] \quad c = a + b \quad H[\alpha[y]] = < a, i > \quad \delta' = \delta[\alpha[x] \mapsto c] \quad H' = H[\alpha[x] \mapsto < a, i >]}{x = y + z; s, \delta, \gamma, H, A, F > \rightarrow < s, \delta', \gamma, H', A, F >} \text{ Pnt-Add-y}$$

$$\frac{a = \delta[\alpha[y]] \quad b = \delta[\alpha[z]] \quad c = a + b \quad H[\alpha[y]] = undef \quad \delta' = \delta[\alpha[x] \mapsto c] \quad H[\alpha[z]] = undef}{< x = y + z; s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma, H, A, F >} \text{ NonPnt-Add}$$

Semantics Rules

$$\frac{\delta' = \delta[\alpha[x] \mapsto \gamma] \quad \gamma' = \gamma + \delta[\alpha[y]] \quad H' = H[\delta[\alpha[x]] \mapsto < \gamma, \delta[\alpha[y]] >] \quad A = A \cup \{r\}}{< x = malloc(y); s, \delta, \gamma, H, A, F > \rightarrow < s, \delta', \gamma', H', A', F >}$$

$$\frac{H[\delta[\alpha[x]]] = < a, i > \quad F' = F \cup \{a\}}{< free(x); s, \delta, \gamma, H, A, F > \rightarrow < s, \delta', \gamma, H', A, F' >}$$

Memory Leak

- At the end, you compare if A=F. If not, there is leak
- The static version of the analysis is just to use a work list to remember the states of the paths you haven't explored. And at the end of each path, check if A=F. And of course, you need to use the label L of the malloc to denote the memory allocated at L.