CS 6340 Practice Final Exam - Fall 2016

- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straight-forward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary.

- Partial solutions will be graded for partial credit.

- In accordance with both the letter and spirit of the Georgia Tech Honor Code, you will neither give nor receive assistance on this exam.

Question	Lesson	Weight	Score
1	6	10	
2	7	8	
3	8	8	
4	9	10	
5	10	8	
6	11	6	
Total:		50	

Question 1. [10 points] Pointer Analysis. Consider a flow-insensitive, context-insensitive, but field-sensitive pointer analysis of the following program:

```
class Node {
    int v;
    Node left, right;
    Node(int v) { this.v = v; }
}


static void main() {
    Node root = new Node(5);   // h1
    for (int i = 0; i < 10; i++)
        add(root, i);
}
```

```
boolean add(Node this, int q) {
    int p = this.v;
    if (q < p) {
        Node n = this.left;
        if (n == null) {
            Node l = new Node(q);  // h2
            this.left = l;
            return true;
        }
        return add(n, q);
    }
    if (q > p) {
        Node m = this.right;
        if (m == null) {
            Node r = new Node(q);  // h3
            this.right = r;
            return true;
        }
        return add(m, q);
    }
    return false;
}
```

(a) [4 points] A flow-insensitive pointer analysis analyzes an (unordered) set of simple statements in the given program. Write the statements contained in this set for the above program (in the `main` and `add` functions together). Recall that this set includes only 4 kinds of statements: allocation (v = new h, where h is the commented label), copy (v1 = v2), field read (v1 = v2.f), and field write (v1.f = v2). The allocation statements are listed below for your convenience. You will lose points for listing statements not analyzed by this analysis.

Relevant allocation statements: `root = new h1,` `l = new h2,` `r = new h3`
Relevant copy statements: ____
Relevant field-read statements: ____
Relevant field-write statements: ____

Answer:
Relevant copy statements: `this = root,` `this = n,` `this = m`
Relevant field-read statements: `n = this.left,` `m = this.right`
Relevant field-write statements: `this.left = l,` `this.right = r`

(b) [6 points] For each pair of expressions below, write all correct choices among 1-3 below.

1. They do NOT point to the same object in any (concrete) run of the program.
2. The above pointer analysis proves using allocation-site based heap abstraction that they CANNOT point to the same object in any run of the program.
3. The above pointer analysis proves using the type based heap abstraction that they CANNOT point to the same object in any run of the program.

```
A. root, root.right                           1   2   3
B. root.left, root.right                      1   2   3
C. root.left.right, root.right.left           1   2   3
D. root.left.left, root.right.right           1   2   3
```

Answer:
A. 1, 2
B. 1, 2
C. 1, 2
D. 1, 2

Question 2. [8 points] Constraint-Based Analysis. Let edge be the relation of all edges in a directed graph. Define relation path as follows:

```
path(n1, n2) :- edge(n1, n2).
path(n1, n2) :- path(n1, n3), path(n3, n2).
```

(a) [3 points] Does the following Datalog program compute the same relation P as the above path relation, on the same input edge relation?

```
P(n1, n2) :- edge(n1, n2).
P(n1, n2) :- edge(n1, n3), P(n3, n2).
```

Choose one of the following:
- Always
- Sometimes but not always
- Never

Answer: Always

(b) [2 points] Complete the Datalog program that computes the relation R to contain each node that has at least one incoming edge. (In your answer write the statement that would fill in blank.)

R(n) :- ___________ .

Answer: R(n) :- edge(_, n).

(c) [3 points] Complete the Datalog program that computes relation S to contain each pair of nodes n1, n2 that occur in the same cycle. (In your answer write the statements that would fill in the blanks.)

S(n1, n2) :- ___________ , ___________ .

Answer: S(n1, n2) :- path(n1, n2), path(n2, n1).

Question 3. [8 points] Type Systems. Develop a type system for tracking the parity (even vs. odd) of integer expressions in the following language:

```
      (expression)  e  ::=  c | x | e1 + e2 | e1 * e2
(integer constant)  c
(integer variable)  x
          (parity)  p  ::=  even | odd | top
```

Assume that environment A is a map from each variable to even, odd, or top. Add the eight missing rules, each of which has judgments of the form A |- e : p, meaning that "under environment A, expression e has parity p."

Rules for 'c' and 'x':

```
[c is an odd constant]       [c is an even constant]
-----------------------------  -----------------------------  -----------------------------
      A |- c : odd                  A |- c : even                  A |- x : A(x)
```

Rules for' e1 + e2':

In your answer, write the four rules for e1 + e2 in the following form:

```
 [   ]    [   ]
--------------------
      [   ]
```

where each [   ] is filled in with an appropriate statement.

Rules for 'e1 * e2':

In your answer, write the four rules for e1 * e2 in the following form:

```
 [   ]    [   ]
--------------------
      [   ]
```

where each [   ] is filled in with an appropriate statement.

Answer:

The four rules for e1 + e2 are as follows:

```
A |- e1 : even       A |- e2 : even
-------------------------------------------
        A |- e1 + e2 : even

A |- e1 : even       A |- e2 : odd
------------------------------------------
        A |- e1 + e2 : odd

A |- e1 : odd        A |- e2 : even
-------------------------------------------
        A |- e1 + e2 : odd
```

```
A |- e1 : odd       A |- e2 : odd
-------------------------------------------
          A |- e1 + e2 : even
```

The four rules for e1 * e2 are as follows:

```
A |- e1 : even      A |- e2 : even
----------------------------------------------
          A |- e1 * e2 : even
A |- e1 : even       A |- e2 : odd
--------------------------------------------
          A |- e1 * e2 : even

A |- e1 : odd        A |- e2 : even
---------------------------------------------
         A |- e1 * e2 : even

A |- e1 : odd        A |- e2 : odd
--------------------------------------------
        A |- e1 * e2 : odd
```

Question 4. [10 points] Statistical Debugging. Consider the following code fragment:

```
/* The sets of runs reaching A and B are disjoint. */
if P then
    …     /* A: 30 runs reach this line, 10 fail on this line. */
else
    …     /* B: 50 runs reach this line, 10 fail on this line. */
/* All 60 runs reaching this line succeed. */
```

(a) [3 points] What is Increase(P)?

Answer: Increase(P) = 10/30 - 20/80 = 0.083

(b) [3 points] What is Increase(¬P)?

Answer: Increase(!P) = 10/50 - 20/80 = -0.05

(c) [4 points] Consider this alternative elimination strategy to the one presented in the lecture: Once predicate P is chosen in an iteration, only failing runs where P is true are discarded. For the above example, what is Increase(¬P) under this strategy in the iteration immediately after P is chosen?

Answer: Increase(!P) = 10/50 - 10/70 = 0.057

Question 5. [8 points] Delta Debugging. A program takes any input that is a sub-string of abcd. That is, the following are possible inputs: abcd, abd, bc, d, the empty string, etc. The program fails on inputs abcd and bc, and passes on all other inputs. This is the key. It only crashes when abcd or bc is passed in

Suppose we are given the failing input abcd and seek to minimize it. Define c_F = { δa,δb,δc,δd } where the elementary changes are defined as follows:

δa = keep character a in its position (1st)
δb = keep character b in its position (2nd)
δc = keep character c in its position (3rd)
δd = keep character d in its position (4th)

(a) [6 points] Fill in the steps of the delta debugging algorithm, stopping when appropriate (the number of rows provided might be more than the number of times the algorithm iterates; you will lose points if you unnecessarily fill extra rows).

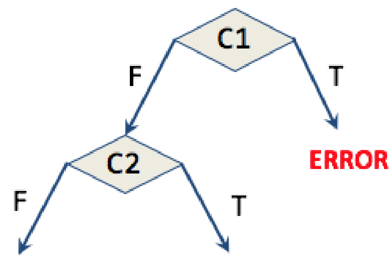
Iteration	n	Δ	$\Delta_1, \ldots, \Delta_n, \nabla_1, \ldots, \nabla_n$
1	2	abcd	ab, cd
2			
3			
4			

Answer: Iteration 2: ***n*** **= 4 Δ = abcd Δ1, …, Δn, ∇1, …,∇n = a, b, c, d, abc, bcd, acd, abd**

(b) [2 points] Write down the reduced failing input that the algorithm finally outputs.

Answer: abcd

Question 6. [6 points] Dynamic Symbolic Execution. Consider the following computation tree:



(a) [4 points] List each of these 8 constraints that DSE might possibly solve in exploring the tree:

C1	(not C1)	(not C1) and C2	C1 and C2
C2	(not C2)	C1 and (not C2)	(not C1) and (not C2)

Answer: C1, (not C1), (not C1) and C2, (not C1) and (not C2)

(b) [2 points] In the worst case, in how iterations (i.e. test cases) will DSE discover an input that reaches ERROR? Give as tight a bound as possible. Count the original random test input as well as the final test input on which the ERROR is reached.

Answer: 3

THE END