CPSC 410 Sample Questions

Note:

The following questions are indicative of the types of questions on the final exam. However, they do not account for all the material that will be covered on the exam. In addition to practicing these exercises, it is also important to go over the lecture notes, assigned reading, and practice exercises which we have done in class.

For example, where questions here are asked about Security Principle X; it is quite possible the final exam will ask questions about Security Principle Y.

For example, where questions here are asked about using a forwards analysis; it is quite possible the final exam will ask questions about backwards analysis.

etc... etc...

The final exam will contain 80% newer material and 20% older material. Only newer material is touched on in these exercises.

1.

a. A software analysis tool which uses an under-approximation could possibly declare that every single object dereference in a program might exhibit a null-pointer exception.

True, in the case the every object dereference could indeed have a null-pointer exception.

b. A reaching definition determines where a variable must have been defined.

False, it deteremines where it may have been defined.

c. Based on the principle of complete mediation all source code should be analyzed or inspected before a product is released.

False, complete mediation addresses authorization of resource usage not source code.

d. An input format vulnerability can leave a system open to unauthorized information corruption but not to unauthorized information disclosure.

False, for example, an SQL Injection could allow access to some private database tables.

e. Java programmers don't need to worry about buffer overflow attacks in their software.

True, Java VM prevents array index out of bounds errors.

2. Short answer (two to three sentences)

a. Based on the principle of Psychological Acceptability, why is possible that sometimes adding more security checks in software can end up reducing the overall security of a system?

Adding more security checks may require more input from the user to configure for their system. This may become frustrating for the user, so they might looks for ways to subvert the security checks altogether.

b. Explain how a resource depletion attack can cause a denial-of-service.

An attack might cause a Web site to exhaust its CPU or Memory servicing requests from an attacker. Then the Web site would not have CPU or Memory resources available to service legitimate users.

c. Explain what kind of problem can be avoided by following the principle of Least Privilege.

In some cases a trusted user or trusted program could cause damage to a system, e.g. a trusted user could make a mistake or a trusted program could have an error or virus. In these cases the amount of damage that is caused would be limited if the system follows the principle of least privilege.

3. Convert the following Java code into equivalent code that used the PreparedStatement API. Assume the **name** is a string type in the database and **id** is an integer type. Information on usage is available here: http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

```
Connection conn = createConnection();

String name = request.getParameter("name");

String id = request.getParameter("id");

Statement s = conn.createStatement() ;

ResultSet rs =
s.executeQuery("SELECT * FROM Cust WHERE name ='"+ name +"' " AND id=" + id);
```

```
Connection conn = createConnection();

String name = request.getParameter("name");

String id = request.getParameter("id");

PreparedStatement s =
conn.prepareStatement("SELECT * FROM Cust WHERE name = ? AND id= ?");

s.setString(1, name);

s.setInt(2, Integer.parseInt(id));

ResultSet rs = s.executeQuery();
```

4. Consider the code sample below.

```
1.    void zaxis(int x, int y) {
2.          int z=0;
3.          while(z < 10) {
4.                x = z * x;
5.                z = z + 1;
6.          }
7.          if(y != 0) {
8.                z = z + y;
9.                print(z);
10.         }
11.         else {
12.               z = z- y;
13.               print(z);
14.         }
15.   }
```

a. A local variable or parameter is considered to be live at a program statement if its current value MAY be used at that statement or during the remaining execution of some function. For each statement in the function zaxis, list whether the variables x, y, and z are live.

	x	y	z
1.	yes	yes	no
2.	y	y	n
3.	y	y	y
4.	y	y	y
5.	y	y	y
7.	n	y	y
8.	n	y	y
9.	n	n	y
12.	n	y	y
13.	n	n	y

b. A local variable or parameter is called very-busy at a program statement if its current value MUST be used during the remaining execution of some function before it is re-assigned. For each statement in the function zaxis, list whether the variables x, y, and z are very-busy.

	x	y	z
1.	yes	yes	no
2.	y	y	n
3.	y	y	y
4.	y	y	y
5.	n	y	y
7.	n	y	y
8.	n	y	y
9.	n	n	y
12.	n	y	y
13.	n	n	y

5. In this class, we had never discussed how to draw a CFG for IF statements that include the boolean operators. Convert the following Java function to one with equivalent behavior for which you can draw a CFG. Hint: Recall that Java (also C, C++, etc...) uses short-circuit logic for evaluating boolean operators.

In other words, at `(x && y)`, y will not be evaluated if x is false;
at `(x || y)`, y will not be evaluated if x is true.

```
static int check_parameters(SMat A, long dimensions, long iterations) {
   int error_index = 0;

   if (iterations <= 0 && iterations > A.cols || iterations > A.rows)
       error_index = 5;
   else if (dimensions <= 0 || dimensions > iterations) error_index = 6;
   if (0 != error_index)
       svd_error("svdLAS2 parameter error");
   return(error_index);
}
```

```
static int check_parameters(SMat A, long dimensions, long iterations) {
        int error_index = 0;

        if (iterations <= 0) {
            if(iterations > A.cols) {
                  error_index = 5;
            } else if(iterations > A.rows) {
                  error_index = 5;
            }
        } else if(iterations > A.rows) {
            error_index = 5;
        }
        else if (dimensions <= 0) error_index = 6;
        else if (dimensions > iterations) error_index = 6;

        if (0 != error_index)
            svd_error("svdLAS2 parameter error");
        return(error_index);
}
```

6. Consider the code for the following program.

```
public static void main() {
      String a = foo("x", null);
      String b = foo(null, a);
      System.out.println(b.toLowerCase());
}

static String foo(String x, String y) {
      Object z = "default";
      if(x == null)
            if(y == null)
                  z = y;
      return z;
}
```

a. Draw the control-flow graph using the Method Summary approach and solve for the NullPointerAnalysis.

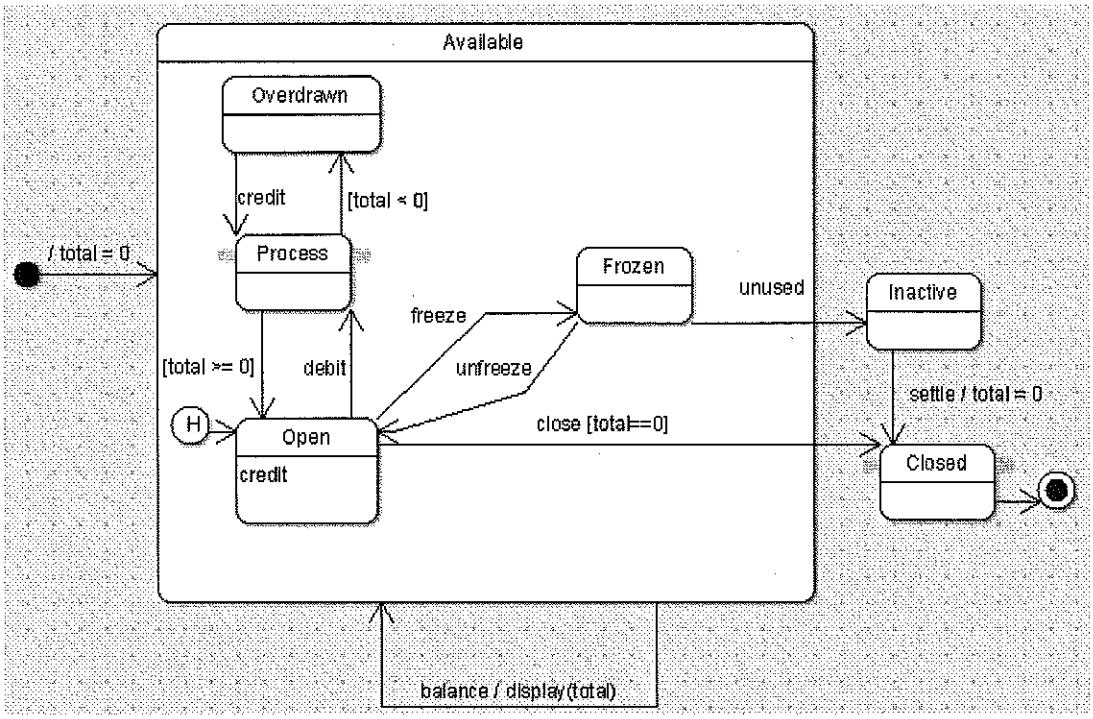
See Appendix

b. Draw the CFG using the Method Inlining approach and solve for the NullPointerAnalysis.

See Appendix

7. CTL

Consider the Account statechart as discussed in class, and answer the following (it is possible that the answer to some questions is "none"):



1. *For which states, s, is it true that EX(s)?*
   i.e. By considering which states as the "root" or "current" state is the expression satisfied?

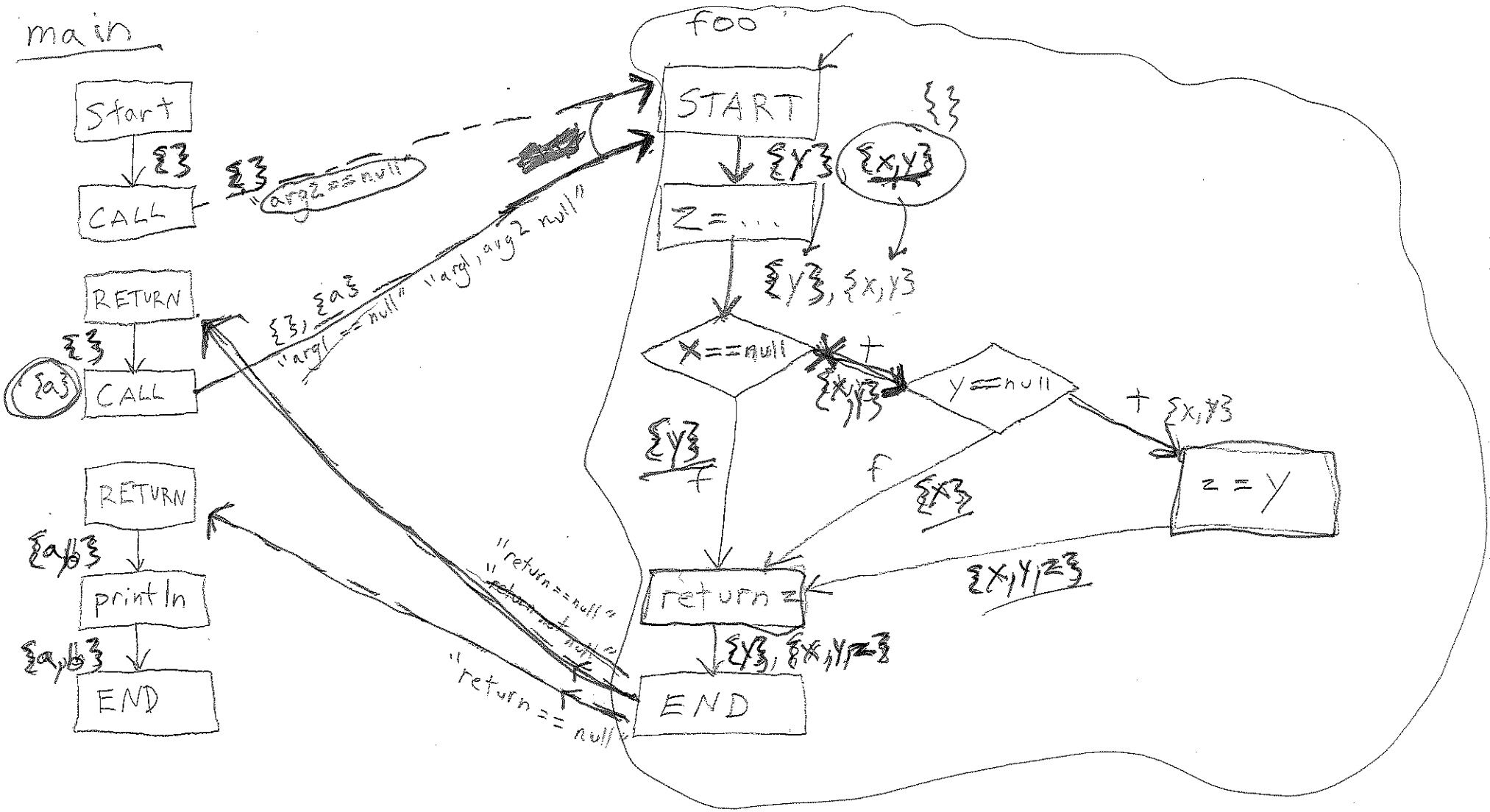
   Overdrawn, Process, Open, Frozen

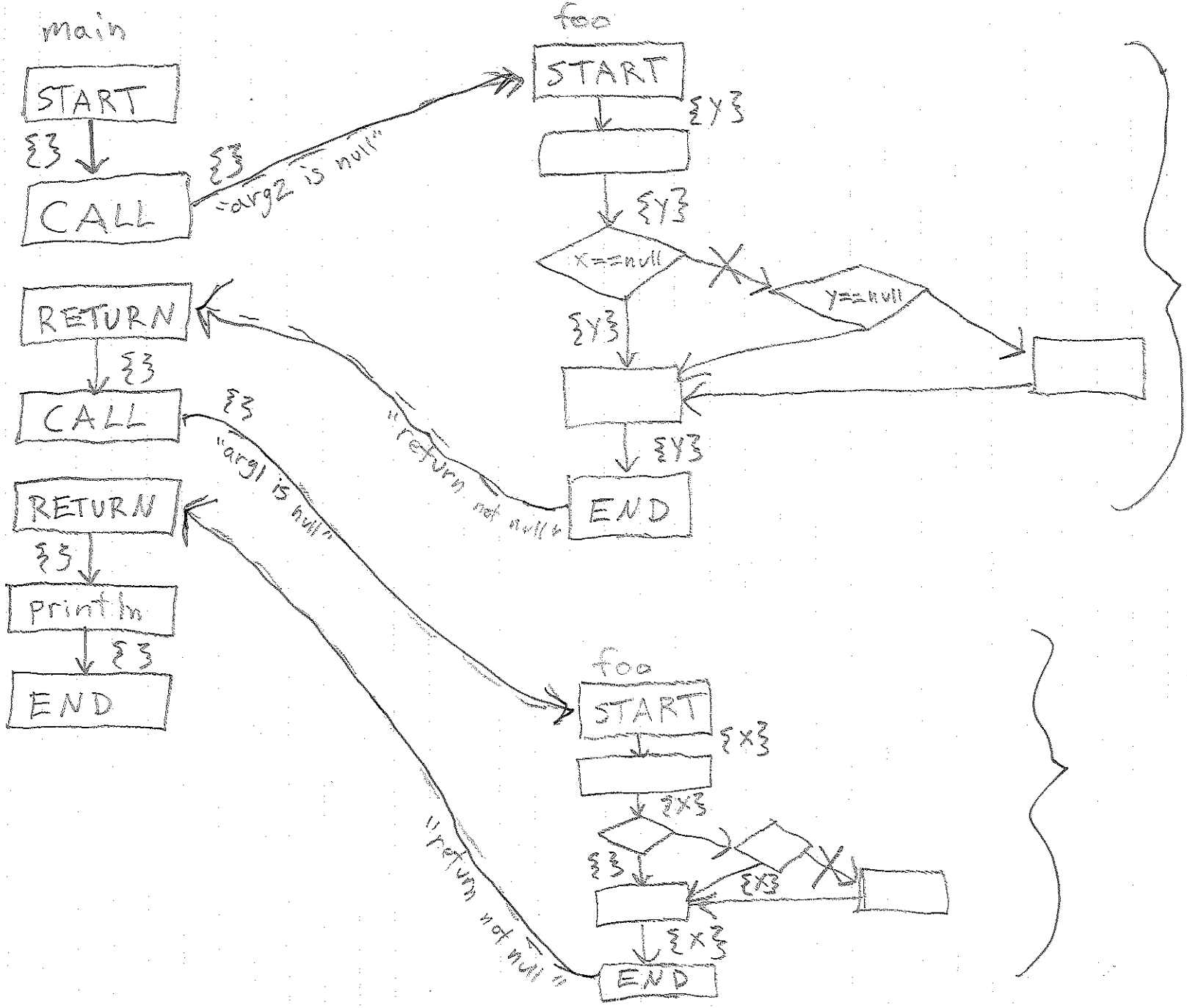
2. *For which states, is it true that* A(EG(total == 0) U Inactive)?

   Inactive

3. *For which states, s, is it true that,* EX(total > 0) and EX(EX(EX(total < 0)))

   Overdrawn, Process, Open, Frozen


main
Start
CALL
RETURN
CALL
RETURN
println
END
"arg2 == null"
"arg1 == null" "arg1, arg2 null"
"return == null"
foo
START
z = ...
x == null
y == null
z = y
return z
END


main
START
{}
CALL
{}
"arg2 is null"
foo
START
{x}
{y}
x==null
y==null
{y}
{y}
END
"return not null"
RETURN
{}
CALL
{}
"arg1 is null"
RETURN
{}
println
{}
END
foo
START
{x}
{x}
{}
{x}
{x}
END
"return not null"