Setup:

Git Repository link: https://github.com/CSCI599/DR

Java version: 7
External libraries: BCEL, Cobertura (along with any other dependency they might have).

Running:

The Java file Runner.java is the starting point of the analysis.

It expects 3 arguments:

1. Path to .class file
2. Path to generate the dotty file for the CFG.
3. Line number to run

The “Bookstore" application was used as a sample application for this project.
Understanding the output:

The output contains:

1) Location of dotty file.
2) Conditions that the line number depends on
3) Some detailed information about the conditions like source line number, variable and its expected value etc.

Approach:

1) Prepare CFG and reachability information
2) Find the conditions (IF statements in byte code) on which a particular node depends on. This is done by determining control dependency for each node (as described in the Jeanne Ferrante et al Algorithm):

3) The next step is to determine the variables on which a given condition depends on.
This is a complex step in terms of determining the value that a condition expects to be true.
This is because there can be different types of IF comparisons involving different types of data types.
I concentrated on basic integer and string comparisons.
From the byte code, I found the instructions that were loaded before the IF statement.
These instructions load the variables and constants required for the comparison.
From the LocalVariableTable I determine the name and value of these variables.

4) calculate reaching definition: Verify that the variable is obtained from "outside" and is not modified within the code.
If it is modified, mark the variable.
For each node in the graph, we compute the reaching definitions. This iterative algorithm is used to compute reaching definition:

// Boundary condition

out[Entry] = ∅

// Initialization for iterative algorithm For each basic block B other than Entry
For each basic block B other than Entry {
	out[B] = ∅
	gen[B] = {d} // gen of B is definition(s) generated in block B (could be ∅ if B does not contain defintion
	kill[B] = set of all other defs in the rest of program to the variable(s) defined in B
}

// iterate
While (Changes to any out[] occur) {
	For each basic block B other than Entry {
		in[B] = ∩(out[p]), for all predecessors p of B
		out[B]=gen[B] ∪(in[B]-kill[B])
	}
}

5) Print output: Print the line numbers, conditions, variables (and their expected value if possible) that the given node depends on.
If the variable was "Marked" in the reaching definition as modified in the code, display a message saying that the variable was modified in the code (with the line number where it was modified).

Evaluation:

I had used the Bookstore application for evaluation purposes (specifically, the Registration_jsp.java file)

If the input to the application contains line number “493“ as the line to run, the output is as follows:

Parsing classes/Registration_jsp.class.
Dotty File generated at: classes/
Line 493 depends on : 3 conditions

Instruction: 314: if_icmpeq[159](3) -> aload_0(at source line number: 491)
Must evaluate to : false
For instruction to be TRUE:
Variable:
(VALUE = 1)
=
iAction (int)
---
iAction is not an external variable. It is initialized/modified at line: 457
iAction is not an external variable. It is initialized/modified at line: 459
iAction is not an external variable. It is initialized/modified at line: 460
iAction is not an external variable. It is initialized/modified at line: 461

Instruction: 320: if_icmpne[160](3) -> ldc 86(at source line number: 491)
Must evaluate to : true
For instruction to be TRUE:

Variable:
(VALUE = 2)
!=
iAction (int)
---
iAction is not an external variable. It is initialized/modified at line: 457
iAction is not an external variable. It is initialized/modified at line: 459
iAction is not an external variable. It is initialized/modified at line: 460
iAction is not an external variable. It is initialized/modified at line: 461

Instruction: 329: ifeq[153](3) -> aload_0(at source line number: 492)
Must evaluate to : true
For instruction to be TRUE:
Variable:
fldmember_login (java.lang.Object)
=
EMPTY
---
member_login must have the value: EMPTY
fldmember_login depends on external variable member_login
fldmember_login is not an external variable. It is initialized/modified at line: 480

member_login must have the value Generating new test case
title : Sample
Generated new test case Tests/Sample_2014-05-10 18:02:52

The output contains the conditions on which the given line depends upon, as well as some variable information.
The main goal here was to determine client side actions which affect execution of a certain line. Hence, those variables which were declared locally within the application were not analyzed in detail. Instead, the analysis gives a message saying that the variable is not external.

Variable:
(VALUE = 1)
=
iAction (int)

This means that the variable iAction should have the value 1.

To determine external variables, I searched for “getParam” calls. The variables which were external (and unmodified) were further analyzed. For eg.,
fldmember_login (java.lang.Object)
=
EMPTY
---
member_login must have the value: EMPTY
fldmember_login depends on external variable member_login

This means that fldmember_login should be “Empty” (or “”).

This information was used to prepare new test cases from existing one. However, this functionality needs a lot of improvement.
A Selenium test case is parsed and searched for the required external variable. Once found, the desired value is “plugged” in, and the new test case is created.

Further areas of improvement:

1) The Bookstore application did not have a lot of Tableswitch cases. The analysis covers some portions of Tableswitch cases, but may not function properly on complex applications.

2) Parsing test cases and generating new ones. Currently, the name and path to the Selenium test case is hard-coded.

```
(tcParser.parseTestCase("Tests/Sample", "name=" + var_name, var_val);)
```

This can later be modified to be accepted as a command line argument, once the desired functionality is working properly.