COM S 413/513: Homework 8 [Project]
Building an Automatic Debugger

October 24, 2023

Learning Objectives:

In this homework, you will

1. strengthen the understandings of delta-debugging
2. gain experiences of implementing an automatic program analysis tool
3. understand the challenges of implementing such tools
4. gain experiences of reproducing work in the research paper

Instructions:

1. Total points: 31 pt
2. Early Deadline: Nov 01 (Wed) 11:59PM
3. Deadline: Nov 03 (Fri) 11:59PM
4. This homework is accomplished by the team of 2 students. Individual submissions are strongly discouraged.
5. Only one team-member need to make a submission on Canvas. Each submission must have the team-members names.
6. How to submit:
   - Submit 2 files, a PDF and a Zip. Don't bundle the PDF within the Zip file.
   - PDF should contain the table and screenshots.
   - Zip should contain all the code and results and should have a README to explain which files/-folders are associated with each question and how to compile and test your implementation.

1 Description

In this homework, we are going to implement an automatic debugging tool based on the delta-debugging technique. Given a buggy version and a correct version (an old release and a new release) with a list of changes in between, delta-debugging systematically searches for the changes to identify in which change(s), the bug is introduced. Please see the following guidelines to proceed your project:

1. Read the paper: "Yesterday, my program worked. Today, it does not. Why?"

2. Implement the basic delta-debugging technique specified under Algorithm 1 in the paper. You can use C/C++, Java or Python whatever languages you feel comfortable and useful.

3. Run your tool against the test case given below.

   (a) Initial Version:

```
1  public class file1v1 {
2
3    public int fun(int a, int b, String type) {
4      int x = 0;
5      if (type.equals("summation")) {
6        return a + b;
7      }
8
9      int y;
10     if (type.equals("minus")) {
11
12       return a - b;
13     }
14
15     if (type.equals("modulous")) {
16       a--;
17       a++;
18       return a % b;
19     }
20
21     int bcbc = 1;
22
23     if (type.equals("subsquare")) {
24       return (a * a - b * b);
25     }
26     int v = 0;
27
28     return 0;
29   }
30
31   public static void main(String[] args) {
32     new file1v1().fun(Integer.parseInt(args[0]), Integer.parseInt(args[1]),
33         args[2]);
34   }
35 }
```

   (b) Second Version:

```
1  public class file1v1 {
2
3    public int fun(int a, int b, String type) {
4      if (type.equals("summation")) {
5        return a + b;
6      }
7      if (type.equals("multiplication")) {
8        return a * b;
9      }
10
11     int ax = 0;
12     if (type.equals("minus")) {
13       a++;
14       a--;
15       return a - b;
16     }
17     if (type.equals("modulous")) {
18       return a % b;
19     }
20     int by = 0;
```

```
21      if (type.equals("addsquare")) {
22        return (a * a + b * b);
23      }
24      int bcbc = 1;
25
26      if (type.equals("division")) {
27        return a / b;
28      }
29
30      if (type.equals("subsquare")) {
31        return (a * a - b * b);
32      }
33
34      if (type.equals("addsubsquare")) {
35        return (((a + b) * (a * b)) - ((a - b) * (a - b)));
36      }
37
38      return 1;
39    }
40
41    public static void main(String[] args) {
42      new file1v1().fun(Integer.parseInt(args[0]), Integer.parseInt(args[1]),
43          args[2]);
44    }
45  }
```

(c) Changes generated using `diff -U 0 <file1> <file2>`:

```
1  --- test_case/file1v1.java      2023-10-15 23:18:47.816021302 -0500
2  +++ test_case/file1v2.java      2023-10-15 23:16:10.144623636 -0500
3  @@ -6 +5,0 @@
4  -    int x = 0;
5  @@ -9,0 +9,3 @@
6  +    if (type.equals("multiplication")) {
7  +      return a * b;
8  +    }
9  @@ -11 +13 @@
10 -    int y;
11 +    int ax = 0;
12 @@ -13 +15,2 @@
13 -
14 +      a++;
15 +      a--;
16 @@ -16 +18,0 @@
17 -
18 @@ -18,2 +19,0 @@
19 -      a--;
20 -      a++;
21 @@ -22 +22,4 @@
22 -
23 +    int by = 0;
24 +    if (type.equals("addsquare")) {
25 +      return (a * a + b * b);
26 +    }
27 @@ -24,0 +28,4 @@
28 +    if (type.equals("division")) {
29 +      return a / b;
30 +    }
31 +
32 @@ -28 +34,0 @@
33 -    int v = 0;
34 @@ -30 +36,5 @@
35 -    return 0;
36 +    if (type.equals("addsubsquare")) {
37 +      return (((a + b) * (a * b)) - ((a - b) * (a - b)));
```

```
38 +     }
39 +
40 +     return 1;
```

(d) Calling the second version with input `5 0 division` will throw a divide by zero bug. Since this function was not present in initial version, it is a bug introduced in the second version.

(e) An example output for your tool can be as shown below. Note that, the output was generated using different change set than given here.

```
Delta-debugging Project
@@ -6 +5,0 @@
@@ -9,0 +9,3 @@
@@ -11 +13 @@
@@ -13 +15,2 @@
@@ -16 +18,0 @@
@@ -18,2 +19,0 @@
@@ -22 +22,4 @@
@@ -28 +30,0 @@
@@ -30 +32,9 @@
# of Total Change sets is = 9
Step 1:  c_1:  1 2 3 4 PASS
Step 2:  c_2:  5 6 7 8 9 FAIL
Step 3:  c_3:  5 6 PASS
Step 4:  c_4:  7 8 9 FAIL
Step 5:  c_5:  7 PASS
Step 6:  c_6:  8 9 FAIL
Step 7:  c_7:  8 PASS
Step 8:  c_8:  9 FAIL
Changes where bugs occurred: [9]
```

2 Deliverables

1. (4 pt) Please submit the screenshot to show your delta-debugging tool successfully locates the fault, including any intermediate steps in the PDF submission.

2. (4 pt) Please submit a summarization table like Table 1 in the paper via the PDF submission.

3. (20 pt) Please submit the source code as well as the binary of your delta-debugging tool in the zip submission.

4. (3 pt) Please provide details in a README file to explain how TA should build and run your tool. If TA fails to build your tool due to unclear instructions, your points will be reduced. If your tool can work with the given test case, you get full points. Otherwise, TA will inspect the source code to give you partial credits.