Spectrum Based Fault Localization

Anita Devi

Department of Computer Science and Applications, Kurukshetra University (Haryana)

Abstract—Spectrum Based Fault Localization (SBFL) is the technique used for fault localization. Fault localization means locating the errors or faults. To locate faults developers must identify statements involved in failures and select suspicious statements that might contain faults. In this research paper various spectra like node spectra, edge spectra, edge-pair spectra and block spectra have been discussed. Regression testing is done to check whether the changed program is working properly or not and also to check that unchanged code has not been affected due to modifications. This paper includes node spectra, edge spectra, edge-pair spectra and block spectra which traverse the nodes, edges, edge-pairs and blocks of the original and modified programs. These spectra will help developers to know whether the changes in the behavior of the new program is due to regression fault or due to change in code.

Keywords—regression testing, program spectra, SBFL, fault localization.

I. INTRODUCTION

Regression testing is sometimes referred as “Program Revalidation”[10]. The process includes generation of a test suite, modification of the source code, executing the test cases using modified version, searching for deviation in the output. Regression testing is needed when a subsystem is modified to generate a new version of an application. When one or more components of an application are modified, the entire application must also be subjected to regression testing. Software maintenance involves modifying programs as a result of errors or alterations in the user requirements. When a new version P’ of a program P is produced, developers must check whether the changes that they have introduced in P’ behave as expected and do not affect the unchanged code. Developers rerun either all of the test cases or a subset of the existing test cases. If one or more test cases that were executed successfully on P cause an unexpected failure when executed on P’, the developer would know that the changes introduced regression faults.

SBFL ranks program entities according to their risks of being faulty. SBFL first collects the information from various spectra and the associated result in terms of fail and pass criteria of each individual test case. Program spectra can be node spectra, edge spectra, edge-pair spectra, prime path spectra, predicate path spectra, complete path spectra, branch spectra, block spectra etc and any type of run time information (e.g. the binary coverage status, the execution frequency etc). Using information from these spectra, SBFL gives ranking list [4].

II. LITERATURE REVIEW

In this section, we briefly review related work on software diagnosis. Program spectra were first introduced in the context of fixing year 2000 problems in software [Reps et al 1997]. Various classes of program spectra such as branch spectra, path spectra, execution trace spectra and output spectra were introduced. In our study, we concentrate on node spectra, edge spectra, edge-pair spectra and block spectra. In the research paper “Using program spectra to improve effectiveness of regression testing” proposed by Christopher Hayden[1], different program spectra have been used to improve the effectiveness of regression testing. They told that program spectra can improve regression testing by reducing the number of test cases that need to be run by revealing faults that do not propagate to output and by localizing the portion of code causing deviations. Tom, Rui and Arjan proposed “Zoltar-Spectrum based fault localization tool” which states that automated fault localization of software faults can improve the efficiency of debugging process. They presented a tool set for automatic fault localization. Zoltar adopts a spectrum based fault localization
technique[8]. “On the accuracy of SBFL” proposed by Rui, Peter and Arjan Gemund[6] investigated the diagnostic accuracy as a function of several parameters (such as quality and quantity of the program spectra collected during the execution of the system). By varying the quality and quantity of the observations on which the fault localization is based, they established the result in much wider context. “A model for SBFL” proposed by Lee, Hua and Kotagiri presented an improved approach to assist diagnosis of failure in software by ranking program statements or blocks according to how likely they are buggy. They present a very simple single bug program. The model also helps to identify groups of metrics which are equivalent for ranking[9]. “Regression Testing in research and practice” by Xuan tried to do the survey of current research on regression testing and current practice in industry and also tried to find out whether there are gaps between them[10]. “An empirical investigation of relationship between spectra and regression faults” proposed by Mary, Harrold, Gergg, Kent and Rui states that many software maintenance and testing tests involve comparing the behaviour of program spectra[3]. They examined the relationship between difference in program spectra and the exposure of regression faults existing in modified version of a program that were not present prior to modification.

### III. PROBLEM STATEMENT

Regression testing has a few shortcomings:
1. Expensive to rerun all the entire test suite.
2. Assume deviation propagate to output.
3. Difficult to find the cause of a deviation.

Program Spectra can help to overcome these limitations. A program spectrum is collection of data that provides a specific view of the dynamic behaviour of software. It contains information about the parts of the program that were executed during the execution of several test cases. The parts of a program can be individual statements, blocks, branches, nodes or edges etc. During execution of test cases, data is collected indicating the statements that are excuted. Each test case is classified as passed or failed. There are some problems in SBFL like dealing with multiple faults, efficiency of locating faults for large scaled programs, applying SBFL in the absence of test oracle and selecting the most effective risk evaluation formula[1].

To address the problem of multiple faults, clustering-based method and model-based method can be used. For applying SBFL in the absence of test oracle, metamorphic testing technique is used. To remove oracle problem, MT is used. MT[9] generates test cases by making references to “metamorphic relations” (MR). For program P, an MR is a property of its function f. The unique character of MR is that it involves multiple executions, for example if \( f(x)=e^x \), then the property \( e^a \cdot e^b = 1 \) is a typical MR[7].

Program Spectra captures the dynamic behaviour of the program statements. It indicates which program statements are responsible for observed failure, is the most expensive phase during software development. To improve the efficiency of debugging process, there should be automated diagnosis of software faults.

### IV. PROPOSED SOLUTION

The comparison of spectra of old and new version of program depends on the following assumptions: Node Spectra, Edge spectra, Edge-Pair Spectra and Block Spectra of both old and new version of program are compared and checked whether they are lexicographically equivalent or not [1], e.g locating the software components, which are responsible for observed failure, is the most expensive phase during software development.

To improve the efficiency of debugging process, there should be automated diagnosis of software faults.
For this purpose, there is a tool named ZOLTAR, which adopts spectrum based fault localization technique. Let P be the original program and P’ be the modified program[8]. These programs can be implemented in C, C++ and also in linux.

**Program P**

```c
0   Int i=2;
1   If(x<y)
   { 
2       While(i<4)
            { 
4           x=x*2;
5           i++;
          }
3   } Else if(x>y)
6   x=x-y;
7   Else
8   x=x+1;
9   return x;
```

**Program P’**

```c
0   Int i=2;
1   If(x<y)
   { 
2       While(i<4)
            { 
4           x=x*2;
5           i++;
          }
3   } Else if(x>y)
6   x=x-y;
7   Else
8   y=y+2;
9   x=x-2;
10  If(x<6){
11      x=x+3;
12      While(x%3==0){
13          x++;
14    }
15  }
```

P and P’ are the two programs, P is the original program. After making some changes in P, the resulted program is P’. Three different conditions are given in both of the programs. The changes are made in program P when x>y. In original program, eight statements are there and after making some changes, new program is of 14 statements.

![Fig 1: CFG of program P](image)

Fig 1 shows the control flow graph of program P. The graph is of eight nodes. This is a directed graph, where arrow shows the direction of flow of control. Fig 2 shows the control flow graph of modified program P’. The graph consists of 14 nodes. Each node represents the corresponding statement. This is also a directed graph. The change in control occurs when the condition x>y is executed. The different nodes are attached with each other through edges and edge pairs.
Table 1: Computed Program Spectra for program P

<table>
<thead>
<tr>
<th>Test_id</th>
<th>Input values</th>
<th>Node spectra</th>
<th>Edge spectra</th>
<th>Edge-pair spectra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test_1</td>
<td>{2,5}</td>
<td>{0,1,2,4,5,7}</td>
<td>{{0,1}, {1,2}, {2,4}, {4,5}, {5,2}, {2,4}, {4,5}, {5,7}}</td>
<td>{{0,1,2}, {1,2,4}, {2,4,5}, {4,5,2}, {5,2,4}, {2,4,5}, {4,5,7}}</td>
</tr>
<tr>
<td>Test_2</td>
<td>{5,3}</td>
<td>{0,1,3,6,7}</td>
<td>{{0,1}, {1,3}, {3,6}, {6,7}}</td>
<td>{{0,1,3}, {1,3,6}, {3,6,7}}</td>
</tr>
<tr>
<td>Test_3</td>
<td>{3,3}</td>
<td>{0,1,3,8,7}</td>
<td>{{0,1}, {1,3}, {3,8}, {8,7}}</td>
<td>{{0,1,3}, {1,3,8}, {3,8,7}}</td>
</tr>
<tr>
<td>Test_4</td>
<td>{9,2}</td>
<td>{0,1,3,5,7}</td>
<td>{{0,1}, {1,3}, {3,5}, {5,7}}</td>
<td>{{0,1,3}, {1,3,5}, {3,5,7}}</td>
</tr>
</tbody>
</table>

Fig 2: CFG of modified program P

Fig 3: Block diagram of program P

Fig 3 shows the block diagram of program P. A block that ends in a decision has two edges going out of it.
Table 2: Computed Program Spectra for Program P’

<table>
<thead>
<tr>
<th>Test_id</th>
<th>Input values</th>
<th>Node spectra</th>
<th>Edge spectra</th>
<th>Edge-pair spectra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test_1</td>
<td>{2, 5}</td>
<td>(0,1,2,4,5,2,4,5,7)</td>
<td>[[[{0,1}, (1,2), (2,4), (4,5), (5,2), (2,4), (4,5), (5,7)]]]</td>
<td>[{0,1,2}, (1,2,4), (2,4,5), (4,5,2), (5,2,4), (2,4,5), (4,5,7)]</td>
</tr>
<tr>
<td>Test_2</td>
<td>{5, 3}</td>
<td>(0,1,3,6,9,10,11,13,14,7)</td>
<td>[[[{0,1}, (1,3), (3,6), (6,9), (9,10), (10,11), (11,13), (13,14), (14,7)]]]</td>
<td>[{0,1,3}, (1,3,6), (3,6,9), (6,9,10), (9,10,11), (10,11,13,14), (11,13,14,7)]</td>
</tr>
<tr>
<td>Test_3</td>
<td>{3, 3}</td>
<td>(0,1,3,8,7)</td>
<td>[[[{0,1}, (1,3), (3,8), (8,7)]]]</td>
<td>[{0,1,3}, (1,3,8), (3,8,7)]</td>
</tr>
<tr>
<td>Test_4</td>
<td>{9, 2}</td>
<td>(0,1,3,6,9,10,12,7)</td>
<td>[[[{0,1}, (1,3), (3,6), (6,9), (9,10), (10,12), (12,7)]]]</td>
<td>[{0,1,3}, (1,3,6), (3,6,9), (6,9,10), (9,10,12), (10,12,7)]</td>
</tr>
</tbody>
</table>

Fig. 4: Block Diagram of Program P’

Fig. 4 shows the block diagram of program P’.

Table 3: Block Spectra of Program P

<table>
<thead>
<tr>
<th>Test_id</th>
<th>Block Spectra</th>
</tr>
</thead>
<tbody>
<tr>
<td>{3,1}</td>
<td>{B1,B2,B4,B5,B7}</td>
</tr>
<tr>
<td>{2,4}</td>
<td>{B1,B2,B3,B7}</td>
</tr>
<tr>
<td>{2,2}</td>
<td>{B1,B2,B4,B6,B7}</td>
</tr>
</tbody>
</table>

Table 4: Block Spectra of Program P’

<table>
<thead>
<tr>
<th>Test_id</th>
<th>Block Spectra</th>
</tr>
</thead>
<tbody>
<tr>
<td>{3,1}</td>
<td>{B1,B2,B4,B5,B6,B7,B10}</td>
</tr>
<tr>
<td>{2,4}</td>
<td>{B1,B2,B3,B10}</td>
</tr>
<tr>
<td>{2,2}</td>
<td>{B1,B2,B4,B10}</td>
</tr>
</tbody>
</table>

There are two programs P and P’. Fig 1 shows the control graph of original program P and Fig 2 shows the control graph of program P’. CFG of program P consist of eight nodes, but when some modifications are made to this program resulting in program P’, the CFG will be of 14 nodes. Table 1 defines the different spectra for program.
P and Table 2 shows different spectra for program P'. Different test cases are run to provide different spectra. For first and third test case spectra for both of the programs are same but different for second and fourth test case. In program P, three conditions are given (1) \( x < y \) (2) \( x > y \) (3) \( x == y \). In program P' the same conditions are given in the code, but some changes are made in program code P in condition (2) i.e. when \( x > y \) resulting in the program P'. For the test_id test_1 and test_3 node spectra, edge spectra, edge-pair spectra and block spectra are same for both of the programs P and P'. There is difference in node spectra, edge spectra, edge-pair spectra and block spectra of programs P and P' for the test_id test_2 and test_4 where x>y. Block spectra are also defined for both of the programs P and P'. A block is a section of code which is grouped together. Blocks consist of one or more declaration and statements for example in program P and P' block B3 consist of three statements, in program P' block B5 consists of two statements and block B7 consists of three statements. Table 3 shows the block spectra for program P and Table 4 shows the block spectra for program P'. For first test case, block spectra for both of the programs are different, but for second and third test case, block spectra are same for both of the programs.

Assume that Program Spectra of program P is \( Ps(P) \) and of modified program is \( Ps(P') \). \( Ps(P) \) and \( Ps(P') \) are considered equivalent if and only if sequences of conditional branches are identical as the program executes. If the \( Ps(P) \) and \( Ps(P') \) are not equal, it means there is behavioural difference between the two programs. The behavioural difference between the two programs P and P' are arranged for Spectrum Based Fault Localization. For program P' four spectra are computed i.e. Node Spectra(\( NS \)), Edge Spectra(\( ES \)), Edge-Pair Spectra(\( EPS \)) and Block Spectra(\( BS \)).

\[
P': \{NS, ES, EPS, BS\}
\]

To track the computed result of each spectra, we need to create a vector \( T_v \) and a matrix \( S_{mat} \). If the spectra is equivalent to the corresponding spectra of P for a given test case, then it indicates 1 otherwise it indicates 0.

\[
\begin{align*}
T_1: & (2,5) \\
T_2: & (5,3) \\
T_3: & (3,3) \\
T_4: & (9,2)
\end{align*}
\]

\[
\begin{array}{cccc}
NS & ES & EPS & BS \\
1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0
\end{array}
\]

If the corresponding value of NS, ES, EPS and BS is 1 then it represents that there is no behavioral difference between old and modified program. This changed code is given to the developer to check whether the change in behavior of new program is due to regression fault or due to change in code. If regression fault is there, then these faults are removed without execution of test cases and there is no need to write oracles or specification for them. If behavior of the program is changed due to changes in code then new test cases are generated[7].

V. CONCLUSIONS

Regression testing has some limitations like its expensive to rerun all the existing test cases, it assumes that deviation propagates to outputs and difficult to find the cause of failure. To overcome these problems, spectra are used to selectively sample the program state during test execution. In this paper node spectra, edge spectra, edge-pair spectra and block spectra are defined. The extra information recovered could help to address the problem. This is the idea behind program spectra. Program spectra can improve the regression testing by reducing the number of tests that need to be run, by revealing faults that do not propagate to output and by localizing the portion of the code causing deviation. SBFL approach is simple and effective. But some limitations are there like we need test oracles to apply SBFL to an application. Recently,
Metamorphic Testing has been proposed to remove the oracle problem. The behavioural difference between old and new version of program are recorded in form of vector $T_v$ and matrix $S_{mat}$ that can be used as essential information required for fault localization. In future, the approach of spectrum based fault localization is used with MT to remove the oracle problem.

References