CLPS-MFL: Using Concept Lattice of Program Spectrum for Effective Multi-Fault Localization*

Xiaobing Sun¹, Bixin Li², Wanzhi Wen²

¹School of Information Engineering, Yangzhou University, Yangzhou, China
²School of Computer Science and Engineering, Southeast University, Nanjing, China

Abstract

Fault localization (FL) is an important but challenging task during software testing. Among techniques studied in this field, using program spectrum as a bug indicator is a promising approach. However, its effectiveness may be affected when multiple simultaneous faults are present. To alleviate this limitation, we propose a novel approach, CLPS-MFL, which combines concept lattice with program spectrum to localize multiple faults. Our approach first uses formal concept analysis to transform the obtained program spectrum into a concept lattice. Then, it uses three strategies to identify the failure root causes based on the properties of the concept lattice. Our empirical studies on three subject programs validate the effectiveness of the CLPS-MFL approach for localizing multiple faults.

I. Introduction

An important step in software debugging is to localize the root causes of software failures [1], [2]. To achieve this goal, many fault localization techniques have been proposed in the last decade [2], [3]. Existing approaches may either rely on analyzing the dependence between program elements to track the cause of the failure [4], or rely on the coverage information of program test-runs to identify the likely position inducing the program failure [3]. However, most of current FL approaches are designed for single fault case, and their effectiveness will be reduced in the context of multi-fault environment. Specifically, most of existing FL approaches used the suspicious degree of program elements, i.e., the likely position inducing the program failure, to identify the failure roots [3]. And users almost need to check all the program elements to remove all the faults in the program, which is really a difficult, costly and labor-intensive activity. In this paper, we propose a novel FL approach, CLPS-MFL, for multi-fault localization. As program spectrum has good scalability and easiness for implementation, a number of researches on fault localization is based on this technique [5], called program spectrum based FL approaches. The CLPS-MFL proposed in this paper takes as inputs program spectrum, and uses formal concept analysis technique to transform program spectrum into a concept lattice, called concept lattice of program spectrum (CLPS). Based on the labeling approach and hierarchy properties of CLPS, three strategies are proposed to characterize the correlation between program elements and the test runs to indicate the correlation between a failure and the multiple faults. In addition, these three strategies can not only guide us to effectively localize the faults and can also help us effectively remove some program elements that are not faults, which effectively saves much effort for the whole FL process.

II. Preliminaries

Formal Concept Analysis (FCA) is a very useful clustering technique to detect meaningful groupings of data and properties within the data. The input of the FCA process is the formal context, which can be easily represented by a relation table, as shown in Table 1. Based on the formal context, some interesting clusters called formal concepts can be identified [6]. And all the formal concepts forms a partial order on the set of all formal concepts, called concept lattice. Taking Table 1 as the input, we use the concept lattice construction algorithm to generate the concept lattice [6], as shown in Fig. 1. In this concept lattice, each lattice node represents a formal concept, and we use the labeling approach to denote a lattice node. For example, the lattice node \( \text{co}3 \) is labeled by the extent \( \gamma(t1, t2) \) and the intent \( \mu(s11) \). It shows that: (1) all

*This work is supported partially by National Natural Science Foundation of China under Grant No. 61070133 and No. 61202006, and partially by the Jiangsu Province Natural Science Fund under Grant No. BK2010134
TABLE I. Formal context

<table>
<thead>
<tr>
<th>co9</th>
<th>co1</th>
<th>co2</th>
<th>co4</th>
<th>co3</th>
<th>co5</th>
<th>co6</th>
<th>co7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s1,s2,s3,s4)</td>
<td>(s1)</td>
<td>(s1, s12, s9)</td>
<td>(s8, s9, s7)</td>
<td>(s1)</td>
<td>(s12)</td>
<td>(s10, s14)</td>
<td>(s1)</td>
</tr>
<tr>
<td>E0</td>
<td>E1</td>
<td>E2</td>
<td>E3</td>
<td>E4</td>
<td>E5</td>
<td>E6</td>
<td>E7</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

![Graphical representation of the concept lattice]

As introduced above, the input of both concept lattice and program spectrum can be represented with a relation table, as shown in Table 1. Then through concept lattice construction algorithm [6], a concept lattice such as Fig. 1 can be generated. In this paper, such concept lattice is called Concept Lattice of Program Spectrum (CLPS). And we use the simple labeling approach to represent their extents and intents. Therefore, in the CLPS, the labeling of lattice nodes represent the characterization of the extent or intent, which can be used to identify the specific featured program elements of passed test runs or failed test runs. In addition, the CLPS presents a hierarchical classification of formal attributes, which can be used to define the suspiciousness of the program elements being faulty. And the CLPS-MFL approach is performed based on the labeling approach and the hierarchy property of the CLPS.

As Fig. 1 shows, some lattice nodes are labeled with both extent and intent, e.g., co3, while some others are labeled with only the extent or only the intent, e.g., co0. In addition, we need to differentiate the failed tests and passed tests in the lattice, and use these information to identify the probable failure-caused program elements and remove the failure-free program elements, respectively. Then, based on the labeling approach and hierarchy property of this concept lattice, three strategies are proposed to characterize the correlation between program elements and test runs. These strategies are as follows.

1) Including strategy. If a lattice node is labeled with both extent (tests) and intent (program elements), and the extent is only labeled with failed tests, it indicates that the failed tests are characterized by the program elements in the intent. The program elements labeled by this kind of lattice nodes are included with the highest priority in the failure cause set.

2) Filtering strategy. According to the similar mechanism of including strategy, if a lattice node is labeled with both extent (tests) and intent (program elements), and the extent of a lattice node is only labeled with passed tests, the program elements labeled by this lattice node are removed from the failure cause set.

3) Ranking strategy. For other lattice nodes in the CLPS, we rank the program elements labeling these lattice nodes based on a suspiciousness metric. The suspiciousness metric considers two aspects: (1) the further the distance between the program elements labeling upward reachable lattice nodes and the failed tests, the less likely the program elements to be the cause inducing the failure; (2) the more failed tests the lattice nodes including program elements upward reachable from, the more likely these program elements to be the faults. Based on these two considerations, we propose a new suspiciousness model to indicate the possibility of the program elements being cause of the failure. It is defined on the lattice node $k$ of CLPS:

$$\text{Suspiciousness}_k = \frac{\sum_{i=1}^{n} \frac{\text{Failed}_i}{\text{Distance}_i}}{\sum_{i=1}^{n} \frac{\text{Failed}_i}{\text{Distance}_i} + \sum_{j=1}^{m} \frac{\text{Passed}_j}{\text{Distance}_j}}$$

In this formula, $n$ and $m$ are the number of failed tests
int max=0, a, b, c, d;
read(a,b,c,d);
if (a>b)
  max=a;
else
  max=b;

max=a1//fault; it should be: max=c;
if (max>c)
  max=a1;
else
  max=b1;

max=a1//fault; it should be: max=c;
if (max>c)
  max=a1;
else
  max=b1;

s14 print(max);

Fig. 2. An example program

TABLE II. Program statements selected from three strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Lattice Node</th>
<th>Statement</th>
<th>Suspiciousness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Including</td>
<td>co1</td>
<td>s11</td>
<td></td>
</tr>
<tr>
<td>Filtering</td>
<td>co1</td>
<td>s10, s12, s9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>co2</td>
<td>s4, s5, s7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>co5</td>
<td>s13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>co6</td>
<td>s8</td>
<td></td>
</tr>
<tr>
<td>Ranking</td>
<td>co1</td>
<td>s1, s14, s2, s3</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>co6</td>
<td>s6</td>
<td>0.4</td>
</tr>
</tbody>
</table>

and the number of passed tests reachable from lattice node $k$ to these lattice nodes labeling these tests, respectively. $Failed_i$ and $Passed_j$ represent the $i$th failed test and the $j$th passed test, respectively. $Distance_i$ is the minimum distance between lattice node $k$ and the lattice node labeling a test$^1$. Finally, all the left program elements screened by the including strategy and filtering strategy are ranked according to the Suspiciousness values, and program elements with higher Suspiciousness values have higher probability of being faulty.

The CLPS-MFL approach to localize the multiple faults in the program is based on the strategies above. First, we use the filtering strategy to remove the program elements labeled by these lattice nodes as these program elements are seen as failure-free program elements. Then, we use the including strategy to select the program elements labeled by these lattice nodes. These program elements are included with the highest priority in the failure cause set, and should firstly be checked to see whether they are faulty. Finally, we use the ranking strategy to rank the rest program elements according to the Suspiciousness values. The ranked program elements are sequentially submitted for inspection.

We use the example program in Fig. 2 to clarify the CLPS-MFL process. In this program, there are two faults, i.e., $s6$ and $s11$. The test runs used to identify the faults are assumed to be in Table 1, in which we assume that there are three failed tests ($t1, t2, t3$) and five passed tests (the others). First we generate the CLPS as shown in Fig. 1. Then, we use three strategies to remove the failure-free program statements and identify the faulty program statements. The results selected by these three strategies are shown in Table 2. First, we use the filtering strategy to filter the program elements {$s10, s12, s9, s1, s6, s7, s13, s8$} labeling lattice nodes {co1, co2, co5, co6} from the failure cause set. There are in all 13 program statements in the program, and eight of them are removed from the failure cause set, which can really save developer’s or tester’s much effort. Then, we use the including and ranking strategy to identify the suspicious faulty program statements. On the one hand, we use the including strategy to inspect the program elements of co3, that is $s11$. And we find that $s11$ is one of the faults. On the other hand, we use the suspiciousness metric to compute the Suspiciousness values for the rest program elements in the CLPS, which are shown in the last row of Table 2. According to the results, we should first inspect $s6$, and find it is just the other fault. Then, we check the other statements, and they are all not faulty. Until here, the CLPS-MFL process is finished. Therefore, our CLPS-MFL approach is effective to localize multiple faults in the program.

IV. Empirical Study

A. Setup

As the CLPS-MFL approach is implemented with Java language in the Eclipse environment, we select three open source Java projects for our studies as shown in Table 3. The first two subject programs are selected from the SIR$^2$ (Software-artifact Infrastructure Repository) library, which is widely used for experimentation with testing and analysis techniques. The first subject program is Binary Search Tree$^3$ (BST), which is a data structure having low memory overhead and still allow fast searches. The second is Red Black Tree$^4$ (RBT), which is a self balancing tree whose search/insert/remove methods have a time complexity of $O(\log n)$. The final subject is Java Hierarchical Slicing and Applications (JHSA), which is a prototype tool used to construct hierarchical dependence graphs from package level to statement level, and with its application in various software maintenance activities [7]. And we select three versions of JHSA for empirical study.

$^1$The Distance range is starting from 1 to prevent the denominator from ever becoming 0, that is, the distance of the failed or passed test directly on the lattice node itself is 1.

---

2http://sir.unl.edu
3http://sir.unl.edu/content/bios/Binary-Search-Tree.php
4http://sir.unl.edu/content/bios/Red-Black-Tree.php
To perform our empirical study, we randomly seeded $k$ faults ($2 \leq k \leq 5$) into each program to create a set of multi-fault versions for each subject program. The test cases for these three programs are generated based on branch coverage criteria. The number of test cases for each subject program can be seen in the last column of Table 3. To validate the effectiveness of the CLPS-MFL approach, we use the wasted effort metric, which is referred from Abreu et al. [5]. It measures the percentage of excess work incurred when all faulty program elements are identified.

### B. Results

The lower the value of wasted effort is, the better the efficiency of the CLPS-MFL approach is. The wasted effort results of the CLPS-MFL for different number of faults are shown in Table 4. The results that the CLPS-MFL approach only needs a few extra effort to localize all the faults at any cases of the number of faults. For example, the maximal wasted effort is only 5\% for the BST subject when the number of the faults is four. And the wasted effort for all the other subjects with different number of faults is below 5\%. Therefore, for multiple faults in the program, our CLPS-MFL approach can effectively locate them.

## V. Related Work

A number of approaches have been studied to alleviate the difficulty of effectively localizing fault. Current studies in fault localization are either based on dependence analysis [4] or coverage analysis [3]. And most of these techniques were proposed for single fault localization. And there were also a few techniques focusing on handling multiple faults. Abreu et al. proposed a multiple-fault localization technique, BARINEL, which is based on the dynamic, spectrum-based approach from statistical fault localization methods, combined with a probabilistic reasoning approach from model-based diagnosis [5]. In addition, Cellier et al. proposed an interactive fault localization method based on two data mining techniques, i.e., formal concept analysis and association rules [8]. The proposed technique is similar to us, and uses the lattice to formalize the partial ordering and the dependencies between the sets of program elements that are most likely to lead to program execution failures. In this paper, we proposed a novel multi-fault FL technique, which combines concept lattice with program spectrum technique. It can not only effectively identify the multiple faults, at the same time, it can also effectively remove those that are not faults.

## VI. Conclusion and Future Work

In this paper, we proposed a novel multi-fault localization approach, CLPS-MFL, which is based on the properties of concept lattice combined with program spectrum. During the fault localization process, three strategies are used to effectively find the faults and remove the unrelated program elements. The initial empirical studies on three real-world programs showed the effectiveness of the CLPS-MFL approach. In the future, we mainly focus on a more widely and comprehensive empirical evaluation of the CLPS-MFL approach on more open and large-scale programs.

### References


