Spectrum-Based Fault Localization Aided by Program Slicing

Péter Soha
Software Engineering Department
University of Szeged
Szeged, Hungary
psoha@inf.u-szeged.hu

Árpád Beszédes
Software Engineering Department
University of Szeged
Szeged, Hungary
beszedes@inf.u-szeged.hu

***Abstract*—Determining the actual location of the fault during debugging can be aided by various automated techniques. Spectrum-Based Fault Localization (SBFL) is a popular automated fault localization method that is based on test execution statistics (also known as Statistical Fault Localization). The fundamental approach in SBFL is to observe pass/fail and coverage statistics of each test case and based on these assign suspiciousness ranks to program elements using one of the many possible heuristical formulae. A fundamentally different approach is to use the syntactic relationship of the program elements to follow the computation path from the observed behavior to the actual fault. For example, with (backward dynamic) Program Slicing (PS) one computes a program subset (including the fault itself) which might have contributed to the computation at the observed program location.**

Both areas have large literatures on their own, but attempts have been made to combine the two as well in the hope to find the fault more effectively. Previous research has shown that these hybrid solutions can combine the advantages of the base techniques, but most of these articles focus on a specific SBFL and PS algorithm, and a specific kind of combination is proposed. The goal of our work is to systematically investigate the possibilities of combining these two approaches for fault localization, and because we believe that the area has not yet been fully explored, we hope to be able to devise novel methods as well.

***Index Terms*—Program Slicing, Spectrum-based Fault Localization, debugging, combined FL**

I. Introduction and motivation

Different (semi-)automatic debugging and fault localization techniques have been proposed that aid the programmer in these activities. In our research, we focus on two important areas, namely *Spectrum-Based Fault Localization* (SBFL) [13, 17] and *Program Slicing* (PS) [7, 11, 22]. These two approaches are fundamentally different: while program slicing is based on the structure and the syntactic relationship between the elements, spectrum-based fault localization is a statistical approach which relies on test execution statistics.

SBFL uses the execution information of a program to locate the faulty code element. It takes the program spectra as input, which has two components, the code coverage matrix and the error vector. A code element is then assigned a suspiciousness value based on how many failing test cases are executing it compared to passing ones. Since there is no perfect formula to compute the suspiciousness ranking [23], the results are approximate and the efficiency is difficult to predict [2].

On the other hand, using PS, we can compute a subset of a program (the *program slice*) which might have an influence on a program point of interest (in our case, the point where the failure has been observed). There are numerous program slicing techniques available with many different applications. However, debugging is one the most important ones [19]. In particular, *dynamic slicing* (DS) [12] is often advised as a technique most suitable for debugging (and fault localization) because it computes the result for specific program executions, as opposed to *static slicing* (SS) where all possible executions are considered [19].

There could be different ways of combining the two approaches, such as using the program slice to further enhance the SBFL rank list, or using the highly suspicious elements to parameterize program slicing. Also, there is a vast set of concrete algorithms to choose from in both areas, so it is not evident how the combination would work the best.

Some examples of existing methods are the following. Reis *et al.* used SBFL and DS to increase the diagnostic accuracy by reducing the suspiciousness of those components that often failed but were not involved in passed tests and were not related to the fault according to DS [14]. Soremekun *et al.* examined a hybrid approach [16], and found that if the programmer first checks the most suspicious elements and then uses DS on them, the average lines of code that need to be examined can be reduced. Alves *et al.* combined DS and SBFL to reduce the cost of inspection without increasing the computational cost [3]. To achieve this, the authors used additional analysis information to remove non-faulty statements from the SBFL ranking. Shu *et al.* improved SBFL with "failed execution slices" [15], which were prioritized using the SBFL ranking.

The main motivation for the work presented in this paper is that we did not find a systematic overview of the possibilities to combine statistical fault localization with program slicing, and it is not clear which particular approach performs best in practical situations. Also, since the ultimate application is debugging and the target users are programmers working with their integrated development environments, it would be important to understand how these hybrid methods could be best used in practice.

II. Background

A. Fault Localization

In Spectrum-Based Fault Localization (SBFL), every entity of the investigated program is assigned a suspiciousness score that indicates the possibility of the failure at that entity. To calculate this score, these techniques use the *program spectrum* which contains information about the dynamic behaviour of the code [1]. This information can be represented using a binary matrix. Each column of it denotes a component of the program (blocks, functions, etc...), and every row is a test case. The value a_{ij} denotes if the i^{th} test case covered the j^{th} component (1) or not (0). In addition, a binary vector is used to hold the test outcomes, that is, $e_i = 1$ if the i^{th} test case failed and 0 otherwise.

Based on this information, we can calculate the suspiciousness score of each component with a frequency aggregation function $n_{pq}(j) = |\{i|M_{ij} = p \wedge e_i = q\}|$ where $n_{pq}(j)$ is the count of runs where the j^{th} component was active $(p = 1)$ or not $(p = 0)$ and the test failed $(q = 1)$ or passed $(q = 0)$ [1]. These four fundamental metrics are used in the SBFL formulae which typically increase the suspiciousness of code elements activated by many failed test cases compared to the others. Some of the most popular formulae include Tarantula, Ochiai, DStar, and many others [20, 21].

B. Program Slicing

Program slicing is a code analysis technique which aims to reduce the program into a set of instructions that contains those statements that may affect or may be affected by the values at some point referring as the *slicing criterion* [18]. The former is called the *backward slice* and the latter the *forward slice*. Generally, slicing algorithms can be classified by the information they use to find the faulty instructions. In *static slicing* only the statically available information will be used, while *dynamic slicing* algorithms compute those statements which affect the value of variables for the given program input or inputs (effectively, the test cases).

Dynamic slicing is consequently more precise, and is often proposed as the better alternative for debugging. In particular, with backward DS one computes a program subset which might have contributed to the computation at the observed program location in a debugging session, which typically manifested the failure. The computed backward slice should contain the faulty program element, so if it is small enough, it should aid the programmer in locating the fault.

Over the past decades, various slicing algorithms have been designed that effectively compute the slices, but most of them are based on some form of following the data and control dependences. Also, other techniques have been devised that combine static and dynamic analysis. For example, relevant slicing extends the dynamic slices with potentially dependent predicates and their static data dependences [9], while union slices are defined as a union of all possible dynamic slices regarding to a criterion [4].

III. Research Goals

The goal of our work is to systematically investigate the possibilities of combining SBFL and PS to enhance the effectiveness of automated fault localization, and in a broader context, debugging. Based on our preliminary review of the related literature, we believe that the area has not yet been fully explored, hence we hope to be able to devise novel methods as well. We will address the following main Research Questions.

RQ1: What is the state-of-the-art in the combined application of statistical (spectrum-based) fault localization and program slicing?
This will include a systematic literature review and surveying the existing techiques with the goal to identify current achievements, challenges and open questions, which will enable us to identify the most promising research directions.

RQ2: Which slicing approach can most successfully enhance SBFL methods?
Several program slicing techniques have been proposed since Weiser's original definition [19], and some excellent surveys have been published [5, 6, 8, 18]. Our goal is to examine the impact of the different techniques on SBFL.

RQ3: Which SBFL method is most suitable to be combined with program slicing?
As mentioned, SBFL uses various formulae to rank program elements according to their suspiciousness. Jie examined this question in his PhD Thesis [10] and gave a basic comparison of several formulae. We would like to continue this line of research and find out how the different techniques can be best combined with program slicing.

RQ4: How such a combination can be applied in actual debugging scenarios and debuggers? Real life usability is an important measure of an automated debugging and fault localization technique. Our goal is to integrate our approach into popular IDEs, such as IntelliJ or Netbeans, and to devise practical usage scenarios combined with existing debugging approaches. To test the scenarios, we plan to involve real developers to get relevant feedback on our approaches.

IV. Ongoing Research

Following the overall research goals, currently we are working on two topics:

Survey article: First, we will address RQ1, and compare the existing methods also in terms of accuracy, space and time requirements. We plan to publish the results in a survey paper.

Combined fault localization: In RQ2 and RQ3, we will more systematically investigate the possibilities of combining different SBFL and PS methods based on the results from RQ1. However, one direction is particularly promising, which we already started working on. Namely, we first compute a backward dynamic slice starting from the observed failure in a test case, and then use the obtained program slice to filter the results of the ranked program elements from SBFL. The expectation is that this way we would obtain the faulty element more effectively.

References

[1] Rui Abreu, Peter Zoeteweij, and Arjan J.C. Van Gemund. "On the accuracy of spectrum-based fault localization". In: *Proceedings - Testing: Academic and Industrial Conference Practice and Research Techniques, TAIC PART-Mutation 2007* (2007), pp. 89–98.

[2] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J.C. van Gemund. "A practical evaluation of spectrum-based fault localization". In: *Journal of Systems and Software* 82.11 (2009), pp. 1780–1792.

[3] Elton Alves, Milos Gligoric, Vilas Jagannath, and Marcelo d'Amorim. "Fault-localization using dynamic slicing and change impact analysis". In: *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. 2011, pp. 520–523.

[4] Árpád Beszédes, Csaba Faragó, Zsolt Mihály Szabó, János Csirik, and Tibor Gyimóthy. "Union slices for program maintenance". In: *Conference on Software Maintenance* May (2002), pp. 12–21.

[5] David W Binkley and Keith Brian Gallagher. *Program slicing*. 1996.

[6] David W Binkley and Mark Harman. "A survey of empirical results on program slicing." In: *Adv. Comput.* 62.105178 (2004), pp. 105–178.

[7] Andrea De Lucia. "Program slicing: Methods and applications". In: *Proceedings First IEEE International Workshop on Source Code Analysis and Manipulation*. IEEE. 2001, pp. 142–149.

[8] Keith Gallagher and David Binkley. "Program slicing". In: *2008 Frontiers of Software Maintenance*. IEEE. 2008, pp. 58–67.

[9] Tibor Gyimóthy, Árpád Beszédes, and István Forgács. "An efficient relevant slicing method for debugging". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1687 LNCS (1999), pp. 303–321.

[10] Lee Hua Jie. "Software Debugging Using Program Spectra". PhD thesis. 2012, pp. 244–258.

[11] Mariam Kamkar. "An overview and comparative classification of program slicing techniques". In: *Journal of Systems and Software* 31.3 (1995), pp. 197–214.

[12] Bogdan Korel and Janusz Laski. "Dynamic program slicing". In: *Information Processing Letters* 29.3 (1988), pp. 155–163.

[13] Priya Parmar and Miral Patel. "Software fault localization: A survey". In: *International Journal of Computer Applications* 154.9 (2016).

[14] Sofia Reis, Rui Abreu, and Marcelo D'Amorim. "Demystifying the combination of dynamic slicing and spectrum-based fault localization". In: *IJCAI International Joint Conference on Artificial Intelligence* 2019-Augus (2019), pp. 4760–4766.

[15] Ting Shu, Lei Wang, and Jinsong Xia. "Fault Localization Using a Failed Execution Slice". In: *2017 International Conference on Software Analysis, Testing and Evolution (SATE)*. 2017, pp. 37–44.

[16] Ezekiel O. Soremekun, Lukas Kirschner, Marcel Böhme, and Andreas Zeller. "Locating Faults with Program Slicing: An Empirical Analysis". In: *CoRR* abs/2101.03008 (2021).

[17] Higor Amario de Souza, Marcos Lordello Chaim, and Fabio Kon. "Spectrum-based Software Fault Localization: A Survey of Techniques, Advances, and Challenges". In: *CoRR* abs/1607.04347 (2016).

[18] Frank Tip. "A Survey of Program Slicing Techniques". In: *Journal of Programming Languages* 5399.3 (1995), pp. 1–65.

[19] Mark Weiser. "Program Slicing". In: *IEEE Transactions on Software Engineering* SE-10.4 (1984), pp. 352–357. ISSN: 00985589.

[20] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. "A survey on software fault localization". In: *IEEE Transactions on Software Engineering* 42.8 (2016), pp. 707–740.

[21] Xiaoyuan Xie, Tsong Yueh Chen, Fei-Ching Kuo, and Baowen Xu. "A Theoretical Analysis of the Risk Evaluation Formulas for Spectrum-based Fault Localization". In: *ACM Trans. Softw. Eng. Methodol.* 22.4 (Oct. 2013), 31:1–31:40.

[22] Baowen Xu, Ju Qian, Xiaofang Zhang, Zhongqiang Wu, and Lin Chen. "A brief survey of program slicing". In: *ACM SIGSOFT Software Engineering Notes* 30.2 (2005), pp. 1–36.

[23] Shin Yoo, Xiaoyuan Xie, Fei-Ching Kuo, Tsong Yueh Chen, and Mark Harman. "Human Competitiveness of Genetic Programming in Spectrum-Based Fault Localisation: Theoretical and Empirical Analysis". In: *ACM Trans. Softw. Eng. Methodol.* 26.1 (June 2017), 4:1–4:30.