Analyze Software Defects with Program Structure Dependency

Hui He[1], Lei Zhao[1], Qiao Li[1], Weizhe Zhang[1], Dongmin Gao[1],Hongli Zhang[1]

[1] School of Computer Science and Technology,
Harbin Institute of Technology,
Harbin, HL, China
{hehui,wzzhang}@hit.edu.cn

Abstract. Software vulnerabilities are may lead to huge loss. In this paper, we have proposed a new model, called Reverse Data Dependence Analysis Model. This integrates the current methods by analyzing the program structure. On this basis, we finally put forward a new automated fault localization method, which is named CPD.

Keywords: Software debugging, Fault localization, Data dependent, Coverage rate

1. Introduction

Software testing is an important means of defect detection. And it is also not replaced by defect detection methods within the rest stages in the software development process [1, 2]. This is because in the development phases before the software testing, the software itself is the only object of defect detection considering, including source code, requirements analysis and design documents. And only in the testing phase, the software has a run-time environment. At the same time, a lot of defects will be triggered only in a specific runtime environment and lead to software failure. Therefore, software analysis and software testing play a pivotal role to improve software quality and reliability.

Software fault localization and diagnosis is a critical step in the software testing. Only correctly diagnose software failure and locate code with defects in the software can be modified from the sources. Therefore, we have carried out a study of their advantages and defects, and put forward a new mode, named Reverse Data Dependence Analysis Model, which integrates the two methods by analyzing the program structure. On this basis, we finally put forward a new automated fault localization method, which is named CPD. This method is not only the automation lossless, but also changes the basic location unit into single sentence, which makes the location effect more accurate.

2. Reverse data dependence model based on analysis of program structure

In current fault localization methods, the most common methods are CBFL and program slicing. Here we will discuss these two methods, analyze defects of them in fault localization.

2.1 Defects of CBFL and program slicing fault localization

CBFL methods compute defect tendency value of code segment in isolation. The difference between the various methods is reflected on the computational formulas of debugging tendency value, but core ideas of them are the same [3, 4]. However, during program execution, the code segment is not isolated but interdependent. Especially after false data generated, the continuous code segments will be affected, resulting in a successive erroneous data. Therefore, it's likely to produce error in results while calculating the defect tendency in isolation. CBFL method is equivalent to calculating the failure to perform the code segment covered defect tendency. And this equivalent calculation method simplifies the process of program execution, thus affects the accuracy of location.

Program slicing technique is based on the different functions of codes. It picks up codes implementing same function from the source and assembles them into a new code segment. Program slicing technology plays an important role on understanding the program function, and there are some achievements applied to fault localization in the software testing phase. However, the growth of software scale has led to the expansion of the code space. It causes, even with the traditional program slicing technology, the problem of huge amount of sliced codes, and also makes the fault localization become difficult by using program slicing technology.

2.2 Reverse data dependence analysis model

Based on the above description of the problem, we propose the reversing data dependent model. Reversing data dependency analysis mainly simplifies the dependencies in program execution paths, aiming to reduce the amount of sliced codes.

Reverse data dependence analysis focuses on the same program execution path to analyze the data, extract and store the data dependencies on this path. Then traverse the stored data dependence reversed, based on a particular variable. Find code statements related to the specific variable, and achieve the purpose of streamlining program execution path, to facilitate the subsequent calculation of code coverage information. Figure 1 shows the execution flow of reverse data dependence analysis.

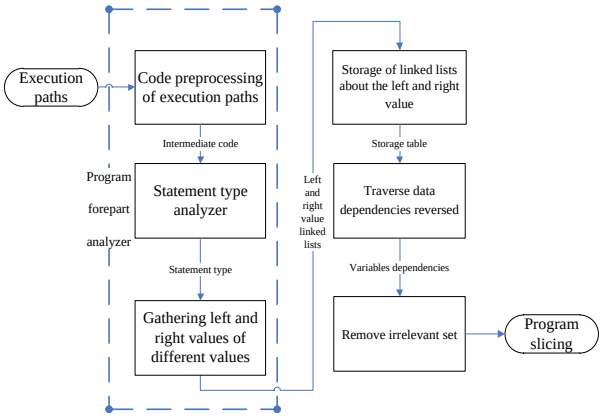

Fig. 1. Data dependence extraction function data flow diagram

Preprocessing: The code structure of C language is flexible, and different programming habits of everyone result in different code structures. Without preprocessing before analyzing code, a great deal of difficulty will appear next. So preprocessing converts various style codes into a unified specification code.

Statement type Analyzer: Because different types of statements need different extraction ideas, it is necessary to analyze the type of statement. Confirm types of statements analyzed by lexical analysis and syntax analysis, such as the basic assignment statement or control statement.

Gather the left value and the right value: Different extraction methods are designed for different specific statements, to collect left values and right values of different statements.

Storage of linked lists about the left and right value: Confirm the linked lists storage structure for left and right values, for the subsequent seeking variables by traversing as a matter of convenience. Good storage structure will improve the efficiency of searching.

Traverse data dependencies reversed: Taking the reverse traversal, traverse the data dependencies table from the end.

Remove irrelevant set: Remove irrelevant statement set with specified variables, and the remaining relevant statement set shall be the final program slicing.

3. CPD fault localization model

Considering the defects and advantages of both CBFL and program slicing fault localization, and combining with the theoretical basis of the analysis of program structure, we propose a fault localization solution CPD (Coverage and Program Data slicing), based on program execution path data dependency analysis.

N test cases of a complete program correspond to N different execution paths. Combining all test cases with the control flow analysis, a number of different

execution paths can be got. These execution paths contain different code statements, and the data stream is passed between these code statements. That is, when a variable is defined or assigned, its data dependencies are passed on as the program's execution path.

Our method analyzes the data stream of each execution path of the program synthetically. Based on an output variable with basic expectation, dependencies analysis will be made between reverse data variables respectively. Then confirm data dependencies of statements and irrelevant statement set for each path. Irrelevant statement set represents all statements in this collection are not related to other statements for this execution path, and they are irrelevant and can be ignored. We can delete the irrelevant statement set of each path, incorporate new execution paths without irrelevant sets into a complete data flow diagram of execution paths, and get a new program slicing of small code amount. And then analyze the data flow diagram to full data dependencies, according to coverage based fault localization. Count failure and successful execution frequency of each node, and calculate the defect tendency values. Sort the values, and ultimately locate the defect in a statement exactly in the program.

In this way, the granularity of location is the basic code statement. Compared to only control flow based fault localization, positioning granularity is reduced to a code statement instead of the basic code block. Under the premise of no loss the degree of automation of location, it improves the accuracy of location.

Combining data flow analysis with coverage based fault localization, it will improve the accuracy of fault localization and reduce the granularity of fault localization on data variables level. And it’s more convenient to debug for programmers.

The basic idea of the study is shown in Figure 2 as follows:

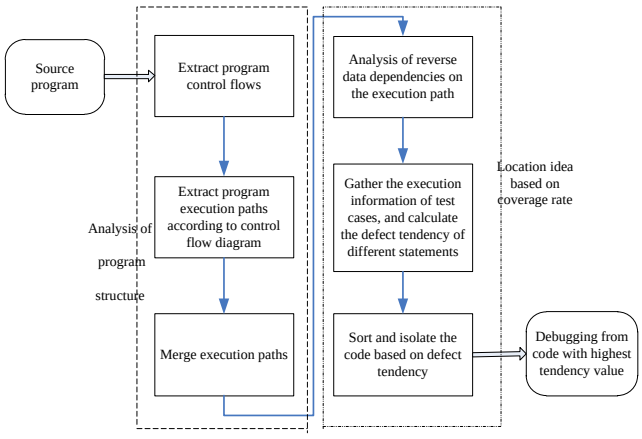

Fig. 2. Block diagram of CPD research idea

Extract program execution path: Control flow analysis mainly extract various execution paths of the program according to the execution of test cases, combining relevant gcc commands.

Merge execution paths: Integrate the streamlined execution paths into a complete dependence graph of the program, to facilitate the follow-up statistical analysis of coverage.

Analysis of reverse data dependencies on the execution path: This is the main work of reverse data dependencies model. Extract the data dependencies on the execution path by analyzing the text of program.

Calculate the defect tendency: Quantify calculation model by combining the statistical error correction, and realize the calculation of different code statements defect tendency by programming.

Sort and isolate the code based on defect tendency: Sort the values of defect tendency from high to low, and then make the isolation.

4. Conclusion

In this paper, by analyzing the advantages and disadvantages of the more generally applicable CBFL and program slicing method, for shortcomings, we proposed a new fault localization method CPD while retaining the advantages. This article needs to do further research in the following areas: how to reduce the cost of data dependencies extracted and how to further improve the accuracy of fault localization without loss of the automated process.

References

1. Changxiang Shen, Huanguo Zhang, Dengguo Feng,Zhenfu Cao, Jiwu Huang. Summary of Information Security. Science in China (Chinese version) , 2007,37(2):129-150
2. Binxing Fang, Tianbo Lu, Chao Li. Survey of software assurance. Journal on Communications (China version),2009,30(2):106-117
3. M. Weiser. Program Slicing. In IEEE Transactions on Software Engineering， 1982 ， SE-10(4)：352-357
4. T. Chilimbi, B. Liblit, K. Mehra, A. Nori, and K. Vaswani. Holmes: effective Statistical Debugging via Efficient Path Profiling. In Proceedings of ICSE 2009. IEEE Computer Society Press, Los Alamitos, CA, 2009.