A NOVEL APPROACH TO ERROR DETECTION AND CORRECTION OF C PROGRAMS USING MACHINE LEARNING AND DATA MINING

Prof. KhushaliDeulkar[1] , Jai Kapoor[2] , Priya Gaud[3] , Harshal Gala[4]
Department Of Computer Engineering D.J Sanghvi College Of Engineering ,Mumbai, India

ABSTRACT

There has always been a struggle for programmers to identify the errors while executing a program- be it syntactical or logical error. This struggle has led to a research in identification of syntactical and logical errors. This paper makes an attempt to survey those research works which can be used to identify errors as well as proposes a new model based on machine learning and data mining which can detect logical and syntactical errors by correcting them or providing suggestions. The proposed work is based on use of hashtags to identify each correct program uniquely and this in turn can be compared with the logically incorrect program in order to identify errors.

KEYWORDS-COMPONENTS

Machine Learning Device(MLD), Data Mining Device(DMD), Databases, Hash-tag.

1. INTRODUCTION

The conventional text-book based learning has gradually been replaced by a more convenient and affordable computer-assisted learning considering its availability and accessibility. [1]With the Introduction to Programming course being made compulsory in almost all academic institutions students with no background in coding struggle to get hold of the syntax and logics of the programs.

Correcting such errors can be a very tedious job when you can't get the meaning of the compile time error messages that you receive. The evaluation of such a huge amount of programs, given the vast amount of pupils the institution has to take, can be very tasking and takes a lot of time.

Similarly, at an organization which involves coding from its employee, sitting and correcting all the syntactical errors will waste a lot of energy and resources. To deal with such issues, we have proposed an automatic error detection and correction system for programs in C. There have been various studies and tools used previously to assist novice programmers to form an error free program and get a correct output.[7] Our system mainly deals with compile time errors i.e. syntactical errors since syntactical errors are significant for the novice programmers. This paper proposes a system which focuses on integration of machine learning, data mining and system programming to detect the errors in the program.

Data mining can be defined as process of analysing data from different and summarizing it into useful information. It is a process of finding correlation or patterns in the information available. [8]The five major steps that will constitute the data mining aspect of the system are: collecting data to mine, determining the table to assist, pre-processing the data, extracting relevant data from raw, data cleaning and formatting it, adapting a mining algorithm, and at last applying mining results.

The four major categories of mining algorithms are[9]:

1) Pattern matching: Finding the instance to data for given pattern.
2) Clustering: Assembling the data in clusters.
3) Classification: Classifying data on the basis of the already classified data.
4) Frequent pattern mining: Locating the frequently occurring pattern.

Now, the correct programs will be stored in the database and organized using data mining. Each of the correct programs will be assigned a hash-tag with help of system programming identification. Whenever an incorrect program with that particular logic will be entered, the system will make a hash-tag based comparison with correct program and using data mining it will detect those errors.

With the help of machine learning, the system will analyse the data, recognize the pattern, learn and then display the errors in the program and suggestions to correct those errors. This will allow the programmers to save a lot of energy and thus get the output faster.

In section II, we reanalyse the previous works on the methodologies and techniques used. In section III we discuss our proposed solution for the issue. In section IV, we have written advantages and disadvantages of the proposed solution. In section V we conclude our paper while describing the scope of the project and the future work possibilities.

2. RELATED WORK

Our research aims to design error detection and correction methods in C programs using data mining and machine learning. Several other researchers have previously worked on the similar domain.

K .K Sharma and Kunal Banerjee [1] have concentrated on the problem of the precedence of the if-else statements and the incorrect ordering of conditions leading to a logical error which standard compilers fail to determine. This is later resolved by tabulating the if-else statements using a set of systematic steps. Firstly, the precedence of the if-else conditions is identified. Secondly, after ordering according to the precedence the innermost conditions are executed and they are compared with the rule table. Comparison is done by converting these statements in normalized form. After normalization of each condition, we now check the ordering of conditions in an else-if construct. Thirdly, a complex analysis is done to compute the time complexity and make it more efficient. The time complexity is done taking two things in consideration: the complexity of comparing two normalized (conditional) expressions and the number of times such comparisons have to be done. This can be used in our project to tackle the if-else condition problem and will reduce the possibility of logical errors related to the if-else conditions to occur.

YuriyBrun and Michael D. Ernst [2] propose a technique for identifying program properties that indicate errors. The technique generates machine learning models of program properties known to result from errors, and applies these models to program properties of user-written code in order to classify and rank properties that may lead the user to errors. Given a set of properties produced by the program analysis, the technique selects a subset of properties that are most likely to reveal an error. An implementation, the Fault Invariant Classifier, demonstrates the efficacy of the technique. The implementation uses dynamic invariant detection to generate program properties. It uses learning techniques like support vector machine and decision tree to classify these properties. It is done by technique in which it has two steps: training i.e. pre-processing step that extracts properties of programs containing known errors and classification i.e. the tool applies the model to properties of new code and selects the fault-revealing properties. This is followed by creation of models and detection of fault-revealing properties. This technique may be most useful when important errors in a program are hard to find. It is applicable even when a developer is already aware of (low-priority) errors.

Tatiana Vert, Tatiana Krikun and Mikhail Glukhikh [3] in their paper “Detection of Incorrect Pointer Dereferences for C/C++ Programs using Static Code Analysis and Logical Inference” have done static code analysis precision using classic code algorithm with dependencies. The key characteristics of error detection methods are based on soundness, precision, and performance. To achieve all the characteristics is contradictory as one of them is to be compromised to increase the efficiency of the other two. This is solved by a logical interface tool by constructing a source code model for the program. The most convenient model which can be used for code analysis is a control flow graph (CFG). In predicate analysis, information about exact values of variables and relations between them is extracted during analysis of program statements. These values and relations can be represented as logical predicates. Later, in pointer analysis, the rules are: pointer correctness, deference of a pointer to a simple variable, deference of a pointer to an element of complex object and summation of pointer and integer constants. Future considerations of the parameters of the rule can be defect detection rule are incorrect pointer deference , buffer overflow and array out of bounds.

The paper by George Stergiopoulos, PanagiotisKatsaros and DimitrisGritzalis [4] is based on automated detection of logical errors based on profiling the intended behaviour behind the source code. The territory of logical errors has yet been untouched and this paper is an attempt to put a light on this subject based on a profiling method that is combined with analysis of information and cross checks dynamic values with its natural symbolic execution and use of fuzzy logic. Errors are classified using fuzzy logic membership ie severity- values from a scale quantifying the impact of a logical error, with respect to how it affects the AUT’s execution flow and vulnerability- with values from a scale quantifying the likelihood of a logical error and how dangerous it is. Profiling is done based on- intended program functionality as Rules (Dynamic Invariants), Program states and their variables, Source code profiling for logical error detection and severity(critical source code points).

The paper by PrakashMurali, AtulSandur and Abhay Ashok Patil [5] is about a logical error correction system in C using genetic algorithm techniques for error correction along with statistical control flow techniques. It is done using expression mining by considering a reduced subset of the C language to probe the challenges of error correction. The logical error occur in the expression is hypothesized of the input program. The first task is to employ data mining on the input program and extract the expressions in the program. The stepwise analysis is shown in the paper. This algorithm can include logical errors in non-mathematical expressions by suitably modifying the code.

M. I. Glukhikh, V. M. Itsykson, and V. A. Tsesko [6], analyse the development of dependency analysis methods in order to improve static code analysis precision “Using Dependencies to Improve Precision of Code Analysis”. They explain the reasons for precision loss when detecting defects in program source code using abstract interpretation methods. Dependency interpretation based on logic inference using logic and arithmetic rules is proposed by them. Defect detection based on abstract interpretation that provides both soundness and high precision is characterized by high computational complexity. The main cause is the need to analyze all the possible execution traces and to store the values of all the reachable objects in these traces. This complete dependency analysis is shown in this paper. The presented research demonstrates that extraction and interpretation of data dependencies is one of the most important aspects of code analysis. There are several directions to improve the suggested approach: development of more precise dependency merging rules, and use of automated theorem proving methods when interpreting dependencies.

3. PROPOSED WORK

Logical Errors has been a rather intangible area to be touched on. Logical error correction can help in saving loads of time along with the desired efficiency and better time complexity. The proposed approach is to compare the two programs and find out errors by detecting the missing invariants using machine learning and data mining. An invariant can be a missing element, statement or number of iterations in a program. There will be a predefined database which will have the profiling of missing invariants. Profiling is basically used for defining the basic use of each invariant using a predefined database .The use of missing invariants will be recognised and organised using data mining and this use will allow the machine learning device to learn the use of each invariant with respect to a particular program. Functions of each problem have similar set of statement unless it has been solved by a different algorithm. With this realisation, each program should be written as a function. So, the two functions can be compared to detect errors. The suggestions provided by machine learning device based on profiling and previous knowledge are displayed to the user wherein the user has a choice of modifying the code or embedding the correct code that is only stored in the database . Embedding the code can be done by replacing the whole code.

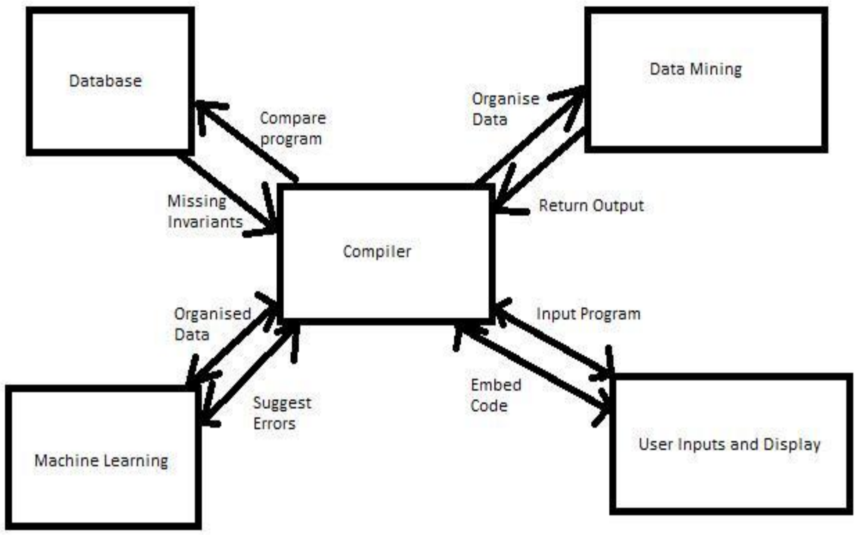

Figure 1:Block Diagram

Step 1: Compiler Construction

The compiler will consist of 4 parts-Machine Learner Device (MLD),Data Mining Device(DMD),C compiler, Database 1 and Database 2.Database 1 will be used to store the correctly executed programs with a hash-tag attached to it(shown in *Figure 2*). A new algorithm for the same problem can be stored separately. Database 2 will be the database used by MLD and DMD for profiling of each variant possible in a program. The data mining device is used to uncover the use of each invariant. The machine learning device will identify the use of each invariant in the user's program.

The compiler should be capable of comparing two programs efficiently and should also be able to find out the missing invariants in the program. The comparison should not be time consuming and the database and the software should work in tandem with that compiler.

Step 2: Programming Construction

The program should be divided into modules or functions. The function provides a base for comparison of the program. The function should provide the same logic as that of the correct program even if variables are defined differently unless a different .Modularity is a must. The construct willdepend upon-indentation, function use and parameters. It would be feasible if parameters in the function are same as it will provide better comparison structure and better efficiency.

Step 3: Comparing the programs

The compiler will compile two programs-the correct program and the incorrect program. The correct program will be the one stored in the database. One thing needs to be taken care of is that the logic should be same. If logic is different but a correct one, then that program will be stored in database. If the logic is different or incorrect one, then correct program will replace the incorrect one without showing the errors. If logic is same, it will evaluate the program and find out the missing code. The comparison should not take much time and should be done efficiently. This missing code will be transferred on to DMD for base profiling.

Step 4: Deducing the errors

Errors will be deduced by predefining the use of each operator, variables and functions etc. The code which was missing will be designated and organised using data mining. The data will be organised according to program and then the use of each element will be checked by DMD. Here, the use of machine learning will play an important part. The machine will learn the use of each element and will implement it according to that particular program. The efficiency and accuracy of the logical error correction should be a prime concern. The machine learning will deduce the errors based on the program as well as the use of profiling.

Step 5: Classifying the errors

The errors here will be classified into logical, syntactical and runtime errors. The syntactical errors can be found out using pre defined profiling. The logical errors require the description of each element along with machine learning of the program. Each of the missing elements have a logic and they need to be identified and classified accordingly. The syntactical and the run time errors faced can be stored in the database as the MLD can learn from these errors for providing future suggestions. They can be used as a reference while executing a similar program. The logical errors need to be individually processed for each program as each program can possibly have multiple logics. The new logic needs to be stored in the database with the same subject name/hash-tag. A hash-tag is basically a reference for storing and retrieving a particular program having a particular logic. The rules for assigning a hash-tag are as follows:

1) Each Hash-tag should be unique.

2) Each Hash-tag will contain a program of unique logic.

3) Hash-tags are case-sensitive. This needs to be taken care of during accessing of program through a hash-tag.

Step 6: Recommending and giving the right solution

The right solution will be based on the type of error detected. If the logic is different and the output is also incorrect, the correct code from the database will directly replace the incorrect code, else it will suggest the solution so that the user is aware of the mistakes. The solution if implemented, works, then that solution can be learnt by the machine so that in the future use, the machine realises the different implementations of logic.

Step 7: Embedding the correct solution in a program

The correct solution can be embedded in the form of macros or new functions as per the user requirement. System programming will play a part here if macros are required else functions will ser etter rea ieve his time co

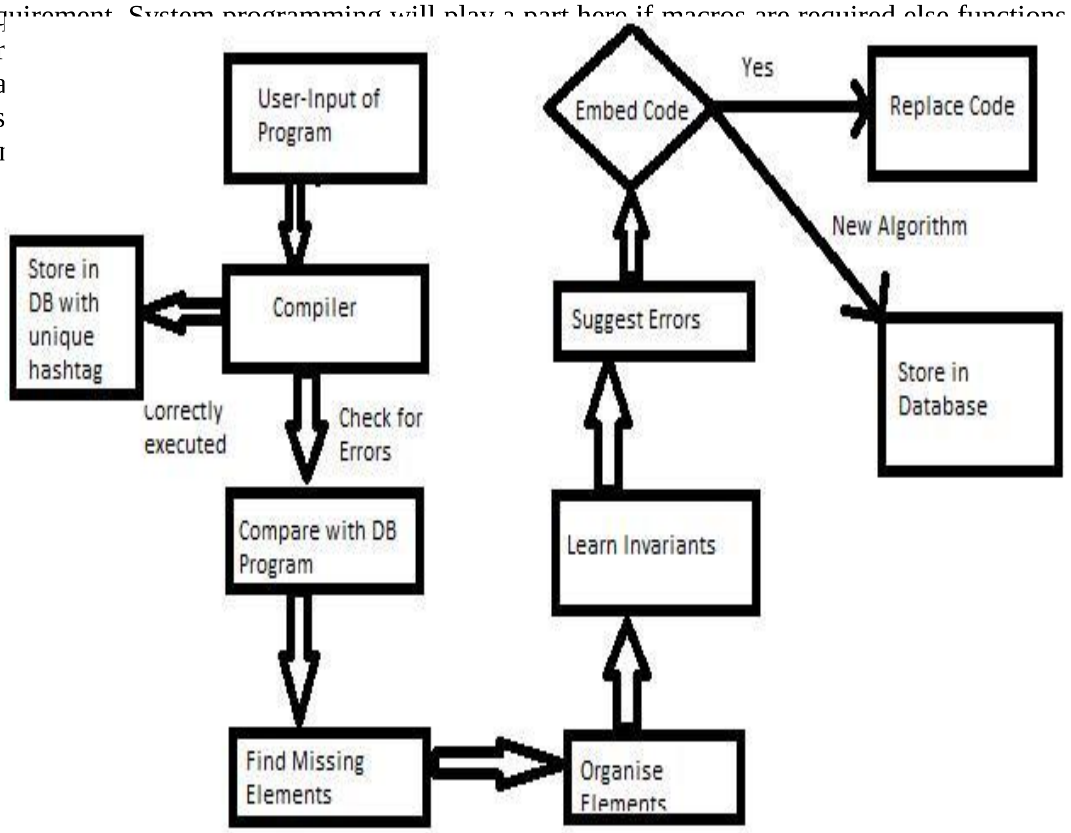

Figure 2: Flow Chart

4. ADVANTAGES AND DISADVANTAGES OF PROPOSED WORK

The advantages of our proposed system are as follows:

- Reverse Engineering will become simpler and efficient as understanding of the programming
- logic will become much easier.
- Time taken for debugging will reduce significantly as logical errors will be suggested which will make user realize of the mistake in the program
- Use of cloud computing will allow sharing of code across the world which in turn will

provide more test cases for machine learning.
- Time complexity and space complexity can also be compared of each correct program, thereby saving loads of time and space in a program.
- It can be implemented across all languages and all platforms.
- The program will only be stored if it is executed correctly. So, the comparison with the program in the database will be syntactically correct.

The disadvantages of our proposed system are as follows:

- Logical errors cannot be detected if the algorithm used to a problem is different. This can however be corrected by adding a new approach (correct program) in the database so that machine can learn the new approach.
- The correct program which is stored in database for reference should be logically correct itself. For example, if the program for addition is stored in the hashtag for subtraction, the comparison of the program will be done incorrectly.
- The logic and operators which is not defined in the database will not be compared which will reduce the efficiency to detect the errors.
- 100 percent efficiency will never be achieved.
- The system is susceptible to various security issues.

5. CONCLUSION

Developing the program is impossible unless the code gets compiled correctly. Therefore it is a very important part of the error correction process. A methodology has been proposed which will assist the novice programmers in realizing various types of syntactical errors and how they can be dealt with. The scope of this project deals with inclusion of more complex run-time and logical errors and correcting programs written in different programming languages like C++ and java.

REFERENCES

[1] K K Sharma, Kunal Banerjee, IndraVikas, ChittaranjanMandal, “Automated Checking of the Violation of Precedence of Conditions in else-if Constructs in Student’s Programs”, IEEE International Conference on MOOC, Innovation and Technology in Education (MITE), 2014

[2] YuriyBrun, Michael D. Ernst, “Finding latent code errors via machine learning over program executions”, Proceedings of the 26th International Conference on Software Engineering (ICSE).,2004

[3] Tatiana Vert, Tatiana Krikun, Mikhail Glukhikh, “Detection of Incorrect Pointer Dereferences for C/C++ Programs using Static Code Analysis and Logical Inference”, Tools& Methods of Program Analysis, 2013.

[4] George Stergiopoulos, PanagiotisKatsaros, DimitrisGritzalis, “Automated detection of logical errors in programs”, Springer-Verlag Berlin Heidelberg 2014.

[5] PrakashMurali, AtulSandur, Abhay Ashok Patil, “Correction of Logical Errors in C programs using Genetic Algorithm Techniques”, International Journal of Recent Trends in Engineering, Vol. 1, No. 2, May 2009.

[6] M. I. Glukhikh, V. M. Itsykson, and V. A. Tsesko, “Using Dependencies to Improve Precision of Code Analysis”, Automatic Control and Computer Sciences, 2012.

[7] V. Neelima, Annapurna. N, V. Alekhya, Dr. B. M. Vidyavathi, “Bug Detection through Text Data Mining”, International Journal of Advanced Research in Computer Science and Software Engineering, May 2013.

[8] Data Mining, available at: https://www.wikipedia.org/

[9] DataMining,availableat: http://www.anderson.ucla.edu/faculty/jason.frand/teacher/palace/datamining.html