Pattern Design for Software Testing Base Finit Automato Machines

Seyyede Roya Alavi[1]

[1]**Department of Computer, Khoy Branch, Islamic Azad University, Khoy, Iran**

Abstract

On method for effective testing is the identification of critical rout in the program. Standardized test of software is somehow impossible because the production and control of critical routs is difficult for software with average size. For creating control in the routs Finit Automate Machines (FAM) are used, in order to design a series of grammer and then the language for each rout by making use of FAM. Grammers create a chain sequences which should be followed through the program. These produced sequences decrease the complexity of identification of errors for effective testing in the process of examination purposefully.

Keywords: *Finit Automato Mach, Grammer, Path, Software Testing.*

1. Introduction

The process of testing any software system is an enormous task which is time consuming and costly software testing spends almost 50% of software system development resources. Random test simply runs the inputs and then clarifies the performed structures but it can't extract some of the accessible information from black box. Dynamic test white and black box methods are used in combination to produce finite amount of test instances. Testing involves three main steps: generation a set of test inputs, execution thos inputs on the program under tests, and then checking whether the test executions reveal faults.

Software testing is a time consuming and costing process. For applying a standardized test all possible paths should be studied. As the program follows running paths according to different inputs, it is impossible to explore all the errors of the program before its practicle use by the software users. Dealing all run paths before program delivery is very difficult or maybe impossible. So, many of errors are concealed in the program and will be recovered after using.An Automatic testing software can generally decrease the cost of software development. Moreover, it can causes quick running of the test and the reliability of test result decreases. Automatic testing is not a direct and confort a progress process [1].

Error Location finding methods are generally distinguished into two types: static and dynamic. Static methods tries to identify the location of errors in the program according to Program Dependent Graph [2,3,4]. On the other hand, dynamic methods try to approximate location of the error by comparing successful and unsuccessful runs of the program [5,6]. Gathering needed data for modeling of run paths of program is an important problem in error finding of the software. Storing all run data, which is produced by the program is not possible in the practice. But only a part of run data of program can be stored.

Solving this problem a pattern of administration can be offered. According to this pattern the behavior of the program can be analyzed during different runs. Offering the pattern for modeling of run paths of the program can be done by grammers in FAM.

The rest of the paper is arranged as follow: In section 2, the review of literature of software testing are discussed. In section 3, suggested methods, by the use of Finit Automate Machines are examined and in section 4, conclusion is given.

2. Related Work

Static testing methods running the program on test by input data testing and recovering its output [7]. The goal of dynamic analyzing methods is comparing the behavior of running program time in successful and unsuccessful run, which detection the program errors [5]. Dynamic methods applying with care of successful and unsuccessful data and without any attention to the programs static structure. Static method is distinguished into two type: black box and white box. Black box testing is essential only in examining the output in response to the input data.

White box testing is made by the use of information about how it works inside the unit [8,9,10]. In dynamic techniques the data results of run program is stored and after analyzing them chaos inside the program code is explored and introduced to the user. Previous research for finding the location of errors from timing behavior of program administration such as program spectra used memory graphs and history of examining determinants of the program [11,12]. Among analyzing technique of dynamic method, techniques based on examining of determinant for finding locations of errors were more successful [13,14]. In dependent graph program, data dependence and controlling dependence of program are shown [15]. In fact, the nodes of this graph are sentences of the program and determinant statements. And their edge shows data dependence and controlling dependence of the program.

Similar to control and data flow coverage criteria, state base testing relies on coverage criteria defined . These include: all-transitions, all-transition-pairs, all predicates and allsequences. Other coverage criteria like all-states and all-ntransitions are very often used. However, all-states (each state must reached at least once) is subsumed by all-transitions criterion which is in turn subsumed by all-n-transitions. While these criteria are typical for state-based testing, in many research publications state machines have been extended to deal with dataflow coverage criteria [15]. In Fig.1, the code of one program with related CFG (Control Flow Graph) is illustrated.

```
0.    Function (int a, int b)
1.   {
2.   Int c;
3.   If (b>a) {
4.   c=a;
5.   a=b;
6.   b=c;
7.   }
8.   C=a/b;
9.   While (c!=0)
10.  {
11.  a=b;
12.  b=c;
13.  c=a/b;
14.  }
15.  Return  b;
16.  }
```

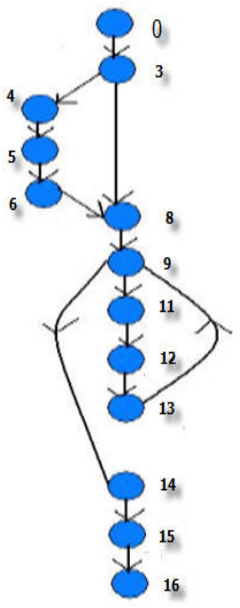

Fig.1 Code With CFG

3. Proposed Pattern

The path which the programs starts and ends, is the program behavior. It is clear that a program can have different behaviors. The more complexity of software the more behavior domains of the program. The behavior of the program can be sequence of its occurrences, such that, the occurrence can be the calling function, running a line of program, or the return value of function. The behavior of program can be true or false. LTL (Linear Temporal Logic) formulae can be converted mechanically into testautomata and then used in a model checking procedure, using the algorithm outlined [16]. The automaton will accept all those, and only those, execution sequences that correspond to a violation of the property [16]. The model checker SPIN contains the conversion algorithm, and can detect the violating sequences with a standard model checking run [16].

Any violations that are detected can then be reported as execution traces through the original implementation source code of the application.

The test automata are often also simple enough that they can be constructed by hand, and in some cases the hand-tuned automata are smaller than the machine generated ones,which translates to reduced run-time requirements for the model checking process [16].

3.1 Finit Automato Machine

Machines are design according to a language, and an input string enter finally gives an output which are YES or NO. In this part, the program characteristics are enters instead of language characteristics and the output is produced based on the language of machine grammer. The figure of a machine is illustrated in Fig. 2. The finite Automate which display with M, is formed from five elements: M = (Q, $\sum$, S, q_0, F), in which Q is series of q_0,q_1,q_2, …, q_i , $\sum$ is series of input string scripts, S is the rules of transfer or shifts, q0 is a member of Q and F is series of final conditions.

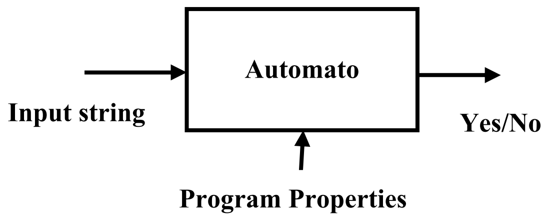

Fig.2 Automato

There is used a graph for showing a finit automate machine, Fig.3 display a simple graph of FAM. Such that the communication between state is called edge and illustrated S(qi , a) = qj.

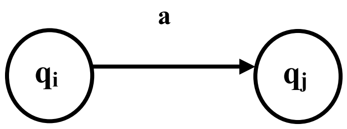

Fig. 3 edge of Automato

The grammer is used for description the language of FAM. The grammer is formed of four elements which expressing with G: G = (V, T, S, P). In grammer G, V element implies the de-terminal series and T implies the terminals, S is the sign of start and P is sign for series of theorems. For obtain the sequence of strings carring the left side of string and right side of string are replaced in a term, which said derivation.

Grammer:

S→ aSb
S→ λ

For example, the sequence of ab, aabb is done by following:

Derivation:
S→ aSb → ab
S→ aSb→ aaSbb → aabb

The path which is negotiate during running, is designed with FAM. Using FAM cause designing a series of grammer for each function inside the testing program. In Fig. 4 a FMA gas designed for program in Fig. 1. For drawing a graph of this FAM, follows that each line of program numbered and each line is a state and edges of graph for defining function take F and "RETURN", data defining, "{", "}" lines take label and because of conditions, like "IF", are labeled I and because of "WHILE", are labeled W, because of "ELSE" labeled E, because of correcting condition labeled C inside the while, if, … and because of correct and wrong conditions it gives t and F. because of every rule and terms which are running in the program, the label S is given for edge of graph.

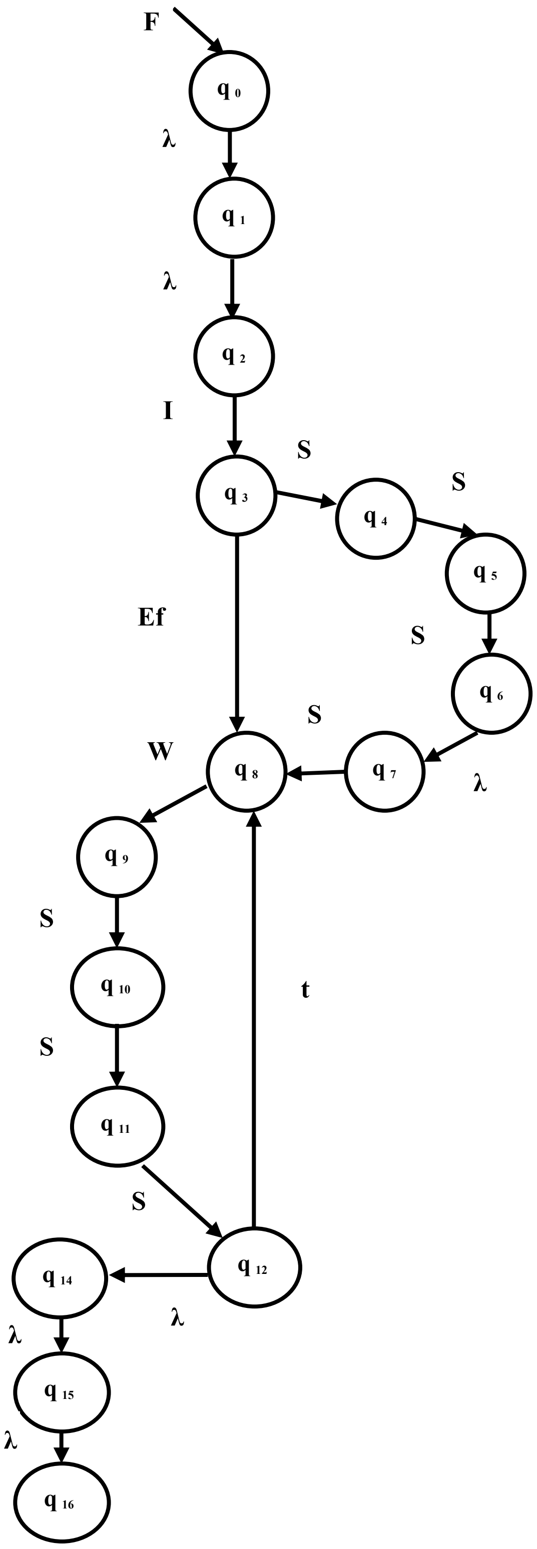

Fig. 4 Finit Automato Machine

In this case, the value of de-terminal V element and value of terminal T element are given:

V = { F, E, I, W, C, S}
T={ S, λ, t, f}

Therefore, the grammer which applied for Fig. 4 4 is in the followed. According to this grammer, language (like L) developed for it.

Grammer Fig. 4:

$F \rightarrow IF \mid EF \mid \lambda$
$I \rightarrow CS \mid \lambda$
$C \rightarrow t \mid Ef$
$S \rightarrow sS \mid s \mid \lambda$
$E \rightarrow SW \mid \lambda$
$W \rightarrow CW \mid S \mid \lambda$

Language:

$L=\{ (fs)^n (tsss)^m \mid n >= 0, m > 0\}$

Different strings are designed for grammers which displays the sequence of followed paths, next these sequences compared with that language. If the sequence of produced string is differ from produced strings of the languages, there is an error in the code. For example, if there is a wrong in line 3 (b<a) string "tsss" for input (2,3) produced from grammer and language. String "fsf" produced from grammer and shows that inside of part "IF", because of wrong conditions, deviated from run path of program.

Derivation(true by grammer and language):

$F \rightarrow IF \rightarrow I \rightarrow CS \rightarrow tS \rightarrow tsS \rightarrow tssS \rightarrow tsss$

Derivatin (false by grammer):

$F \rightarrow IF \rightarrow I \rightarrow CS \rightarrow fES \rightarrow fSWS \rightarrow fsCWS \rightarrow fsfEWS \rightarrow fsfWS \rightarrow fsfS \rightarrow fsf$

4. Conclusion

In software testing, location finding or suspect cases for errors in codes of program is the goal. FAM is used for finding the errors location, so that one way for error detection designed base on grammer and language is for program. In this paper, using this pattern, all the paths should followed are designed used of program language. All these produced series should be the string series which produced of program grammer. In this case, the left series of strings are recording as paths which the errors occur in them.

References

[1] Srivastava, P. and Kim, T., "Application of Genetic Algorithm in Software Testing", International Journal of software engineering and its applications vol.3.no.4,2009, pp. 87-95.
[2] W. R. Bush, J. D. Pincus and D. J. Sielaff, "A Static Analyzer for Finding Dynamic Programming Errors", Software Practice and Experience, Vol. 30, No. 7, 2000, pp. 775−802.
[3] D. L. Detlefs, R. M. Leino, G. Nelson, J. B. Saxe, "Extended Static Checking", SRC Research Reports SRC−159, Compaq SRC, December 1998.
[4] W. E. Wong, S.S. Gokhale, and J.R. Horgan, "Measuring distance between program features", In International Conference of Computer Sotware and Applications (COMPSAC), 2002, pp. 307-312.
[5] T. Ball,"The Concept of Dynamic Analysis", In Proceedings of the 7th European Software Engineering Conference and the7th ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE'99), September 1999, pp.216-234.
[6] M.D. Ernst, J. Cockrell, W.G. Griswold and D. Notkin, "Dynamically Discovering Likely Program Invariants to Support Program Evolution", In IEEE Transactions on Software Engineering,Vol. 27, No. 2, February 2001.
[7] Myers, G. J., The Art of Software Testing, Revised and Updated by Tom Badgett and Todd M.Thomas with Corey Sandler,John Wiley & Sons , Inc, Second Edition, 2004.
[8] BCS SIGIST, "Standard for Software Component Testing", British Computer society, SIGIST, 2001.
[9] Gardner, D., "Software Testing Guide", Information management systems& services, California institute of technology, 2006.
[10] B. Liblit, M. Naik, A.X. Zheng, A. Aiken, and M.I. Jordan, "scalable statistical bug language Design and Implemenation (PLDI)", 2005.
[11] M. Renieris and S. Reiss, "Fault Localization with NearestNeighbor Queries", Proc. 18th IEEE Int'l Conf. Automated Software Eng. (ASE '03), 2003, pp. 30-39.
[12] P. Arumuga Nainar, T. Chen, J. Rosin, and B. Liblit, "Statistical debugging using compound Boolean predicates", In proceeding of International Symposium on Software Testing an Analysis, London, 2007.
[13] B. Liblit, A. Aiken, X. Zheng, and M.I. Jordan, "Bug isolation via remote program sampling", In Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, San Diego, 2003, pp 141–154.
[14] S. J. Zeil,"Perturbation techniques for detecting domain errors", IEEE Transactions on Software Engineering, 15, June 1989, pp. 737-764.

[15] Bouchachia, A., Mittermeir, R., Siclecky, P., Stafiej, S. and Zieminski, M., “Nature-Inspired Techniques for Conformance Testing of Object-Oriented Software”, Applied Soft Computing. J, 2009, pp. 1-16.

[16] Holzmann, J. G. and Smith, H. M., “A Practical Method for Verifying Event-Driven Software”, 1998.

Seyyede Roya Alavi received the M.Sc degree in Computer Engineering in 2011 from Iran university of Islamic Azad University, Zanjan, Iran. She has published several papers about Software Testing. She is also lecturer at Azad University Khoy branch.