Graph-Based Analytical Approach to Testing Programs

Mitko M. Mitev and Pavlina Vladimirova[2]

Abstract: **In the paper is presented an approach based on theoretical properties of graphs for structural interpretation of programs and opportunities are provided for planning and structural testing programs. The approach includes a number of developed tools, providing specialized programming environment for testing software modules.**

Keywords: **graph theory, generators of input streams, roads and contours, programming controls**

I. INTRODUCTION

The process of testing is one of the milestones in software technologies for industrial production of programs. It is characterized by the following major parameters:

- *Type of the test system:* ordinary software systems, systems with distributed databases, network systems, real-time systems, Internet applications etc. The technology and used methods of testing directly dependent on the type of tested system.
- *Object of test:* from separate programming procedure, through testing the functionality of the developed classes and ends with a series of tests of the software system.
- *Goals of tests:* detection and localization of errors at different stages of used software technology, examining the various operating parameters, determined in the technical assignment, such as productivity, response time, a priority system, behavior in extreme conditions, data protection and opportunities for recovery etc.
  - *Used methods*: different types of internal and external tests, real-time tests etc.
  - *Tools:* input stream generators, programs for simulating phenomena and processes, fictitious modules etc.
  - *Compatibility with the design process:* preliminary - based on preliminary project, accompaniment software system development and final – after the final development of separate parts or the whole system.

The proposed graph-based analitical approach for testing refers primarily to local executed program modules for detection and correction of errors at the design stage. It is in the class of internal, structural tests, called "white box" method.

[1]Mitko M. Mitev, Technical University - Varna, Department of Computer Sciences and Engineering, Studentska 1, 9010 Varna, Bulgaria, E-mail: mitevmm@abv.bg

[2]Pavlina St. Vladimirova, Technical University - Varna, Department of Computer Sciences and Engineering, Studentska 1, 9010 Varna, Bulgaria, E-mail: pav_varna@yahoo.com

II. FORMALIZATION OF THE PROBLEM FOR TEST

For internal test it is necessary to represent the programming module as a structure [3]. Therefore, in every program module the structure determined operators have to be found. Without violation the algorithm of the program, in accordance with the semantics of the operators, they are transformed into so-called IF structures. This allows the control structure of the program to be presented as a finite oriented graph *G (X, U, P),* where:

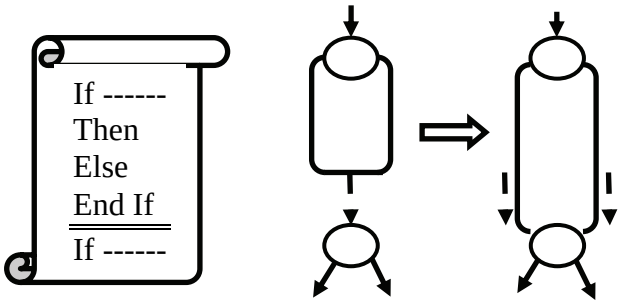
X is the set of vertices. Each vertice represents IF operator of the modified structure.

U – Set of edges of the graph. Each edge represents a linear section of the program. The edge must be oriented and must connect two vertices.

P – Incidentor, which is three-place predicate and introduces relationship between the vertices and arcs of the graph. The predicate takes value true, if two specific vertices and edge are incident each other. Otherwise, the predicate takes value false.

In this transformation there are two special cases:

- Sequence of two IF operator with linear section between them (the example on Figure 1 is in *Visual Basic. Net*). By analogy, the same is true for other programming languages.



a b c

Fig.1. Transformation of the IF structure with a conecting linear section

a) code of the program
b) performance with impaired connectivity
c) conversion with duplication of the common linear section

- Cycle with post condition (Fig. 2)

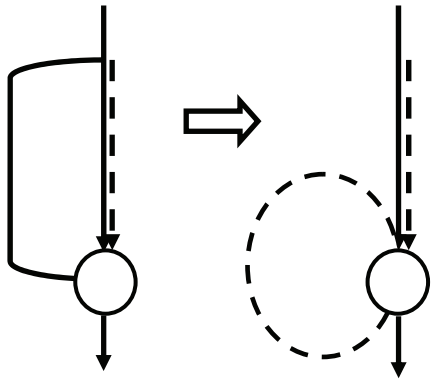


Fig.2. Transformation of cycle with post condition
a) the presentation of impaired connections
b) conversion with duplication of cyclic section

Such presented program can be tested with known methods[1,2].

Let's is given the set of data $D = \{d_k\},\ k \in \{1, K\}$, which is usually classified as input, output or intermediate. On the other hand input data can be divided into data for calculations, control and diagnosis. By analogy, output data are numerical results, diagnostic or data used for control (structuring) of the next software modules.

For the purposes of testing data can be classified as:

- data involved in the composition of the operands, but does not change its value,
- data involved in a decision making,
- data, which during the execution change their value,
- double or triple mixed data from the previous three groups.

On this basis, for each data in the structure of the program may be specified path (paths) of its movement. Each path starts from the place in the structure of the program, where the data appears for the first time and ends with its last use. The path of the input data, represented by its values, starts from the moment of calling the module for execution. The path may end in the logical end of the module if they appeared as results included in the interface.

The path of each data can be described as a sequence of vertices and edges. For example, let $X = \{x_j\}, i \in \{1, N\}$, where $N = |X|$ and $d_k \in D$, with initial vertex ***i*** and vertex *m (m> i)*, one of the possible paths $s_{i,m}^k \in S$ of testing is described as

$$x_i, u_{i,i+l} x_l, u_{l,l+1} \text{K K K } u_{l+n,m} x_m \qquad (1)$$

In particular, it is possible *i = 1* and *m = N*. This is a case, when data is both input and output. Otherwise, it is an input and participates in the formation of intermediate results or it is output, but caused of the execution of the program module. Let's are given two vertices $x_i x_j \in X$ and $i \neq j$ and $P(x_i, u_{i,j}, x_j)$ is true, i.e. $u_{i,j} \in U$. Therefore, can be determined following, generally crossed subsets of data, i.e. the conditions in the two vertices is determined by data included in them

$$\{d_k^i\} \subset D;\ \{d_k^j\} \subset D \qquad (2)$$

Similarly,

$$\{d_k^{i,j}\} \subset D \qquad (3)$$

are data included in the calculation of the linear section of the program.

Paths of two data are *directly independent* unless they include common elements such as vertices and edges. Therefore, these paths can be differentiated as two independent tests. Otherwise, they are *directly dependent,* because they have common data, which in accordance with the algorithm can determine changes in values or paths of execution.

It is recommended directly dependent paths to be tested independently of each other in an exchange sequence. Implementation of the second test will depend on the results of the first test and contrariwise. This sequence does not exhaust all possible cases of directly dependent paths, but decided the main ones.

Two routes are *indirectly dependent*, if there is a third path, which is directly dependent on the previous two paths, because its implementation could lead to termination of one (or both) path or to include other paths. The concept of *indirect dependence* can be extended to two sequences of directly dependent paths, but obligatory finish with a path, which is directly dependent on paths in the last two sequences.

Testing at this situation can be done in reverse order, starting with the path connecting the two sequences and alternatively the rest directly dependent paths of both sequences are changed.

This approach is convenient for a description of a linear section of the programming module. In the case of cycles, calling the other modules for execution, implementation of early or late binding or creation instances of classes, the approach is not applicable due to its low efficiency.

III. METHODOLOGY FOR TESTING

Testing of the program module includes the following steps:

1. *Preparation of the module.*
   - Determination of the programming module for testing and placing it in the program environment.
   - Analysis of program text and identification control statement.
   - Select the linear sections (if any), which are out of a control statement and its subsequent inclusion in the linear sections of the block structure of the previous control statement.
   - Finding of cycles with post-condition (if any) and duplication of the cycle body, according to method, suitable to the occasion.

- According to the text sequence, numbering of all real and additionally included IF operators, resulting from the transformation of the control statements.
- Establishment of the related linear sections in the obtained IF structure and respectively their dual indexing.
- Tabular or list representing of the structure as a finite oriented graph ***G (X, U, P)*** with a completely determined sets and values of the three-place predicate.
- For each vertex or edge is determined the data, included in their composition, whish are a subset of all data.
- An unique link is determined between the vertices (2), edges (3) and subsets of data.

2. *Formation of paths and marking of test sequences.*
- For each of the specified data is determined the location of its first appearance and respectively the final point of its existence.
- According to (1) are determined the paths for the existence of the data in the program. For this purpose in the graph G is looked for routes from the specified initial and final vertex.
- The routes are tested in pairs in order to establish their *direct independence*. Firstly these routes are numbered in increasing sequence (counter and identifier of the test path in the program are formed).
- If there are any *direct dependence* paths, tests are numbered with the next counter values by alternative change of execution of the two roads.
- If there are proved *indirectly dependence* on the third or other paths then have to analyze the sequence of roads with *directly dependence* until be found a direct dependent path.
- If such path exists, then the test counter begins with it and alternatively increases in both branches till reach the first two routes.
- Otherwise directly dependent paths are marked by incrementing counter.

3. *Preparation of data and testing.*

Tests are carried out according to their identification number. For that purpose have to prepare the necessary data:
- The input data are determined, also their range of variation, accuracy of presentation, etc.
- The data are classified into specified four groups.
- For each data are defined two classes: correct and incorrect values.
- All data involved in testing path obtained value of the class of correct values and next value of the incorrect class.
- Tests are carried out in a test environment with consistent monitoring of the executed path and temporary and final values.

4. *Analysis of the results.*

The analysis of the results is performed by well known methods of direct comparison or mathematical statistics when a comparison is impossible.

III. Conclusion

In the report is presented a way for transformation programming text into so-called IF structure and its subsequent interpretation as a finite oriented graph. An analysis of the used data is introduced and their classification is proposed, according to their actual participation in the calculation process and the decision making, depending on the conditions in control statements. On this basis, subsets are defined for every linear section or control statement. As a result, is proposed methodological sequence for determining test routes according to their mutual dependence (or independence).

The proposed theoretical algorithms are developed and implemented as a software library of tools. They can be used alone or be included in the composition of a software system for testing.

References

[1] J. Zhao, "Data-Flow-Based Unit Testing of Aspect-Oriented Programs", Department of Computer Science and Engineering Fukuoka Institute of Technology, Japan,2001
[2] P. Jorgensen, *Software Testing. A Craftsman's Approach*,2nd ed.-,CRC Press,2002 , 359
[3] S. M. Kauffman, "Graph Theory and Linear Algebra" , 2007,105 http://home.comcast.net/~smka2436/Graph_Theory_and_Linear_Algebra.pdf