Chinese Journal of Electronics
Vol.21, No.2, Apr. 2012

The Algorithm of Infeasible Paths Extraction Oriented the Function Calling Relationship*

MU Yongmin, ZHENG Yuhui, ZHANG Zhihua and LIU Mengting

(*Beijing Information Science and Technology University, Beijing 100101, China*)

Abstract — It is a hot issue to detect and extract the infeasible paths in the test oriented the function calling relationship. In this paper, an algorithm is proposed to extract the infeasible function paths. By traversing the source codes and analyzing the conditional branch correlations, the proposed algorithm builds a mathematical model oriented the control flow, data flow and the correlations between the modules. The experimental results show that the algorithm can extract the infeasible function paths efficiently and accurately. The algorithm can save the testing cost effectively and improve the testing efficiency.

Key words — Function calling path, Infeasible path oriented function, Conditional branch correlations.

I. Introduction

Path testing[1] is a method using static analysis of codes. With the program control graph, fundamental executable paths can be derived by analyzing control structure and annular complexity. Then we can design the testing cases.

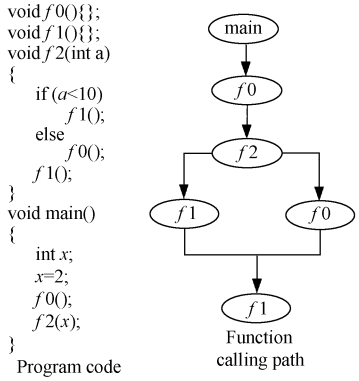


Fig. 1. Function path example

Testing adequacy is an important issue in software test. It is based on code sentences, so the particle size is small and the number is large. It is difficult to realize. Ref.[4] combines the implementation of path test and function test, and then proposed an algorithm to obtain all the static function paths. There are many infeasible paths. Fig.1 is an example as shown.

There are two static function paths: main-> $f0->f2->f1->f1$, main-> $f0->f2->f0->f1$.

Because the value of x is 2, the branch in function $f2$ is impossible to be executed. So main-> $f0->f2->f0->f1$ is an infeasible function path.

The existence of infeasible function paths can bring two problems: one is misdetection; the other is the waste of resources. Therefore, extracting the infeasible function paths is necessary for testing efficiently.

There are two methods to detect the infeasible paths generally in the static codes analysis. The two methods are respectively using path conditional satisfiability[5] and path conditional correlations[6−10]. The former is so complex that it costs much and it is difficult to apply in the project. The latter is using the conditional correlations in the paths to detect the accessibility of the current path. Ref.[11] points out that 9 to 40 percent conditional sentence have the correlations and it is efficient to extract infeasible paths using the conditional correlations in the practical projects. There has been some related research using the path conditional correlations to detect the infeasible paths in all the possible program paths or the base paths. The feasibility has been proved. This paper is aiming at the program paths oriented to the function calling relationship, which is called function paths and the paper proposes an algorithm to extract infeasible function paths using conditional correlations.

II. Basic Concepts

Definition 1 Program block[4], it specifically refers to the specific program structure in the C language and is a statement sequence constitutes of keywords.

Definition 2 Variant field, the variant field of a program block is the sets of variables which can be used in this program block. The solution of the variant field needs the following four sets:

In(B) the sets of all the variables which can reach the program block B.

Gen(B) the sets of all the variables which are generated in the program block B.

*Manuscript Received Oct. 2010; Accepted Oct. 2011. This work is supported by the Key Discipline Construction Foundation of Beijing (No.PXM2011-014224-113530).

Ref(B) the sets of all the variables which is not generated but referred in the program block B.

Current(B,S) the sets of all the variables which the sentence S can use in the program block B, $S \in B$.

Definition 3 There are two operations between the common atomic propositions[12] in the C language.

(1) Sum operation:

$A : x > z$

$B : x_i + y > z$, x_i is the reference of x or $x_i = x$

If $y > z$ then $A \rightarrow B$

(2) Product operation:

$A : x > 0$

$B : x_i * y > 0$, x_i is the reference of x or $x_i = x$

If $y > 0$ then $A \rightarrow B$

If $y \leq 0$ then $A \rightarrow \neg B$

It is obvious to get or indirect reference variables to deduce $A \rightarrow B$ or $A \rightarrow \neg B$ like these two kinds of atomic propositions A and B above, then, we call proposition A and B are correlated, or else, saying that A and B are uncorrelated.

Definition 4 Propositional correlations:

For the two atomic propositions A and B in the Definition 3, if the relationship $A \rightarrow B$ or $A \rightarrow \neg B$ can be derived immediately or indirectly by referring the variable table, then the propositions A and B have the correlations. Otherwise, the propositions A and B are uncorrelated.

The propositions A and B are uncorrelated if they satisfy one of the following conditions:

(1) $I^A \cap I^B = \emptyset$ and $\forall i_A \in I^A$, $\forall i_B \in I^B$, i_A, i_B are uncertain, I^A and I^B are respectively the sets of variables of the propositions A and B.

(2) $A = \{a_i | i = 0, 1, \cdots, n\}$

$B = \{b_j | j = 0, 1, \cdots, m\}$

$\forall a_i$, $\forall b_j$, a_i, b_j are uncorrelated, a_i and b_j are respectively the sets of atomic propositions of the propositions A and B.

(3) $A = \{a_i | i = 0, 1, \cdots, n\}$

$B = \{b_j | j = 0, 1, \cdots, m\}$

a_i and b_i are respectively the simple conjunctive formula of the main disjunctive normal form of the propositions A and B.

$\forall a_i$, $\forall b_j$, a_i, b_j are uncorrelated.

Definition 5 The conditional branch correlation

If the logic value of current condition can't be determined, then we can get it by the correlations of the current condition and the other conditions in this path.

There are two cases to generate infeasible paths generally:

(1) The variables result in identically true or false condition. Thus, some branch will be infeasible. As a result, the infeasible path generates.

① $i \leftarrow 5$;
② if $i > 3$ then
③ do program block 1
④ else
⑤ do program block 2

The branch is identically true in line 2, so the program block 2 is non-executable. Thus the paths contain the program block 2 are infeasible.

(2) The conditions of different program blocks have conflicts, then some infeasible paths generate.

① if $i > 3$ then
② do program block 1
③ if $i < 3$ then
④ do program block 2
⑤ else
⑥ do program block 3

In the example above, the condition in line 1 and the one in line 3 have conflicts. Thus the paths which contain the program block 1 to the program block 2 are infeasible paths.

According to the examples above, we can summary the relationship between the conditional correlations and the accessibility of the program blocks.

The entry conditions of the program block B is called $C^{current}$, the set of conditions which can reach this program block is called $C = c_0 \Lambda c_1 \Lambda c_2 \Lambda \cdots$.

(1) If C and $C^{current}$ are uncorrelated, then the program block is accessible.

(2) If C and $C^{current}$ are correlated and $C \rightarrow C^{current}$, then the program block B is accessible.

(3) If C and $C^{current}$ are correlated and $C \rightarrow \neg C^{current}$, then the program block B is infeasible.

III. The Description of Models

Basing the thought which expresses the semantic and grammatical structure with a control dependence tree proposed in Ref.[13], the paper puts forward the model (a) and the model (b). Then it integrates the two models to form the model (c). Then the paper designs an algorithm according the model (c) and judges the correlation of the conditions using the model (d). Thus the infeasible paths are extracted.

1. Control flow structure[4]

For the three basic structures which are select, sequence, and cycle structure in the program language, the algorithm uses the Boundary-interior thought[14,15] to convert them to select structure. According to the similarity of the select structure and the forest and the mutual conversion of the forest and the binary tree, the paper chooses the child-brother model which is as the Fig.2 shows.

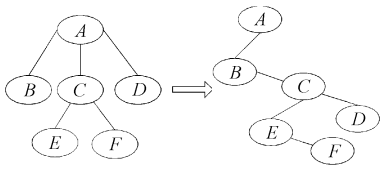


Fig. 2. Control flow structure

In the program, each node of the tree represents a function. Generally, the first branch of the node is the left child. If the node has no branch, then the function which the node will execute is the left child. The right child has the coordinate relationship of the node, thus it represents the branch of the node's parent.

2. Data flow structure

In the process of the program connection, the algorithm needs to obtain the correlation of the set of variables $p^{current}$ in the current program block and the set of the variables p^{parent} in the program block to connect. The algorithm also needs to

access the variables in their life circles in each program block. Thus the paper uses the following data structure as the Fig.3 shows.

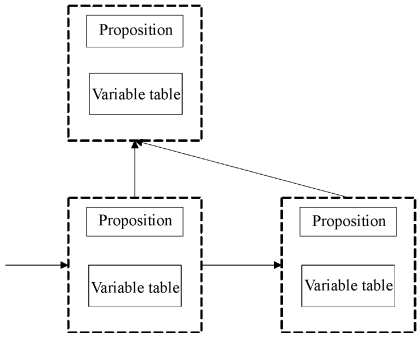


Fig. 3. Data flow structure

As the Fig.3 shows, it is a kind of down tree structure. Each node represents a program block, contains the variables table and the proposition p in this program block.

3. Program block structure

As the Fig.4 shows, there are three program blocks. They respectively call the function a, b and c. The two program blocks in the bottom of the Fig.4 have the coordinate relationship. The proposition in the program block is the conjunction of the propositions from the tree root to the current node.

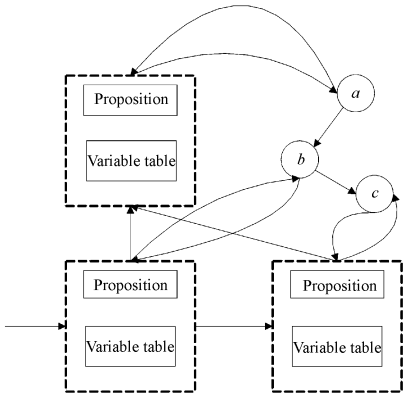


Fig. 4. Program block structure

4. Logical expression

The paper expresses the logical expression with the binary tree structure. The leaf represents the atomic proposition. The node which is not the leaf represents the "and" or the "or" operator. The "not" operator is considered to be the atomic proposition. The logical expression as the Fig.5 shows is $!a\&\&b > 1||x + y < b\&\&c$.

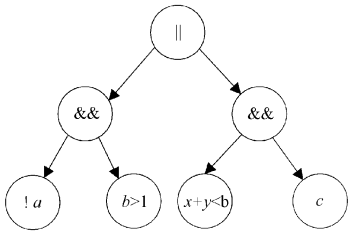


Fig. 5. Binary tree of the proposition

Basing the automatic derivation algorithm for the logical expression with the binary tree structure[16], the algorithm can attain the correlation of the conditions which is expressed as IsFeasible(P). P is the proposition of the current program block.

IV. Algorithm Design and Realization

The algorithm creates a block node, a control flow node and a data flow node for each program block. Then it analyses the proposition P of the program block and refers to the variables table to replace the variable of which the value is determined. Finally, it pushes the block node into the main stack.

When a program block is finished, the algorithm pops the program block tree and analyses the logical value of the proposition in the current program block. If the value is identically true or it is uncertain, then the algorithm traverses all the leaves in each control structure of the parent program block, attaining the correlation between the proposition and the one of the current program block. The algorithm to attain the correlation is following:

SETS
The set of the propositions in a path:
$$p = p^0 \Lambda p^1 \Lambda p^2 \Lambda \cdots$$
VARS
The connecting way: Type
The head of the data flow in the stack top: D_t^{head}
The current head of the data flow structure: D_c^{head}
The accessible of the program block: IsFeasible (P)
The control flow of the data flow: D.cNode
The data flow of the control flow: B.dNode
Copy the program block: Copy(B)
if Type then begin
// the brother connection
$$D^{temp} \leftarrow D_t^{head}$$
while ($D^{temp}.kext$) then begin
$$D^{temp} \leftarrow D^{temp}.next$$
end
$$D_t^{head} \leftarrow D_t^{head}.parent$$
end
$Dt \leftarrow D_t^{heat}($
while Dt begin
$$P^{temp} \leftarrow Dt.p \Lambda p^{current}$$
if (IsFeasible (P^{temp})) then begin
Copy($B^{current}$)
if D^{temp} then begin
$$D^{temp}.next \leftarrow Dc$$
end
$$Dc.parent \leftarrow Dt$$
Connect (B^{parent}.cNode, $B^{current}$.cNode)
while ($Dc.next$) then begin
$$Dc \leftarrow Dc.next$$
$$Dc.parent \leftarrow Dt$$
end
$$D^{temp} \leftarrow Dc$$
end
$$Dt \leftarrow Dt.next$$
end

Connect(B^{parent}.cNode, $B^{current}$.cNode), it means to connect the current program block with the parent program block. There are two kinds of the connection[4]:

• Child connection: Copy the program block to connect with the node of the parent program block. If and only if the relationship between the parent block and the current block are sequence, then we can use the child connection.

• Brother connection: Connect the current block to its parent block tree's child's brother node chain. If and only if the current block has the coordinate relationship with its parent block, we use the brother connection.

V. Experiment and Evaluation

```
Test 1:
i = 1;
j = −1;
if(i < 0)
   a();
else
   b();
if (j < 0)
{
   c();
   if (i > 2)
      d();
   else if (i > −1 && i <= 2)
         e();
      else
      f();
}
```

Note: $a()$, $b()$, $c()$, $d()$, $e()$, $f()$ are functions.

The following figure is the part of the block node of the program above.

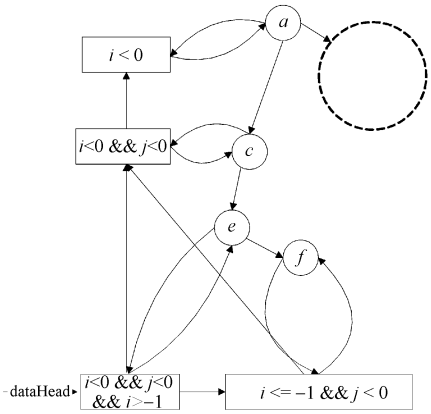


Fig. 6. Part of the program block node

The result of the experiment is as the Fig.7 shows.

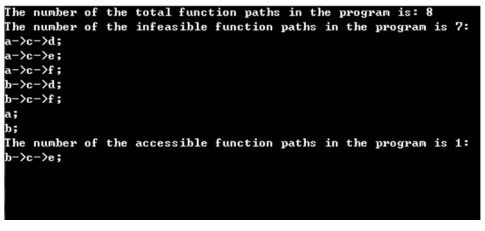


Fig. 7. The result of the experiment

Analysis of the result: As the result shows, the algorithm extracts the infeasible function paths effciently and accurately.

VI. Conclusion

The function calling relationship is complex and there exists the uncertainty[17], such as the function overloading, the polymorphism and so on. Thus it is difficult to analyze the function calling paths. It is probably that the amount of the function calling paths is huge, in which where will so many infeasible paths that it increases the cost of testing. The algorithm analyses the control and data flow of the modules and the correlations of the control conditions, and then attains the correlations of the program blocks. Thus, the algorithm extracts the infeasible function paths. It improves the efficiency of the testing and reduces the testing cost.

References

[1] W.E. Howden, "Reliability of the path analysis testing strategy", *IEEE Transactions on Software Testing*, pp.208–215, Sept. 1976.

[2] Zhang Zhihua, Mu Yongmin, "Research of path coverage generation techniques based Function call Graph", *Acta Electronica Sinica*, Vol.38, No.8, pp.1808–1811, 2010. (in Chinese)

[3] Tang Ying, Xu Liangxian, "Aprecision analytical technology for software code", *Acta Electronica Sinica*, Vol.30, No.12, p.2164, 2002. (in Chinese)

[4] Zheng Yuhui, Mu Yongmin, Zhang Zhihua, "Research on the static function call path generating automatically", *The 2nd IEEE International Conference on Information Management and Engineering*, 2010.

[5] L.M. Peres, S.R. Vergilio, M. Jino *et al.*, "Path selection in the structural esting: Proposition, implementation and application of strategies", *Proceedings of XXL International Conference*, Chilean, pp.240–246, 2001.

[6] Yan Jun, Zhang Jian, "An efficient method to generate feasible paths for basis path testing", *Science Direct Information Processing Letters*, 2008.6

[7] Gong Dunwei, Yao Xiangjuan, Xia Min, "Automatic determination of branch correlations in software testing", *World Congress on Software Engineering*, 2009.3

[8] Li Yonghua, Wu Ying, Wang Feng, Wu Guoqing, "A Research on basis path auto-acquire rnethod with infeasible path", *Journal of Wuhan University of Technology*, 2009.2

[9] Cheng Xin, Zhang Guangmei, Li Xiaowei, "Detection of infeasible path in procedure", *Computer Engineering*, 2006.8

[10] Shen Li, Wang Zhiying, Lu Jianzhuang, "Predicate analysis based on path information", *Acta Electronica Sinica*, Vol.32, No.2, pp.191–195, 2004. (in Chinese)

[11] R. Bodik, R. Gupta, M.L. Soffa, "Refining data flow information using infeasible paths", *Software Engineering Notes ESEC/FSE '97, 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp.361–377, 1997.

[12] Di Suyun, Qu Wanling, Wang Hanpin, *Discrete Mathematics Course*, Publishing House of Beijing University, 2003. (in Chinese)

[13] Wang Tiantian, Su Xiaohong, Ma Peijun, "Research on pointer analysis algorithm for program standardization", *Acta Electronica Sinica*, Vol.37, No.5, pp.1104–1108, 2009. (in Chinese)

[14] Ntafos S., "A Comparison of some structural testing strategies", *IEEE Trans. Software Engineering*, Vol.14, No.6, pp.868–874, 1988.

[15] Howden w'Yudong H., "Software testability analysis", *ACM Transactions on Software Engineering and Methodology*, Vol.4, pp.36–64, Jan. 1995.

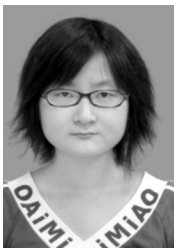
[16] Lan Yang, "An automatic derivation algorthm of the logical exprecsion with the binary tree structure", *Journal of Xinyang Normal University* (*Natural Science Edition*), 2005.4

[17] Huang Zhiyong, Mu Yongmin, Zhang Zhihua, "The application of C++ function overload resolution in regression testing", *Microcomputer Information*, No.24, 2010.

MU Yongmin was born in Yantai of Shandong Province in 1961. Ph.D., professor of computer science in Beijing Information Science and Technology University. His research directions include automated software testing, automatic generation of application software and object-oriented technology. (Email: yongminmu@163.com)

ZHENG Yuhui was born in Jiujiang of Jiangxi Province in 1983, master of computer science in Beijing Information Science and Technology University. His research directions include automated software testing.

ZHANG Zhihua was born in Baoqing of Heilongjiang Province in 1971, associate professor of computer science in Beijing Information Science and Technology University, master. Her research directions include software testing, object-oriented analysis and design. (Email: zhang_zh@bistu.edu.cn)



LIU Mengting was born in Nanyang of Henan Province in 1990, master of computer science in Beijing Information Science and Technology University. Her research direction is automated software testing.