基于函数调用的路径覆盖生成技术研究

张志华,牟永敏
(北京信息科技大学计算机学院,北京 100101)

摘 要: 针对目前路径覆盖方法所存在的缺陷,提出了一种新的基于函数调用的路径覆盖生成方法.根据控制结构与函数调用语法,获取全部的静态函数调用路径,依据程序执行后的动态路径信息,判断测试用例是否覆盖了程序变更部分及受影响部分.该方法既能避免路径数目的急剧增长,又可以保证测试完全.

关键词: 路径覆盖; 函数调用; 控制逻辑; 回归测试
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2010) 08-1808-04

Research of Path Coverage Generation Techniques Based Function Call Graph

ZHANG Zhi-hua, MU Yong-min
((*Computer School*, *Beijing Information Science and Technology University*, *Beijng* 100101, *China*)

Abstract: In view of the gaps of current path coverage approach, this paper presents a new path coverage generation method based on the function call. According to control logic and function call syntax, all of the static function call paths are obtained. Dynamic path information is acquired after program running, and whether the test cases cover the program changes and the affected parts could be determined based on it. This method avoids the sharp increase in the number of paths, but also can guarantee that test completely.

Key words: path coverage; function call; control logic; regression testing

1 引言

软件测试是软件开发过程中的重要阶段,是保证软件质量、提高软件可靠性的关键.随着软件工程技术的发展,软件设计规模的增大,软件测试在软件开发过程中的作用越来越重要.回归测试是在软件开发过程中为确保软件质量进行的一种常用的验证测试方法.在软件生命周期中的任何一个阶段,只要软件发生了变化,就可能给软件的正确性带来影响,就需要进行回归测试[1].回归测试中最简单的方法就是全部重测,但是由于其高昂的代价,在大中型软件项目中很少使用,实际工作中一般采用选择性回归测试方法,如基于变更的选择性回归测试[2]和基于控制流图的回归测试[3,4].本文提出的基于函数调用的路径覆盖生成技术,通过源代码的静态分析结合程序的动态执行,只对被修改代码及其受影响的部分重新测试,从安全性上可以达到与全部重测相同的效果.

2 路径覆盖生成技术

测试充分性是软件测试的一个重要问题.路径覆盖测试是一种针对白盒测试的常用充分性准则,它不仅要求软件测试过程中观察输入、输出的正确性,而且要求观察程序运行的整个路径,要求程序的运行覆盖所有完整路径.路径覆盖是一种非常严格的覆盖准则.但事实上,即使是规模很小的程序,包含的逻辑路径数量也是相当大的,在大型程序中如果多处出现分支和循环则路径数目急剧增加,进行完全的路径测试是不可能的.在这种情况下,测试员往往只能选择一个有一定效果且开销较小的覆盖准则,如分支覆盖准则;或者是根据一定的标准,选择所有完整逻辑路径中的一个有限子集来进行测试,如基本路径覆盖测试就是选择了一个能覆盖系统中所有状态的路径子集.在本文中考虑另外一种客观而又简单的标准:函数调用与控制逻辑相结合,将代码分析粒度由语句扩展到函数,既能避免路径的爆炸式增长,又可以保证测试完全.

路径测试的目标是确定实际程序中的路径集合,用最少的测试用例,发现程序中最多的错误,而最少测试用例应能覆盖程序中的所有路径.为了达到路径覆盖,必须确切知道被测试程序的路径数目,以便安排测试计划、分配测试资源,并对实际测试所达到的覆盖率做出评估,以决定是否结束测试工作.路径覆盖生成技术首先面临的问题是如何确定一条可执行的路径.

现有的路径覆盖生成方法有:基于 DDGRAPH 图的控制流路径覆盖生成方法[5,6]、分段 bi-路径覆盖生成方

收稿日期:2009-09-07;修回日期:2009-10-19
基金项目:北京市教委科技计划(No.KM201010772014)

法[7]和基于边覆盖的路径覆盖估测法[8],下面分别比较它们的优缺点,在此基础上提出新的路径覆盖生成方法.

(1)基于 DDGRAPH 图的控制流路径覆盖生成方法:利用 DDGRAPH 图表示程序结构,并由 DDGRAPH 图生成支配树和蕴含树,形成非限制弧集,最后采用寻找单个测试路径算法生成路径子集.该方法采用了近似最少谓词覆盖策略减少生成覆盖测试路径的数目,降低了构造路径所需要的开销.基于 DDGRAPH 图的路径覆盖方法在测试大程序时,路径表达式会越来越庞大,使其测试时间和运行空间大幅增长,有待于进一步的改进.

(2)分段 bi-路径覆盖生成方法:采用正则表达式来表示被测程序的内部控制结构,将被测程序分成若干个相对独立的测试段.在每一个测试段内部,采用一种改进的 bi-覆盖(边界一内部覆盖)方法来减少由于循环所产生路径数量.实例表明,这是一种十分有效的软件测试方法.分段 bi-路径覆盖生成方法不能识别不可执行路径,不可执行路径的识别问题还没有得到最终解决.

(3)基于边覆盖的路径覆盖估测法:该方法通过包含基本块(Basic Block)和边执行频率信息的边覆盖来估测路径覆盖.生成边覆盖的系统代价较低,且边覆盖易于生成和存储.通过生成边覆盖来获取与程序执行过程相关的数据信息,对该数据信息进行分析,使用一定的路径分析策略来估测程序运行后的可执行路径集和路径集中每条可执行路径的执行频率.

基于边覆盖的路径覆盖估测法可实现在程序动态执行中获取用于测试用例选择的数据信息,但存在估测精确率不高,对重叠路径的识别能力差以及循环结构对可执行路径数目的影响等缺陷.本文针对这些缺陷,提出了一种新的基于函数调用的路径覆盖生成方法来获取程序执行的路径信息,该方法相对于边覆盖估测法具有较高的路径覆盖精度和路径覆盖生成效率.

3　基于函数调用的路径覆盖生成方法

该方法基于的思想是:使用对程序设置“探针”的插装方法[9,10],对被测试程序的源代码进行预处理,生成相应的带有控制逻辑的函数调用关系图,并获取全部静态函数调用路径,确定一组测试用例,在程序的执行中获取动态路径信息(即函数调用序列).将动态函数调用序列转化为静态逻辑路径,建立测试用例和静态路径之间的对应关系,为测试用例选择和测试用例缩减奠定基础.

下面给出相关定义:

定义 1　函数调用图 G 是一种有向图 $G=\langle V,R\rangle$.其中 V 是结点集,$R=\{(x,y)\mid x,y\in V\}$ 是弧集.弧 $e=(T(e),H(e))$ 连接 G 中相邻的两个结点 $T(e)$(称为弧头)和 $H(e)$(称为弧尾).

定义 2　函数调用图 G 的一条路径是一个结点序列 $(v_i=v_{i0},v_{i1},\cdots,v_{im})$,路径中的结点为函数名,路径中相邻结点表示调用关系.

定义 3　在软件测试中,测试用例和路径的关系定义成一个二元关系:给定一个程序 P 和 P 中一条路径 W,T 是一个测试用例,若 P 运行时输入 T,程序运行后的路径转化为 W,则关系 (T,W) 成立.

3.1　预处理

采用函数控制流分支跟踪插装方式,在每个函数入口处和退出处以及每个分支部分插入跟踪代码,记录程序实际执行的函数调用序列.为了确定装代码的位置,需要对源程序进行词法分析和语法分析.这里只对每一个可能的函数调用进行插装,没有函数调用的分支不插装,避免了因插装内容过多而造成的程序性能下降,减少了不必要的系统开销.例如:

```
Func 1(a){
    if(a>0){
        return 0;
    }else{
        return Func 2(a);
    }
}
```

在上面的代码中,if($a>0$)这个分支没有进行函数调用,对整个程序的函数调用图没有影响,不进行插装;else 分支点由于调用了函数 Func 2(),则需要进行插装.上述插装情形同样适用于 switch()、do-while 语句以及各种结构的混合形式.

C 语言有顺序、循环、选择三种语句结构形式,它们可以互相嵌套使用.其中能够产生分支的是循环和选择等基本形式,当然也包括各种形式的混合形式.根据 C 语言的语法规则,以及考虑程序实现的方便性,将能够出现函数调用语句的 C 语言成分划分为四类形式,如表 1 所示.

在插装时可得到每个函数的名称及其定义的起始位置.根据这些信息,找到函数在源代码中的位置,对代码进行静态分析,绘制带有控制逻辑的函数调用图.

3.2　绘制函数调用图

目前关于程序流程图(flow chart)自动生成的工具较多,如 algoview、autoflowchart 等,这些工具绘制流程图的速度很快,能做到代码和图形的双向对照.但它们都是基于语句级,绘制的图形庞大,给出的全局静态路径

表 1 C 语言中可出现函数调用语句的成分划分

分类	形式描述	插装情况
顺序	*S*;	函数内部出现调用关系时进行插装.
循环	for(*cond*) { *S*;}	对 for 和 while 进行插装.满足条件时记录装点.
	while(*cond*) { *S*;}	
选择	if(*cond*) { *S*0;} else if(*cond*) { *S*1;} else{*S*2}	只对有函数调用的情况进行插装.对于 switch 语句来说,对有函数调用的 case 进行插装.而没有函数调用的 case 则不需要插装.
	if(*cond*) *S*;	

	switch(*exp*){
	case const_exp1: *S*1;break;
	case const_exp2: *S*2;break;
	…
	default: *Sm*;
	}	

混合	do{*S*;} while(*cond*);	对引起函数调用的情况进行插装.

数近于天文数字,对于大中型软件项目测试并不适用.

另外,还有开源的 calltree、codeviz、gprof 等可以绘制函数调用关系图(call graph),这些工具简单易用,只需编写少量的代码就能生成函数调用关系图.它们共同的缺点是生成的函数调用关系图没有控制逻辑关系.例如对于以下函数调用语句:

```
main( ) {
    int a = 0;
    f4( );              //顺序调用
    if(a > 1) f1( );    //条件选择
    else f2( );
    f3( );              //顺序调用
}
```

上述工具生成的函数调用关系图只能给出 main 调用了 *f*1、*f*2、*f*3 和 *f*4,没有考虑控制逻辑,如图 1 所示.这样的函数调用关系图所给出全局静态路径不能真实地反映程序控制流程,无法提取函数调用路径信息.

分析可知,函数调用可出现在三类语句中:顺序、分支(if、if else、switch case 等)和循环(for、while 和 do while),其中顺序调用的处理比较容易.把所有包含条件判断的语句(if、for、do…)都作为一个模块处理,称为条件模块,基本思想是用计数器来记录当前的层数以及函数所在的层数,遇到条件语句时,增加一个空的分支,等到它的下一级模块开始时再回填.分析程序结构时记录函数名和所在层次,建立链表用于绘图.得到每个函数的调用图后,根据函数入口,连接顺序执行的函数,得到全局图.图中每个结点是一个函数,每条边是一个调用关系.针对上述语句,采用这种思想绘制的函数调用图如图 2 所示.

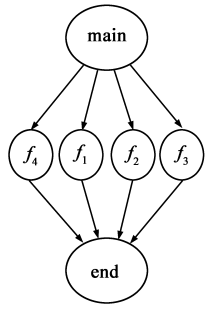

图1 codeviz生成图

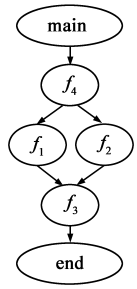

图2 本工具生成图

3.3 动态路径信息获取

执行源程序,记录函数调用顺序.根据装点分析动态路径,然后与静态路径进行精确匹配.基本过程是:通过对源代码的分析,找出所有用户定义的(非库函数)函数,记录函数的装点信息,包括:装点 id、所在的文件以及起始行的行数;通过对分支的插装可以知道每一个可能产生函数调用的分支,记录它的装点信息,包括装点 id、所在的函数等等.函数实际运行时,在每一个函数、分支的进入点和退出点都执行了装点代码,通过对其生成的数据文件进行分析,就能知道每一个测试用例实际执行的函数调用序列,将其与静态函数调用序列进行匹配,就能知道该测试用例覆盖的路径.

该方法是通过程序动态执行来获取用于回归测试用例选择的程序执行路径信息,大大节省了系统时间开销.

4 应用

当前的研究表明通过路径覆盖的比较来进行回归测试用例的选择是一种可行方法.将基于函数调用的路径覆盖技术应用到回归测试中主要包括三个方面,一是找出程序中受影响的部分,选择测试用例重新测试被影响的部分,二是优化测试用例库,三是给出测试用例覆盖度.

(1)程序变更标识,选择测试用例.通过文件比较,在全局图中标出变更的节点.深度优先搜索全局图,找出所有经过变更点的路径.这些路径在回归测试中应该全部被覆盖,才能验证原有功能是否继续可用、软件修改后是否引入新的错误.因此,经过变更点的函数调用路径就是在选择测试用例时的测试需求.

(2)测试用例缩减.分析测试用例和其覆盖的路径,选择必要测试用例,删除覆盖相同路径的测试用例.例如采用贪心算法(或其它启发式算法),当有两个或两个以上的测试用例覆盖同一路径时,可以去掉冗余的测试用例.

(3)测试覆盖度.原有测试用例执行时,已经保存了每个测试用例与函数调用路径之间的对应关系,在

选择回归测试用例时,只需挑选出经过变更点的函数调用路径及其对应的测试用例,然后采用一定的优化算法,选择最少的测试用例满足所有测试需求.如果存在某些经过变更点的函数调用路径,在原有测试用例集中找不到对应的测试用例,则给出相应提示:经过变更点的某条函数调用路径没有被覆盖.

5 结论

如何进行测试用例选择,减少回归测试成本,提高测试效率是回归测试研究中的一项重要工作[11,12].采用基于函数调用的路径覆盖生成技术,通过对软件源代码的静态分析并结合程序运行,获取每个测试用例的动态执行路径,然后与静态路径进行精确匹配,判断测试用例是否覆盖了程序变更部分及受影响部分,从而得出回归测试是否完备的结论.该方法已成功地应用到作者开发的自动路径测试工具中,路径获取高效准确,应用该方法生成的可执行路径数目不受程序中循环结构的影响.

随着软件规模的不断扩大,软件测试变得越来越重要.测试用例设计成为测试的重要工作.基于函数调用的路径覆盖生成技术中,路径测试数据的自动生成是下一步研究重点问题.

参考文献:

[1] M J Harrold, J J Jones, T Li, et al. Regression test selection for Java software[A]. In Proceedings of OOPSLA'O1[C]. Tampa Bay, Florida, USA, 2001. 312 – 326.

[2] G Rothermel, M J Harrold. Analyzing regression test selection techniques[J]. IEEE Transactions on Software Engineering, 1996, 22(8): 529 – 551.

[3] G Rothermel, M J Harrold. A safe, efficient regression test selection technique[J]. ACM Transaction on Software Engineering and Methodology, 1997, 6(2): 173 – 210.

[4] S G Elbaum, P Kallakuri, A G Malishevsky, G Rothermel, S Kanduri. Understanding the effects of changes on the cost-effectiveness of regression testing techniques. Software Testing [J]. Verification and Reliability, 2003, 13(2): 65 – 83.

[5] 丁雪梅,伦立军.基于 DDGRAPH 图的路径覆盖研究[J].微机发展,2004,14(3):29 – 31.
Ding Xue-mei, Lun Li-jun. Research on path coverage based on DDGRAPH[J]. Microcomputer Development, 2004, 14(3): 29 – 31. (in Chinese)

[6] 侯芸,顾刚,高海昌,郭斌.一种路径覆盖自动生成的改进方法[J].计算机工程,2007,33(04):67 – 69.
Hou Yun, Gu Gang, Gao Hai-chang, Guo Bin. An improved method of automatic generation for path coverage[J]. Computer Engineering, 2007, 33(4): 67 – 69. (in Chinese)

[7] 高芳.分段边界内部路径覆盖软件测试方法的研究与实现[D].哈尔滨:哈尔滨工业大学,2002.
Gao Fang. Research and implementation of path coverage software testing based on subsection boundary[D]. HarBin: Harbin Institute of Technology, 2002. (in Chinese)

[8] T Ball, P Mataga, M Sagiv. Edge profiling versus path profiling: The Show down[A]. Proc of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Language [C]. New York: ACM Press, 1998. 134 – 148.

[9] 高海昌,贺晓红,冯博琴,朱利.软件结构测试自动化关键技术研究[J].微电子学与计算机,2005,22(2):25 – 28.
Gao Hai-chang, He Xiao-hong, Feng Bo-qing, Zhu Li. Study of key technologies of software automatic structure testing[J]. Microelectronics & Computer, 2005, 22(02): 25 – 289. (in Chinese)

[10] 孙昌爱,金茂忠.基于程序插装的动态测试技术实现[J].小型微型计算机系统,2001,22(12):1475 – 1479.
Sun Chang-ai, Jin Mao-zhong. Program instrumentation approach to implementing software dynamic testing[J]. Mini-micro Systems, 2001, 22(12): 1475 – 1479. (in Chinese)

[11] 单锦辉,王戟,齐治昌.面向路径的测试数据自动生成方法述评[J].电子学报,2004,32(1):109 – 113.
Shan Jin-Hui, Wang Ji, Qi Zhi-Chang. Survey on path-wise automatic generation of test data[J]. Acta Electronica Sinica, 2004, 32(1): 109 – 113. (in Chinese)

[12] Mary Jean Harrold, Alessandro Orso. Retesting software during development and maintenance [A]. Frontiers of Software Maintenance (FoSM 2008)[C]. Beijing, China, 2008. 99 – 108.

作者简介:

张志华　女,1971 年生于黑龙江宝清,讲师,硕士,主要从事软件测试、面向对象分析与设计等方面的研究.
E-mail: zhang_zh@bistu.edu.cn

牟永敏　男,1961 年生于山东烟台,教授,博士,现为北京信息科技大学开放系统实验室主任,主要从事软件测试技术与方法、计算机网络安全等方面的研究.
E-mail: yongminmu@gmail.com