DOI:10.13543/j.cnki.bhxbzr.2007.s1.007

基于抽象语法树的代码静态自动测试方法研究

高传平[1] 谈利群[1] 宫云战[2]
(1. 北京图形研究所, 北京 100029; 2. 北京邮电大学网络与交换技术国家重点实验室, 北京 100876)

摘 要: 软件测试是排除软件故障, 提高软件质量和可靠性的重要手段。从是否需要执行被测程序角度考虑, 软件测试分为静态测试和动态测试。动态测试通过输入测试数据, 动态执行程序来发现软件中存在的错误。尽管动态测试能发现部分软件错误, 但对于一些特殊类型错误的检测无效。鉴于此, 本文采取了一种特殊的静态分析技术来实现对代码的测试。本文首先讨论了传统软件测试方法的缺点和局限性, 给出了软件的故障模型, 进而提出了基于抽象语法树的静态分析技术, 并给出了故障自动检测算法。依据该算法开发了自动化测试工具, 给出了实验结果和对比分析, 并指出了下一步的研究方向。

关键词: 软件测试; 静态分析; 故障; 故障模型; 语法树
中图分类号: TP302.8

引 言

软件测试是排除软件故障, 提高软件质量和可靠性的重要手段。按是否需要动态执行被测程序考虑, 软件测试分为两类, 即静态测试和动态测试[1]。静态测试是指不运行实际被测的源程序, 而是通过采用其他手段对程序结构进行静态分析, 获得程序结构的相关信息, 并完成对软件故障的检测; 动态测试通过输入测试数据, 动态执行程序来发现软件中存在的错误。动态测试只存在于软件生存期的编码阶段之后。动态测试包括两个基本要素: 一是被测程序, 二是测试数据, 程序一次运行所需要的测试数据称为测试用例, 所以测试数据是测试用例的集合。尽管动态测试能发现程序中存在的某些方面的错误, 但是对于软件本身存在的一些深层次逻辑结构方面的错误或者一些比较特殊的错误的检测是无能为力的。与静态测试相比, 动态测试虽然能在软件运行时发现一些软件故障, 但又明显具有一定的局限性。

(1)动态测试的结果依赖于测试数据的选择。好的测试数据往往能发现软件中存在的更多的故障, 而不好的测试数据很难发现软件中存在的故障, 因此测试数据的选择至关重要。

(2)动态测试在很大程度上受测试人员自身素质和经验的影响。测试人员的经验和素质将会直接影响测试数据的设计, 从而间接对测试结果产生影响。

(3)动态测试的结果有时难以复现。某些故障的发生具有偶然性, 这不仅与测试数据的选择有关系, 而且与程序的操作逻辑及执行次数有关。

(4)动态测试对于一些特殊类型故障的检测无效。动态测试对于软件本身存在的一些深层次逻辑结构方面的错误或者一些比较特殊错误的检测是无能为力的。如动态测试对于不可达代码的测试无效。

为了能有效地解决上述问题, 本文采取了一种特殊的静态分析方法来实现对程序代码的静态检测, 尤其是对于那些用动态方法无法检测的软件问题。

1 故障模型

在传统的软件测试方法中, 一个故障能否被检测到, 是取决于该故障检测概率的大小, 对软件中存在的所有故障的检测而言, 只能以故障检测的百分比来衡量。如果一种故障的故障检测率比较高, 则此种故障比较容易检测, 否则故障就不容易被检测。

目前所存在的软件测试方法, 其测试理论的基础并不是基于故障模型, 而是基于软件中所有可能的故障。这种测试方法的优点是:

(1)能够检测软件中的大部分故障;

收稿日期: 2007-04-05
第一作者: 男, 1977 年生, 博士生
E-mail: cpgzgy@yahoo.com.cn

(2)测试的自动化程度一般比较高。

但存在的问题也是很明显的。

(1)传统的测试方法对“小概率故障”的检测是无效的。“小概率故障”是指那些检测概率很小的故障,例如 10^{-9}、10^{-12} 数量级等,如果不是专门针对这种故障进行测试,而采用传统的笼统的测试方法,这种故障一般是不大可能检测到的。

(2)有些故障是采用传统的测试方法一次性检测不出来的,如内存泄露故障等。采用一个测试用例是很难检测到此类故障的。

(3)难以计算故障的检测效果。只能依概率统计技术来评估故障的检测效果,难以精确计算。

由以上几点可以看出,使用传统的测试方法有其局限性。这些测试方法只能用于一般的软件测试当中。而对于可靠性要求比较高、各种标准比较严格的软件必须开辟新的途径进行测试。

软件测试的故障模型,是为了在软件测试过程中能更好的发现软件中存在的故障,运用形式化语言等数学的方法而对故障进行模型化,进而通过数学运算反映故障内在的规律和逻辑。故障模型的建立不是偶然的,软件测试的最终目标是发现软件中存在的故障,因此建立软件的故障模型有助于从高层次上理解故障产生的原因,把握故障的本质,从而用于指导软件测试工作。如果建立的故障模型能覆盖软件中所有的故障,则说明这个故障模型是比较完整的,同时将这个模型反映到现实的软件测试工作中,也能取得相当好的效果。

软件的故障模型是测试理论的基础,是解决软件问题的关键。建立有效的故障模型对软件中所有故障的标准化将起到不可替代的作用,而对软件故障的标准化会在很大程度上促进软件测试的自动化。

一个成熟的故障模型必须具备下列条件[2]。

(1)该模型是符合实际的。大多数系统中存在的故障都可以用这种模型来表示。

(2)模型下的故障个数是可以容忍的。模型下的故障个数一般和系统的规模是成线性关系。

(3)模型下的故障是可以测试的。也就是说,存在一个算法,利用该算法可以检测模型中的每一个故障。

从上述意义上讲,目前软件中尚无故障模型。所存在的几种软件故障分类方法都是从故障的表现来分类的,或是从故障发生的时间来分类的,显然,这种分类方法对软件测试的实际意义是有限的。故障变异的企图是建立软件测试的故障模型,但这种方法不太符合故障模型的第一个标准,而且由于对变异的对象也不是很明确,它也难以计算实际要被检测的故障个数。因此,软件故障模型的研究势在必行。

2 抽象语法树

2.1 基本概念

定义 **1** 语法树是描述程序语法结构的一种树的图形表示,语法树描绘了如何从文法的开始符号开始推导出它的语言中的一个语句。形象地说,语法树[3]是具有如下特性的树

1. 树根标记为开始符号;
2. 每个叶节点由记号标记;
3. 每个内节点由一个非终结符标记;
4. 如果 A 是某个内节点的非终结符号标记,$X_1, X_2, \cdots, X_n$ 是该节点从左到右排列的所有子节点的标记,则 $A \rightarrow X_1X_2\cdots X_n$ 是一个产生式。这里,$X_1, X_2, \cdots, X_n$ 是一个终结符或非终结符。

依据 C++ 语言的语法规则,语法分析器接受词法分析产生的记号串,并依照相应的规则将记号组织成具有确切含义的语句,并构造相应的语法树。语法树反映了程序语法的推导过程,一棵语法树从左到右的叶节点是这棵语法树生成的结果。譬如赋值语句 position=initial+rate *60 可以用图 1 的语法树表示。

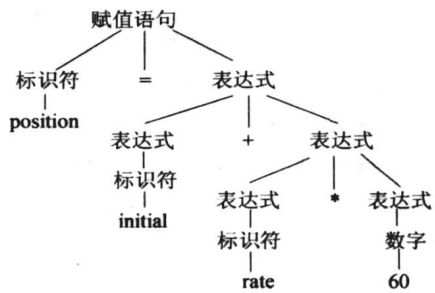

图 1 赋值语句的语法树

Fig. 1 Syntax tree of assignment statement

其中根节点是赋值语句,叶结点表示终结符,即标识符。

语法树是编译程序在经过语法分析以后得到的源程序的另外一种表示,语法分析的结果是生成语法树,每一个语法规则对应一个相应的处理函数,并作为树的一个节点挂在语法树上,提供对外的接口。

由语法树可以生成程序的控制流图和变量的定义使用链，同时定义一些相应的数据结构，为下一步的错误查找作准备。

2.2　语法树的构造

语法树反映了程序结构，语法树的构建是在词法分析、语法分析和语义分析的过程中逐步产生的。语法树的构造过程如图 2 所示。

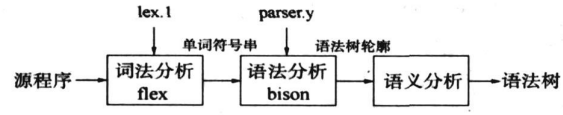

图 2　语法树的构造过程

Fig. 2　Process of syntax tree generation

(1)词法分析[4]　将源程序分解成单词符号串，形成初步的符号表，为语法分析作准备。借助于词法分析工具 flex 编写词法分析程序，编译生成.cpp 文件，通过执行词法分析程序将输入的源程序文件分解成单独的符号串。词法分析提供了语法树所需要的符号节点，如常量和变量等。

(2)语法分析　编写语法分析程序 parser.y，借助于语法分析工具 bison 编译生成.cpp 文件，通过执行语法分析程序将输入字符串识别为单词符号流，也就是判定一个记号串是否能够由一个文法产生。语法分析生成含有代表相应语法结构的中间节点的抽象语法树。该语法树还不是一个完整的语法树，还需要借助于语义分析进一步丰富和细化。

(3)语义分析　语义分析最后通过对名字和操作符的处理将语法树转变为一种标准形式，标准形式中也包括表示类型信息的对象和符号表。经过语义分析后生成的语法树是一个完整的语法树，包含了程序结构的所有信息。

本文使用面向对象的方法来生成语法树。语法树由不同类型的节点组成，每种节点代表了一种不同的语法结构。这些节点包括名字节点、表达式节点、语句节点、声明节点、复合语句、类型节点、函数声明节点、函数体节点、对象节点等等。本文采用一种按类型分层的策略来组织各种节点。这么做的好处是它可以按照某类节点对应的特定的语义规则来设定相应的方法来处理，还可以对同一类型层次上的一系列节点采取同一种语义动作来处理。另外，它还可以针对每一种类型的节点专门制定相应的输出规则。

3　故障检测算法

由图 1 可以看出，树的根节点是 ROOT，其余的是子节点，除了根节点之外，其余的节点被分为两种类型：内节点和叶子节点。内节点有父节点和子节点，而叶子节点只有父节点。每一类节点用相应的符号表示，如代表函数声明的节点 FctDecl，代表语句列表的节点 StmtList，代表变量的数据节点 Object-Variable，每种节点代表一种数据类型，所有类型的基类是 CAstInfo，每种数据类型都定义了相应的属性和操作，以及施加在上面的规则动作、产生的结果、期望的输出及对外的接口。

通过遍历程序语法树，执行函数内置的动作来完成对软件故障的静态检测。因此，基于抽象语法树的代码静态自动测试算法的核心是遍历语法树。语法树的遍历采用递归调用的方法实现。

语法树遍历算法为

输入：源程序文件。

输出：IP(一系列的故障检测点)。

```
void CAstInfo::traverse(begin_actions, after_actions)
{
        bool flag=false;        //节点访问标记
          if(!visited())
          {
              visit(begin_actions); //节点访问前动作
              flag=true;
          if(!sons_visited()) //遍历子节点
          {
              TreePath * p=new TreePath(); //处理回溯
              p->index=tree_path.GetSize();
              p->node=this;
              tree_path.Add(p);
              visit_sons();
              CListIter cali;
              for(sons(cali); cali; ++cali)
              {//递归调用子节点
                  (*cali)->traverse(begin_actions, after_actions);
              }
              if(tree_path.GetSize()>0)
```

```
            {
                delete tree- path. GetAt(tree-
path. GetSize()-1);
                tree- path. RemoveAt(tree-
path. GetSize()-1);
            }
        }
        if (flag)
        {
            visit(after- actions); //节点访问后
动作
        }
    }
```

4 实验

基于抽象语法树的代码自动检测方法是一种静态分析方法,该方法通过对源代码进行词法分析、语法分析和语义分析生成与程序对应的语法树,在遍历语法树前后通过执行相应的动作来完成对软件故障的静态检测。借助于该方法本文开发了自动化测试工具,并对国内外不同专业领域的具体软件项目进行了测试。下面以数组越界故障为例给出部分测试结果,如表1所示。

表1 数组越界故障的部分软件测试结果

Table 1 Results of array bound test on several kinds of software

项目编号	文件大小/M	代码行数	IP	Defect	False Positive	Deprecated	故障密度/(个/KLOC)	准确率/%
1	62.00	310000	150	45	53	52	0.145	30.00
2	17.20	85000	14	6	7	1	0.071	42.86
3	7.70	38500	16	12	2	2	0.312	75.00
4	25.80	132173	50	12	24	14	0.091	24.00
5	15.70	81640	20	4	5	11	0.049	20.00
6	23.20	114144	9	3	1	5	0.026	33.33
7	9.67	50284	58	13	41	4	0.259	22.41
8	6.60	33000	24	16	3	5	0.485	66.67
9	27.70	132960	13	5	4	4	0.038	38.46
10	2.98	11920	53	5	30	18	0.419	9.43
11	23.50	106243	30	6	7	17	0.056	20.00
12	1.11	5620	4	2	2	0	0.356	50.00
13	18.90	100170	24	24	0	0	0.240	100.00
14	6.46	27778	43	9	20	14	0.324	20.93
15	23.20	116018	9	3	5	1	0.026	33.33
合计	271.72	1345450	517	165	204	148	0.123	31.91

表1中故障密度反映了每千行程序代码中存在的数组越界故障个数,即

$$\text{故障密度}=\frac{\text{故障总数}}{\text{代码行数}}\times 1000(\text{个/KLOC})$$

准确率表示经人工复查后确认的故障数目占IP总数的百分比,即:

准确率=Defects/IPs

表1中IP是指源代码经测试工具测试后报告的检测点个数。这些检测点需要经过人工确认后才能确认其状态。每个IP经人工审查后其状态可分为以下三种情况:

(1) False Positive 该处不存在故障,这种用法是正确的;

(2) Deprecated 故障无法确定;

(3) Defect 此处经确认后确是错误。

由表1数据可以看出,每个软件或多或少的都存在数组越界故障,但分布密度不同,为了能直观的反映此类故障在不同软件中的分布程度,本文对上面的测试结果进行了分析,并绘制了不同项目下数组越界的故障密度曲线,如图3所示。

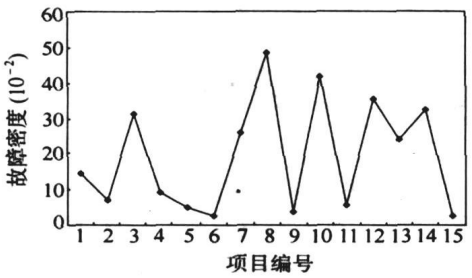

图 3　不同项目下的故障密度曲线

Fig. 3　Fault density curve of different projects

5　结束语

代码测试是目前软件测试研究的一个焦点，程序代码中隐含的错误对软件安全造成了潜在威胁，降低了软件质量及可靠性。传统的代码测试通常采用代码走查或代码走读的方式进行，采用这种人工的方式进行检查不仅费时费力，而且很难发现其中的错误，因此不仅效率低，而且效果比较差。本文提出的这种静态检测方法通过对源程序进行分析，生成与程序对应的语法树，通过遍历程序语法树，匹配软件故障模型，自动查找程序中存在的各种错误。这种测试方法省时省力，效率高，在检测过程中不需要人为干预及开发环境支持，只需要对测试结果进行确认，因此自动化程度高，应用范围广泛。本文通过应用这种静态测试方法，开发了自动化测试工具，并给出了测试工具的实际工程应用。实验结果表明，本文采取的方法是正确的，在实际工程应用中也发挥了很好的作用。在以后的研究工作中，我们将进一步深入研究软件的故障模型，提高软件故障检测算法，减少故障漏报和误报。

参考文献：

[1]　郑人杰. 计算机软件测试技术[M]. 北京：清华大学出版社，1992.

[2]　宫云战. 软件测试的故障模型[J]. 装甲兵工程学院学报，2004，18(2)：1－5.

[3]　AHO A V，SETHI R，ULLMAN J D. Compilers principles，techniques，and tools[M]. America：Addison Wesley，1986.

[4]　吕映芝. 编译原理[M]. 北京：清华大学出版社，1998.

Research on the syntax tree-based method for static and automated code testing

GAO ChuanPing[1]　TAN LiQun[1]　GONG YunZhan[2]

(1. Beijing Graphic Institute，Beijing 100029；2. Network and Exchange Technology Country Key Laboratory，Beijing University of Posts and Telecommunications，Beijing 100876，China)

Abstract：Software testing is an important means to eliminate software faults and enhance software quality and reliability. Depending on if the program is executed，software testing methods are divided to two kinds：static testing and dynamic testing. Through inputting some testing data and executing program，dynamic testing can find some errors in a program，however，it is ineffective in checking certain kinds of errors. Therefore，a special static analysis method is taken to implement code testing. The paper discusses the disadvantage and limitation of the traditional software testing method，builds software's fault models，then puts forward a static analysis technique based on abstract syntax tree，and presents an automatic fault detection algorithm. Based on this method，a software testing system was designed and developed，experiment results were obtained and comparison analysis was made，providing a direction for further study.

Key words：software testing；static analysis；fault；fault model；syntax tree