◎博士论坛◎

# 基于模型检测的数据流异常测试技术研究

陈 涛[1,2,3],许金超[1,2],钮 俊[1,2]
CHEN Tao[1,2,3],XU Jinchao[1,2],NIU Jun[1,2]
1.同济大学 计算机科学与技术系,上海 201804
2.国家高性能计算机工程技术中心 同济分中心,上海 201804
3.安徽财经大学 计算机科学与技术系,安徽 蚌埠 233030
1.Department of Computer Science and Engineering,Tongji University,Shanghai 201804,China
2.Tongji Branch,National Engineering & Technology Center of High Performance Computer,Shanghai 201804,China
3.Department of Computer Science and Engineering,Anhui University of Finance & Economics,Bengbu,Anhui 233030,China

**CHEN Tao,XU Jinchao,NIU Jun.Data flow anomaly analysis based on model checking.Computer Engineering and Applications,2011,47(25):1-4.**

**Abstract:** The execution of program embodies flows of data in variable.A novel method is proposed to detect data flow anomaly including variable undefined or defined but not referenced or multi-used.The trust pattern of program is defined and program is translated into finite state machine.Based on ALCCTL temporal logic and model checking,verification model satisfies the trust pattern defined.Experiment shows that this method is effective and has been implemented in a defect-oriented testing system.

**Key words:** variable referenced;software testing;data flow anomaly;model checking

**摘 要:**程序的执行体现为数据在变量中的流动。对C/C++源代码中变量定义使用情况进行分析,针对变量未赋值就使用、变量重复赋值和变量定义后未使用三种数据流异常情况,使用程序阅读自动机,把程序转换为变量状态机,使用ALCCTL时序逻辑和模型检验工具,验证程序是否满足定义的可信模式。提出了新的静态查找变量使用故障的方法。该方法已应用于面向故障的软件测试系统中。

**关键词:**变量使用;软件测试;数据流异常;模型检测
DOI:10.3778/j.issn.1002-8331.2011.25.001 **文章编号:**1002-8331(2011)25-0001-04 **文献标识码:**A **中图分类号:**TP311.5

## 1 引言

伴随着云计算和物联网时代的到来,计算机给人类发展带来了更多的惊喜,人们越来越依赖于计算机。同时,由于软件错误,计算机也让人类付出巨大甚至惨重的代价,如导致飞机失事、空间飞行器任务失败等各种灾难。为了让软件能正确运行,在过去的十几年里科学家们进行了不懈的努力和研究,比如采用软件测试技术暴露软件的错误,采用软件调试技术对故障进行定位并改正,遗憾的是这些方法只能表明故障存在,而不能证明故障不存在。因此人们发展了形式化方法,如定理证明和模型检验[1],试图从理论上来证明一个程序是无误的。但由于状态爆炸等原因,形式化方法主要应用在嵌入式等一些小型软件上,离实用程度还很远。

找出程序的所有错误是不可判定的[2]。在大型软件开发中,采用软件测试技术来保证软件的开发质量得以首要应用,成为当前保证软件质量的主力军。软件测试以发现和消除软件内部隐藏的故障为中心,以提高软件质量为目的。软件测试的方法很多,根据测试时程序是否运行,可分为静态测试和动态测试。动态测试,是需要输入相应的测试数据,运行程序,检查实际输出结果和预期结果是否相符。动态测试在测试时需要执行程序,不能在早期阶段实施。静态测试是指不实际运行被测软件,通过对需求文件、设计文件及源程序的阅读和分析找出其中的错误或可疑之处。实践表明,大约30%~70%的逻辑设计错误和编码错误可以通过静态测试分析得出。静态测试通过分析或检查源程序的方法、结构、过程、接口等来检查程序的正确性,可以在软件开发生命周期的早期进行,把错误消除在“萌芽”阶段,可以节省大量的开发时间和

**基金项目:**国家自然科学基金(the National Natural Science Foundation of China under Grant No.90718015);国家高技术研究发展计划(863)(No.2007AA01Z425,No.2009AA012201);国家重点基础研究发展规划(973)(No.2007CB316502);NSFC-微软亚洲研究院联合资助项目(No.60970155);教育部博士点基金项目(No.20090072110035);上海市优秀学科带头人计划项目(No.10XD1404400);高效能服务器和存储技术国家重点实验室开放基金项目(No.2009HSSA06);安徽省自然科学基金(No.11040606M151)。

**作者简介:**陈涛(1972—),男,博士研究生,副教授,CCF学生会员,主要研究方向为并行计算和模型检测;许金超(1984—),男,博士研究生;钮俊(1976—),男,博士研究生。E-mail:9chen@tongji.edu.cn

**收稿日期:**2011-01-26;**修回日期:**2011-05-31

成本。静态测试包括代码检查、静态分析两种途径。它可以由人工进行,充分发挥人的逻辑思维优势,也可以借助软件工具自动进行。代码检查包括桌面检查、代码审查、代码走查和技术评审等。主要检查代码的设计是否一致性、代码是否遵循标准性和可读性、代码的逻辑表达是否正确性、以及代码结构是否合理性等。静态分析则是一种计算机辅助的静态分析方法。主要对程序进行控制流分析、数据流分析[3]、接口分析和表达式分析等。

程序的执行体现为数据的流动。目前,一个很有意思且很有前景的辅助性方法是通过数据流分析技术静态地定位错误[4]。数据流分析可以在所有可能的执行路径上找到错误,而且不用等到软件全部完成就可以进行测试。数据流测试目前主要集中在检测内存泄漏[5]、基于变量使用[6]如变量定义未使用[7]、变量未初始化赋值[8]等方面的研究,主要采用以语法树和控制流图的对变量的赋值和使用情况进行跟踪来查找数据流的使用异常。本文结合软件测试和模型检验技术,另辟蹊径,把变量的使用抽象成程序的信任模式,使用模型检测技术进行验证,如果不满足,找出其反例,从而达到找到数据流的异常情况的目的。

## 2 数据流分析及其信任模式

### 2.1 问题描述

从数据流观点来看,程序的执行可以看成变量从声明、定义(赋值)到最后使用的一个循环过程。

**定义**1(变量声明) 在程序中给出变量的类型,并指出该变量所占的内存空间。例如,int $x$,float $y$都是声明一个变量。在C/C++语言中,全局变量和静态变量的初始数据值自动设置为零,而局部非静态变量的初始值则是不确定的。

**定义**2(变量定义) 在程序中给出变量的初始值,因此,比较通俗的说法是变量赋值或变量初始化。可以在变量声明的同时对变量进行初始化,也可以在变量声明后通过赋值语句获得,如int $x$=5,$y$=8.0等。变量的定义有一个特征,变量在赋值号的左侧。

**定义**3(变量的使用) 在程序中使用变量进行运算的过程,如$x$=$x$+5等。

其中,变量声明错误很容易在程序的编译过程检测出来,而变量的定义和使用有时是检查不出来的,而且有时可能是程序员的误写,但变量的定义和使用也会造成程序的一些不可预料的后果或者程序的崩溃。限于篇幅,在本文中对于数据流的分析主要限于对变量使用的分析上,着重解决以下三个问题:

(1)变量未初始化。对于未初始化就使用的变量,一般的编译器都能够进行识别,容易出现未初始化变量故障的情况是在程序执行的某个路径上初始化了该变量,但是另一个路径上,该变量又没有被初始化,从而导致了未初始化变量的故障。此外,变量未初始化可能是由于程序员的误用,比如在程序中使用赋值语句$x$=$x-y-1$,这里$1$以前没有定义。这里$1$有两种情况,第一可能真的是在初始化前使用,这会变得非常危险,因为当变量被使用时,其当前值是未知的,可能会导致灾难性后果;第二种可能是笔误,比如其实可能是数字1而不是字母$l$。

(2)变量定义(赋值)后没有使用。比如函数调用$x$=$f$($x$,$y$),之后再也没有使用$x$,这类异常会造成程序冗余,使程序的可读性比较差;还有一种可能是这个赋值语句用错了,比如应该是$t$=$f$($x$,$y$),而不是$x$=$f$($x$,$y$)。

(3)变量在两次定义(赋值)之间未被使用。比如下面程序:

$$\left.\begin{array}{l}X=f(y)\\ \vdots\\ X=f(w)\end{array}\right\}\text{中间变量}x\text{没有使用}$$

这里有可能出现$x$=$f$($y$)赋值冗余或者在这两个赋值语句之间遗漏了对$x$的某些使用语句。

总之,变量使用异常问题通常不容易被编译器所识别,因为编译器只识别语法,对变量的语义使用无法察觉。因此,这种测试特别对于使用弱类型检查的语言(例如C/C++),通过对变量的检查,可以对程序执行语义进行合理、有穷和近似的计算,它对系统静态表示进行分析以发现问题,文档和代码分析工具对其会有一定帮助。

### 2.2 程序语言中的信任模式

在本文中,把满足以下规则的变量正确使用称为是满足可信模式的使用。

**定义**4(可信模式) 所谓可信模式是指判断研究对象是否可信的标准或依据。

例如,可以根据程序中变量使用的先后顺序也判断程序是否有故障,这里的变量使用的先后顺序就是判断程序是否可信的一个可信模式。

**定义**5(程序信任模式) 所谓程序信任模式是指对于程序中变量的使用进行结构和内容规范性的描述,其作用根据变量的使用来判断程序的故障。

根据上面叙述,对程序中变量的使用情况进行了归纳和总结,提取了程序中变量使用的如下可信模式:

**规范**1 变量在使用之前一定要先定义。

**规范**2 两次变量定义之间一定要使用。

**规范**3 变量最终一定要被使用。

这些规范主要从变量的使用上来对程序进行测试,这些错误也许不影响程序的运行,但会降低软件的质量和程序运行的效率,从规范角度来说,就是降低了程序的质量及可信性。

## 3 基于ALCCTL的程序可信模式描述

### 3.1 ALCCTL描述逻辑

传统上的对软件进行静态测试需要对程序的源代码进行语法分析、词法分析和语义分析,进而生成抽象语法树。然后基于抽象语法树查找相关信息生成定义使用链和控制流图。在这基础上使用研制的软件测试工具测试有关软件,对测试工具认为可能出现的IP(Inspection Points)检测点进行标记,再由测试人员对照这些IP,复查源代码,最终确定这些IP点是否是一个Defect[6]。在本论文中,使用ALC来进行描述。ALC能将一个领域中基于对象的知识表示进行形式化描述,可以用其描述程序中的内容信息;为了描述变量的先后使用定义情况,引入时序逻辑CTL。CTL在普通逻辑的基础上增加了表示时序的修饰符,用其描述变量之间的时序关系,结合了描述逻辑ALC和分支时序逻辑CTL,使用文献[9]提出的一种时序描述逻辑ALCCTL,不仅描述了程序中变量定义与使用之间的时序关系也描述了程序中语法和控制信息。

ALCCTL拥有命题连接词$\neg$,$\subseteq$,$\wedge$,$\vee$,$\forall$,$\exists$等,时态连接

词AG,AF,EF,EX,B,U等。其中,AG表示在所有路径上的所有状态,AF表示在所有路径上都存在一个状态,EF表示某条路径上存在一个状态,EX表示在某条路径上的下一个状态,这些时态连接词后面可以跟一个ALCCTL概念或是一个ALCCTL公式。B,U的前后一般都是ALCCTL概念或ALCCTL公式,$x\ B\ y$表示$x$在$y$之前出现,$x\ U\ y$在$y$出现之前一直是$x$。如$AG(\neg Variable\ Undefined)$表示在所有路径上的所有变量都要满足在使用之前一定要先定义。

### 3.2　可信模式的ALCCTL描述

为了能对第2.2节中定义的可信模式进行验证,需要对可信模式进行形式化的描述。根据软件测试的观点,由于定义的可信模式不仅涉及到变量使用上的时序关系,还涉及程序的语法和语义,因此采用文献[10]所描述的ALCCTL进行可信模式的描述。结合ALCCTL的描述能力,用Variable表示变量,Defined,Undefined,Referenced,Abnormal分别表示程序中变量的定义,未定义,引用以及异常等关系。用ALCCTL时序描述逻辑对第2.2节中定义的规范依次进行描述,如下:

**规范**4　$AG(\neg Variable\ Undefined)$

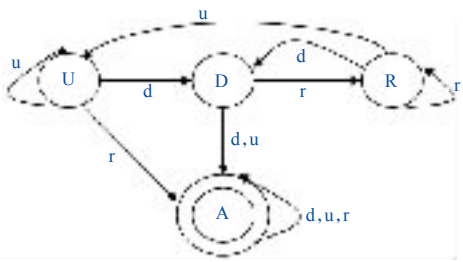
**规范**5　$\neg AE(DefinitionBDefinition)$

**规范**6　$AG(Definition \rightarrow EXUsed)$

## 4　程序阅读自动机

### 4.1　程序阅读自动机

文献[3]把变量的使用看成是程序的一个个状态,把数据的流动就是状态的变迁。在刚开始时,一个变量刚分配内存,还没有赋值,说是"未定义"状态(U);然后,变量经过计算被赋值,说变量处于"定义但没有引用状态"(D);最后,已赋值的变量处于"引用"(R)状态,如果这个变量被读被使用。如果这个变量被重新赋值,则回到"D"状态,一个变量也可以被释放,如关闭已经打开的文件,重新回到"U"状态。图1描述了变量的状态及变迁。




| States | Actions |
|---|---|
| U:Undefined | d: Defined |
| D:Defined but not referenced | r: Referenced |
| R:Defined and referenced | u: Undefine |
| A:Abnormal | |

**图1　程序变量的状态转换图**

当然,不恰当地使用变量会造成变量使用上异常。比如,对处于"未定义"状态的变量进行读操作,对处于"定义但没有引用"状态的变量继续定义或者取消定义操作都会导致异常(Abnormal)。对变量的状态变迁用程序阅读自动机来处理。

**定义**6(程序阅读自动机)　程序阅读自动机是指描述程序变量使用状态的一种特殊自动机,可以用于程序变量信任模式的验证。

### 4.2　程序阅读自动机的构建

如上所述,通过对变量的跟踪可以得到程序的变量状态图,利用这个状态图很容易得到程序阅读自动机。通过阅读自动机可以很形象地得到一些数据流异常的模式。具体的构建算法如下:

**算法**1　程序阅读自动机生成算法Pautogen

输入:程序

输出:程序阅读自动机

```
Pautogen(program,fsm)
{
    1   enum State:U,A,R,D;
    2   State=U;
    3   While (ch<>eof);
    4   Swich (state)
    5  {
    6     Case U:
    7     { if action=u
    8          State=U;
    9      Else if action=d
    10          State=D;
    11       Else action=r
    12            State=A;
    13       }
    14    Case D
    15       If (action=r)
    16          State=R;
    17       Else if (action=d or u
    18            State=A;
    19    Case R
    20           If (action=r)
    21          State=R;
    22       Else if (action=d)
    23            State=D;
    24        Else if (action=u)
    25            State=U;
    26   Case A
    27         If (action=a or u or r
    28            State=A;
    29 }
```

通过算法Pautogen(Program,fsm),就可以将源程序文本转化为基于状态转化的自动机模型,通过自动机很容易转化为模型检测的Kripke结构,是下一步对程序可信模式验证的基础。

## 5　基于模型检测的数据流异常验证算法实现

模型检测是一种形式化验证方法,其基本思想是用Kripke结构来表示状态迁移系统(S)的行为,用时序逻辑公式(F)描述系统的性质。这样"系统是否具有所期望的性质"就转化为数学问题"状态迁移系统S是否是公式F的一个模型?",即S⊨F?。对有穷状态系统,这个问题是可判定的,即可以用计算机程序在有限时间内自动确定。首先把上章产生的fsm自动机转化为Kripke结构M,从内向外逐层抽取待验证性质的子表达式,对满足该子表达式的状态进行标记,标记算法适合于自动机模型的时序逻辑公式的验证。

### 5.1 算法

以第3章定义的程序信任模式和第4章构建的文本阅读自动机为基础,通过以下这个改进的标记算法对程序阅读自动机进行检测以确定其是否满足3.2节中的ALCCTL公式。具体算法如下:

**算法2** 改进的标记算法Label($M$, $\phi$)

输入:结构M和3.2节中的ALCCTL公式集合$\phi$

输出:输出使M,s⊨$\phi$成立的所有s

Label(M, $\phi$)

{

1. 将性质$\phi$由内向外分解为一个子表达式序列:$\{g_0,g_1,\cdots,g_n=\phi\}$
2. 对所有$s\in S$
3. case g=
4. False:没有标记;算法结束
5. $\in AP$:mark(s)$\leftarrow$f;如果f是原子公式
6. $\neg f_1$:mark(s)$\leftarrow$mark(s)+{$\neg f_1$};若$f_1\notin s$
7. $f_1\vee f_2$:mark(s)$\leftarrow$mark(s)+$f_1\vee f_2$;如果$f_1\in s\vee f_2\in s$
8. $E[f_1\cup f_2]$:mark(s)=mark(s)+$E[f_1\cup f_2]$;如果$f_2\in s\vee f_1\in s$且$E[f_1\cup f_2]\in$Next s
9. $EXf_1$:mark(s)=mark(s)+$EXf_1$;如果$f_1\in$Next s
10. Endcase
11. }

整个算法的复杂度是:$O(|f|\cdot(|S|+|R|))$,其中,$|f|$是$f$的子表达式数,$|S|$是状态数,$|R|$是状态迁移数。

### 5.2 实验结果及分析

基于上述变量使用的三个异常(变量未初始化、定义未使用以及变量重复定义),以程序不同的大小块用本文提到的方法和某公司的软件测试工具进行对比检测,实验结果比较如表1所示。

**表1 测试结果统计表**

| 源代码(行数) | 错误数 | | | | | |
|---|---|---|---|---|---|---|
| | 变量未初始化 | | 定义未使用 | | 变量多次定义 | |
| | Reasoning公司 | 本文开发的工具 | Reasoning公司 | 本文开发的工具 | Reasoning公司 | 本文开发的工具 |
| 程序1(5.2万) | 456 | 562 | 0 | 590 | 0 | 407 |
| 程序2(3.7万) | 318 | 380 | 0 | 537 | 0 | 413 |
| 程序3(1.2万) | 235 | 267 | 0 | 401 | 0 | 386 |

从表1中可以看出,开发的工具比该公司的软件测试工具能发现更多的数据流异常,如变量未初始化、定义(赋值)后未使用以及变量重复定义(赋值)等错误。

## 6 结束语

数据流测试可以在程序运行前尽可能多地发现其中隐含的错误,提高程序的可靠性和健壮性,对于发现源代码中错误使用类型和变量使用等语义问题极其有效。本文提出了基于模型验证的数据流异常检测算法,可以检测出三种变量使用上的异常。由于编译器只能检查语法,无法有效地发现这些异常,因此,本方法可以有效弥补编译器的缺陷。实验证明这种方法是有效的。

相对于文献[6-8,11-12],本文所检测的算法可以检测用变量使用的三种异常。作为编译前的辅助静态分析工具,下面工作除了扩大检测范围,比如参数类型不匹配,指针类型错误等,还要提高检测的精确度,并采用一些算法避免检测空间的爆炸。

## 参考文献:


[1] Clarke E M, Grumberg O, Peled D A.Model checking[M].[S.l.]: The MIT Press, 2000.

[2] Aho A V, Lam M S, Sethi R, et al.Compilers: principles, techniques, and tools[M].2nd ed.[S.l.]: Addison Wesley, 2006.

[3] Huang J C.Detection of data flow anomaly through program instrumentation[J].IEEE Transactions on Software Engineering, 1979, 5(3): 226-236.

[4] 李慧贤,刘坚.数据流分析方法[J].计算机工程与应用,2003,39(13):142-144.

[5] 叶俊民,魏鹏,金聪,等.内存泄漏故障静态分析研究[J].计算机科学,2010,37(6):171-175.

[6] 张威,卢庆龄,万琳,等.空指针引用故障模型与测试方法研究[J].计算机工程与应用,2006,42(4):71-73.

[7] 夏玉辉.变量定义未使用故障的一种静态测试方法[J].计算机工程与设计,2007,28(3):515-516.

[8] 赵鹏宇,万琳,宫云战.未初始化变量的一种静态测试方法[J].计算机工程与设计,2007,28(4):751-754.

[9] Schonberg C, Jaksic M, Weitl F, et al.Verification of Web-content: a case study on technical documentation[R].2009.

[10] Weitl F.Document verification with temporal description logics[D]. Passau: Fakultat fur Informatik and Mathematic University, 2007: 114-145.

[11] 曹文静,宫云战.引用未初始化变量故障静态测试方法研究[J].小型微型计算机系统,2007,28(5):948-951.

[12] 宫云战.一种面向故障的软件测试新方法[J].装甲兵工程学院学报,2004,18(1):21-25.

[13] Bundell G A, Lee G, Morris J, et al.A software component verification tool[C]//Proeeedings of International Conference on Software Methods and Tools, Wollongong, Australia, 2000, 22(11):78-83.