基于函数调用序列模式挖掘的程序缺陷检测

崔展齐[1,2] 牟永敏[1] 张志华[1] 王伟光[3]
(北京信息科技大学计算机学院 北京 100101)[1]
(计算机软件新技术国家重点实验室(南京大学) 南京 210023)[2]
(华为技术有限公司南京研究所 南京 210008)[3]

摘 要 程序中通常会隐含大量编程规则,若在程序编写过程中违反此类规则,则可能引发软件缺陷。函数调用规则是其中一类常见的程序隐含规则,常见的函数调用规则挖掘工作将整个函数体内的函数调用作为一个项集来进行分析,未使用程序中函数调用先后顺序等约束信息,导致软件缺陷挖掘结果的误报率较高。通过简单的静态分析即可获取函数调用序列信息,如在缺陷挖掘过程中充分利用函数调用序列信息,将有效提高缺陷挖掘精度。基于上述思路,提出了一种基于函数调用序列模式挖掘的缺陷检测方法,该方法自动检测程序中违反函数调用序列模式的疑似缺陷,并报告可疑度较高的缺陷。基于该方法,在一组开源项目上进行的实验的结果表明,此方法能有效发现程序中由于违反函数调用序列模式而导致的缺陷,减少了缺陷误报,从而降低了人工核查疑似缺陷开销。

关键词 函数调用序列,序列模式挖掘,缺陷检测
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.034

Defects Detection Based on Mining Function Call Sequence Patterns

CUI Zhan-qi[1,2] MU Yong-min[1] ZHANG Zhi-hua[1] WANG Wei-guang[3]
(School of Computer, Beijing Information Science and Technology University, Beijing 100101, China)[1]
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China)[2]
(Nanjing Research Institute, Huawei Technologies Co., Ltd., Nanjing 210008, China)[3]

Abstract Large scale programs usually imply a large number of programming rules. However, if programmers violate those rules in the process of programming, it is possible to cause software defects. The function call rule is one kind of the typical implicit rules in programs. Previous work on mining function rules handle function calls in the body of a function definition as an itemset, and the constraints implied in function call sequences are not utilized, which can lead to high false positive rates. If the function call sequence information is exploited in the process of mining rules, it will effectively improve the accuracy of mining defects. This paper proposed a defect detection approach based on mining function call sequence patterns. In the approach, the suspected defects which violate function call sequence patterns are detected automatically, and the defects with high suspicious degrees are reported. Based on this approach, experiments were carried out in a group of open source projects. The expriment results show that this approach can effectively find defects which violate function call sequence patterns in programs, and reduce false positives. As a result, the overhead of verifying suspicious defects are also reduced.

Keywords Function call sequence, Sequence pattern mining, Defects detection

1 引言

大规模程序开发过程中通常隐含大量编程规则,但由于此类规则数量众多且受到开发时间和进度的限制,软件工程师很少使用规范的文档来描述这类规则。然而,若程序员在编程过程中忽视或违反了该类规则,则可能会引发软件缺陷。软件缺陷挖掘是自动识别程序隐含规则的有效手段,其通过对软件代码、文档等相关数据进行建模来识别隐含的缺陷模式或编程规则,并据此自动发现软件缺陷[1]。软件缺陷挖掘能够较大程度地实现自动化,使得人力成本开销较小,近年来其在大规模软件上取得了一系列成果[2-6],越来越受到工业界的重视。

到稿日期:2016-10-18 返修日期:2016-12-21 本文受国家重点研发计划(2016YFC0801407),计算机软件新技术国家重点实验室开放课题(KFKT2016B12),北京信息科技大学学校科研基金(1625008),计算机学院大类人才培养模式改革项目(5111623409)资助。
崔展齐(1984-),男,博士,讲师,主要研究方向为软件测试及缺陷检测技术,E-mail:czq@bistu.edu.cn;**牟永敏**(1961-),男,博士,教授,主要研究方向为软件理论与应用;**张志华**(1971-),女,硕士,副教授,主要研究方向为软件测试技术;**王伟光**(1984-),男,博士,工程师,主要研究方向为信息安全及软件测试技术。

其中，函数调用模式是一类程序中隐含的最为常见的规则，如：在 C 程序中，malloc()和 free()必须成对出现，否则可能会引发内存泄漏。在软件缺陷挖掘工作中，常见的规则如 PR-Miner[3]，仅将函数内的变量使用和函数调用作为一个项集来进行分析，未考虑函数调用位置所包含的顺序约束，如 malloc()和 free()不仅应该同时出现，且 free()应该出现在 malloc()之后，否则可能会引发缺陷。未考虑函数调用的顺序约束将导致软件缺陷挖掘结果不够精确，从而导致较高的误报率。因此，通过简单静态分析来获取函数调用在源程序中的先后顺序关系，如在缺陷挖掘过程中能够充分利用函数调用顺序信息，将有效提高缺陷挖掘精度。

基于上述思路，本文提出了一种基于函数调用序列模式挖掘的程序缺陷检测方法。首先，分析程序结构，依次为每个函数定义并生成其内部函数调用序列；其次，基于函数调用序列集挖掘程序中隐含的函数调用序列模式；最后，检测程序中违反函数调用序列模式且可疑度较高的疑似缺陷。本文在一组开源项目上进行的实验结果表明，所提方法能有效发现程序中由于违反函数调用序列模式而导致的缺陷，降低了误报的疑似缺陷数。

本文的主要贡献在于：1)提出了一种基于函数调用序列模式挖掘的缺陷检测方法，结合程序结构信息和数据挖掘技术，自动检测程序中由于违反函数调用序列模式而隐含的缺陷；2)基于该方法实现了一个通过分析函数调用模式来检测缺陷的原型工具，并在一组真实的开源系统源代码上进行了实例研究，以评估所提方法的有效性。

本文第 2 节详细介绍基于函数调用序列模式的挖掘及缺陷检测方法；第 3 节介绍原型工具的实现并进行实验和评估；第 4 节介绍与本文相关的工作；第 5 节总结本文并展望未来。

2　基于函数调用模式挖掘的缺陷检测

频繁项集挖掘(Frequent Itemset Mining)是一种常用的数据挖掘方法[7-8]。一个典型的函数中往往含有多处函数调用语句，在函数调用频繁模式和关联规则挖掘方法中，通常将一个函数内的所有函数调用作为一个事务，在程序所有函数定义所组成的事务集上使用 Apriori[9]，FP-Growth[10]等算法进行频繁模式(Frequent Pattern)挖掘，以计算满足支持度(Support)和置信度(Confidence)的函数调用关联规则，再找出违反上述规则的位置，并将其作为疑似缺陷来进行分析和确认。由于在频繁模式挖掘过程中将函数调用作为项集进行处理，丢失了程序中函数调用的先后顺序关系信息，因此对缺陷检测的精度造成了影响。在实际的编程中发现，相互关联的函数通常对调用顺序有特殊要求。一方面，若一组函数调用组成的频繁项集在程序中对应的调用顺序差异较大，不具一定规律，则可能是误报(False Positive)；另一方面，频繁项集挖掘技术不能检测出违反调用序列约束的缺陷，从而导致漏报(False Negative)。

针对上述问题，本文通过分析函数调用序列集挖掘程序中隐含的函数调用序列关联模式，来检测程序中违反上述模式的疑似缺陷，具体流程如图 1 所示。

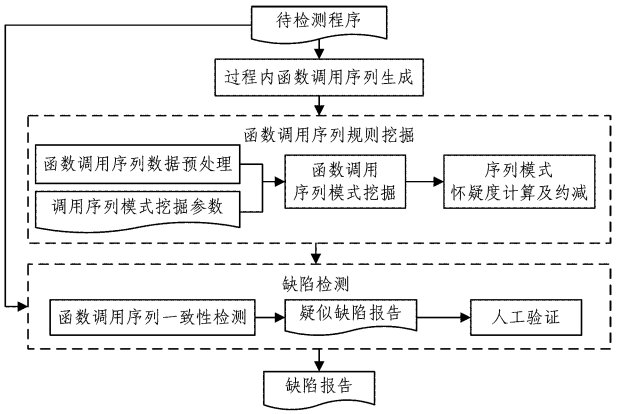

图 1　基于函数调用序列模式挖掘的缺陷检测流程图

2.1　函数调用序列分析

函数定义(Function Definition)是程序(Program)的基本功能模块，表示为 $P=\{fd_1, fd_2, \cdots, fd_n\}$。为完成其预定功能，函数内部需要调用其他函数，在函数 fd_i 中以函数调用(Funcation Call)为基本单位的调用序列$\langle fc_1, fc_2, \cdots, fc_m\rangle$可在一定程度上描述源程序控制流中函数调用位置间的逻辑关系。通过静态分析技术可获取程序的控制流、数据流、指针分析等较为丰富和精确的语义信息[11-12]。为简化处理，本文不考虑分支、循环等控制结构，仅分析从函数入口开始到函数出口结束，函数体内函数调用语句出现的位置间的先后顺序关系。

为获取更一般的源程序表示及简化分析过程，本文扩展了开源的 C 语言分析器 pycparser[1)]，以获取程序抽象语法树(Abastract Semantic Tree，AST)，并在其基础上生成函数调用序列。在分析函数调用序列的同时，记录函数调用对应的源代码位置信息，以在后续缺陷检测时快速定位。函数调用序列生成算法如算法 1 所示。

算法 1　函数调用序列生成算法

输入：程序 P={fd_1, fd_2, …, fd_n}

输出：函数调用序列集 CS={cs_1, cs_2, …, cs_n}

```
1. for fd_i in P//遍历程序中的每个函数
2.    ast=pycparser.getAST(fd_i)//获取函数的抽象语法树
3.    for node in ast//顺序遍历 ast 中所有节点
4.       if(node.type==AST.FuncationCall)//若节点为函数调用
5.          cs_i.append(node)//将 node 加入 fd_i 对应的函数调用序列中
6.    CS.add(cs_i) //将 cs_i 加入函数调用序列集
7. return CS
```

2.2　函数调用序列模式挖掘

序列数据由有序元素或事件组成，设 $I=\{i_1, i_2, \cdots, i_o\}$是所有项的集合，序列模式挖掘就是找出序列数据库中所有的频繁序列模式，即支持度不小于用户给定支持度阈值的所有序列[13]。函数调用关联规则挖掘时待检测程序中的每条程序调用语句即为项。项集是由 I 中若干元素组成的集合，一个序列是由若干项集组成的有序列表$\langle is_1, is_2, \cdots, is_p\rangle$，其中项集 is_i 是由不同项组成的集合，可表示为$(x_1, x_2, \cdots, x_q)$；序列中的项集是顺序排列的，即当 $i<j$ 时，is_i 发生在 is_j 之前，而项集内部的元素是无序的。在函数调用序列中，同时出现的函数调用语句只有一条，即序列中项集只包括一个元素，可

1) https://github.com/eliben/pycparser

去掉括号，如(x_l)可直接表示为x_l。称序列中包含项的个数为序列的长度，长度为K的序列记为K-序列。

序列数据集S是二元组(Sid, s)的集合，Sid是序列编号，s是序列。若序列t中的每个项集都是序列s中一个项集的子集，则称t是s的子序列，s包含t。一个序列s的支持度计数(supCount)指在整个序列数据集中包含s的序列数，给定一个最小支持度阈值$minSupCount$，若序列s的支持度计数不小于$minSupCount$，则称序列s为数据集S中的频繁序列或序列模式，S中所有序列模式的集合为FS。

GSP算法[14]在AprioriAll算法[13]的基础上增加了时间约束，包括连续事件的最大/最小时间间隔(maxGap/minGap)等，即当两个事件间隔的事件数大于$maxGap$或小于$minGap$时，不认为它们在同一序列中。本文使用GSP算法挖掘函数调用序列模式，并根据支持度对候选函数调用序列模式进行删选。在函数调用序列模式的分析中，当两个函数调用语句位置的间隔较远时，其存在关联的可能性较低，因此可通过设置合适的$maxGap$来优化生成的频繁序列模式集。

2.3 缺陷检测

在生成频繁序列模式集FS后，对源程序进行分析，以检测程序中函数调用顺序违反序列模式的位置，从而构成疑似缺陷集。

长度为k且序列中所有项集都只包含一个元素的序列模式$\alpha=\langle is_1, is_2, \cdots, is_u\rangle$，$\alpha$的最大前缀是长度为$k-1$的序列$\beta=\langle is_1', is_2', \cdots, is_{u-1}'\rangle$，其中$is_j=is_j'$。若函数定义$fd_i$对应的函数调用序列$\gamma$包含$\beta$但不包含$\alpha$，则称$fd_i$存在违反序列模式$\alpha$的疑似缺陷，其怀疑度(suspicious)为：

$$suspicious(fd_i,\alpha,\beta)=\begin{cases}\dfrac{supCount(\alpha)}{supCount(\beta)}, & supCount(\beta)>supCount(\alpha)\\ 0, & supCount(\beta)=supCount(\alpha)\end{cases}$$

为合理控制缺陷检测开销，需从两个角度对序列模式和疑似缺陷进行约减：1)在挖掘序列模式时设置最小支持度$minSup$，不采用支持度过低的序列模式，即不关注在函数路径集中出现次数过少的函数调用序列，因为程序员在处理使用次数较少的函数调用时，在使用模式上出现错误的可能性较低；2)引入缺陷怀疑度，即假设程序员在大多数情况下对函数使用的顺序是正确的，只是偶尔会犯错误，因此只将怀疑度超过最小怀疑度($minSus$)的缺陷作为疑似缺陷。

对源程序进行扫描以查找疑似缺陷，并根据频繁序列集检查违反频繁序列模式的函数定义。对于长度为k的序列模式k-fs_j，找出对应函数调用序列中不包含该序列模式但包含该序列模式最大前缀序列$(k-1)$-fs_j的函数定义，若该缺陷的怀疑度大于$minSus$，即为疑似缺陷。扫描源程序查找疑似缺陷的算法如算法2所示。最后，需要人工确认找到的所有疑似缺陷是否为真实缺陷。

算法2 使用序列模式查找疑似缺陷

```
输入：函数调用序列集 CS，频繁序列模式集 FS
输出：bugList//疑似缺陷列表
1. for fcs_i in FS//遍历频繁序列模式
2. β_i = fcs_i 的最大前缀
3.   for cs_i in CS//遍历所有函数序列
4.       if (cs_i 包含 β_i)//序列 cs_i 不包含序列模式的 fcs_i 最大子集
5.         supCount_β_i++//β_i 支持度递增
6.         if (cs_i 不包含 fcs_i)//序列 cs_i 不包含序列模式 fcs_i
7.               tempbugList.add(cs_i)//加入到疑似 bug 临时列表
8.         else
9.               supCount_fcs_i++//fcs_i 支持度递增
10.  if(minSus < supCount_β_i/supCount_fcs_i<1)
11.      //若怀疑度大于最小怀疑度
12.      bugList.add(tempbugList)//将疑似 bug 临时列表加入 bugList
13.      tempbugList.clear()//清空疑似 bug 临时列表
14. return bugList
```

3 实验和评估

3.1 原型工具实现

在上述程序调用序列模式挖掘的缺陷检测技术的基础上实现了一项原型工具。实验和开发该工具的硬件环境为Intel i7 2.3GHz处理器、8GB内存，软件环境为Ubuntu16.04、Python3.4。原型工具主要由两个模块组成：1)函数调用序列生成模块，其生成函数体内的函数调用语句序列，并将生成的序列经预处理后使用数据挖掘工具Rapidminer 7.2.1[1)]进行序列模式挖掘；2)缺陷检测模块，其根据挖掘出的函数序列模式扫描程序来识别违反函数调用序列模式的位置，从而构成疑似缺陷报告。

3.2 实验设计

为评估使用函数调用序列模式检测缺陷技术的有效性，本文实现了基于频繁项集和关联规则挖掘的缺陷检测方法的对比实验。频繁项集和关联规则挖掘方法不使用函数调用语句的顺序信息，而是将整个函数中所有函数调用语句作为一个项集来进行分析。本文将两种方法在同一组开源项目上进行实验，从计算时间、挖掘规则情况和缺陷验证情况3个方面进行比较。作为实验对象的开源项目包括：内存数据库Redis[2)]、跨平台脚本语言Lua[3)]、嵌入式系统数据库Sqlite[4)]，其基本情况如表1所列。

表1 实验项目的基本情况

项目	.c文件数	代码行数(不含空行和注释)	函数定义数	函数调用次数	去重后的函数内调用数
Redis	70	43k	1712	10557	1519
Lua	35	15k	896	2747	663
Sqlite	2	126k	1958	11586	1645

1) https://rapidminer.com

2) http://redis.io

3) http://www.lua.org

4) http://www.sqlite.org

3.3　实验结果分析

实验中函数调用频繁项集和关联规则挖掘方法使用的参数为：最小支持度 $minSup=0.01$，最小置信度 $minConf=0.9$；函数调用序列模式挖掘方法使用的参数为：最小支持度 $minSup=0.01$，最小怀疑度 $minSus=0.9$，最大时间间隔 $maxGap=10$。

表 2 列出了两种方法计算时间开销的结果比较，其中未对使用 Rapidminer 进行关联规则或序列模式挖掘的时间进行统计。从表 2 可知，函数调用序列模式挖掘方法所需的计算时间开销大于关联规则挖掘方法。实验中数据预处理、规则挖掘和疑似缺陷检测是完全自动化的，并不会增加额外的人力成本开销。

表 2　时间开销的结果比较

项目	挖掘方法	函数调用集或调用序列生成时间/ms	疑似缺陷检测时间（不含人工验证时间）/ms
Redis	关联规则	21646.0	75.8
	序列模式	22964.4	222.4
Lua	关联规则	7149.4	30.0
	序列模式	7567.8	40.4
Sqlite	关联规则	10338.8	137.6
	序列模式	11792.2	496.2

表 3 列出了函数调用关联规则和序列模式两种方法挖掘出的有效规则和检测到的疑似缺陷情况。表 3 中，关联规则挖掘方法报告的有效规则数是置信度大于 *minConf* 且小于 1 的关联规则数，序列模式挖掘方法报告的有效规则数是指能发现怀疑度大于 *minSus* 的疑似缺陷的序列模式数。由表 3 可知，序列模式挖掘方法报告的有效规则和疑似缺陷数明显减少，有效减少了缺陷误报。因此，该方法能够有效降低人工确认疑似缺陷的开销。

表 3　规则挖掘及疑似缺陷报告情况

项目	规则挖掘方法	有效规则数	疑似缺陷数
Redis	关联规则	19	38
	序列模式	8	16
Lua	关联规则	2	2
	序列模式	1	1
Sqlite	关联规则	70	111
	序列模式	2	9

如：Redis 项目 t_ set. c 文件中的第 135－139 行 setTypeReleaseIterator 函数的定义如下：

```
135 voidsetTypeReleaseIterator(setTypeIterator * si) {
136    if (si->encoding == OBJ_ENCODING_HT)
137        dictReleaseIterator(si->di);
138    zfree(si);
139 }
```

若采用频繁项集挖掘方法，对于挖掘出的关联规则[dictReleaseIterator()=T]->[dictNext()=T]，由于函数未调用 dictNext()方法，因而认为上述函数存在疑似缺陷，但实际上该函数的功能正是为了释放 Iterator，因此该缺陷为误报。若采用函数调用序列挖掘方法，报告的相关函数调用的序列模式为〈dictNext()，dictReleaseIterator()〉和〈dictGet Iterator()，dictNext()，dictReleaseIterator()〉，该函数并未违反上述函数调用序列模式，因此不会作为疑似缺陷报告，从而减少了误报。

为验证序列模式挖掘方法是否会降低缺陷发现能力，表 4 列出了疑似缺陷的人工确认情况。由于缺少相关系统的领域知识，本文将人工确认的结果分为确认为真实缺陷、确认为误报和暂时还不能确认是否为真实缺陷 3 类。实验结果表明，序列模式挖掘方法在报告的疑似缺陷数量明显减少的前提下，能与关联规则挖掘算法发现同样多的真实缺陷，而确认为误报和暂不能确认是否为真实缺陷的疑似缺陷数则明显减少。

表 4　缺陷确认情况

项目	规则挖掘方法	确认缺陷数	确认误报数	不确定缺陷数
Redis	关联规则	1	24	13
	序列模式	1	14	1
Lua	关联规则	0	2	0
	序列模式	0	1	0
Sqlite	关联规则	0	88	23
	序列模式	0	9	0

此外，为验证序列模式挖掘方法能够发现函数调用顺序错误引发的缺陷，由于关联规则挖掘方法不能发现这类缺陷，本文使用错误注入的方式在 3 个项目中分别注入多个与函数调用顺序相关的错误。

图 2 给出了错误注入方式。如：在 Redis 项目的 redis-cli. c 文件中根据函数 evalMode()复制一个新函数，并调换函数调用语句 fopen()和 fclose()的位置以注入错误；又如：在 Redis 项目的 sentinel. c 文件中，根据函数 sentinelSelectSlave()复制一个新函数，并调换函数调用语句 dictReleaseIterator()和 dictNext()的位置以注入错误。

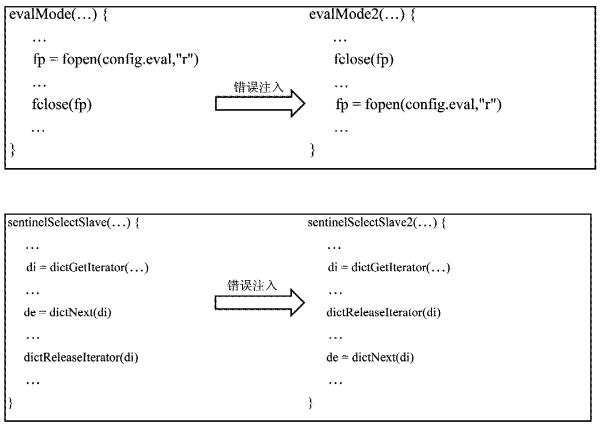

图 2　错误注入示意图

表 5 列出了注入错误数及两种方法的缺陷检测情况。实验结果表明，当 *minConf* 和 *minSus* 为 0.9 时，序列模式挖掘算法能检测出其中 4 个错误，检测率达 67%，而关联规则挖掘方法则不能检测出注入的错误。对未检测到的错误进行分析后发现，由于在注入该错误后，相关序列模式检测到的疑似缺陷怀疑度不足 0.9，因此未将其作为疑似缺陷报告。若将 *minConf* 和 *minSus* 调整为 0.8，则序列模式挖掘算法能检测出所有的注入错误，而关联规则挖掘方法仍不能检测出相关的注入错误。因此，本文提出的序列模式挖掘算法能够有效检测函数调用顺序错误引发的缺陷，而关联规则挖掘方法则不能识别此类缺陷。如：在 Redis 项目中，根据序列模式〈dict-Get-Iterator()，dictNext()，dictReleaseIterator()〉和〈fopen()，

fclose()〉能分别识别出图 2 中的两个注入错误。若采用关联规则挖掘方法，含有错误的函数将分别满足频繁项集{dictReleaseIterator(),dictNext(),dictReleaseIterator()}和{fopen(),fclose()}生成的关联规则。因此，关联规则挖掘方法并不能识别此类与函数调用顺序相关的缺陷。

表 5 注入错误及检测情况

项目	注入错误数	规则挖掘方法	检测注入错误数(*minCon*/*minSusf*=0.9)	检测注入错误数(*minCon*/*minSusf*=0.8)
Redis	3	关联规则	0	0
		序列模式	2	3
Lua	1	关联规则	0	0
		序列模式	0	1
Sqlite	2	关联规则	0	0
		序列模式	2	2

4 相关工作

使用数据挖掘技术分析程序隐含规则并将其用于缺陷检测是软件工程与机器学习领域相结合的研究热点之一。

AutoISES[15]分析程序中的安全规约，以此来检查系统中的安全敏感操作是否受到相关安全检查函数的保护，从而识别源程序中的安全漏洞。iComment[16]使用数据挖掘技术从程序注释中分析与锁和中断相关的前置和后置条件并进行标注，以检查源程序与中断和同步相关的缺陷。aComment[17]在 iComment 的基础上增加了源程序信息来挖掘规约，以提高缺陷检测精度。CR-Miner[2]识别源程序中由于复制粘贴代码片段，但未一致更新而引发的缺陷。MUVI[4]和于秀梅等人[18]通过挖掘变量间的访问规则来检测程序中的不一致缺陷，其中前者使用路径不敏感的方式，而后者使用路径信息来提高缺陷检测精度。MAPO[5]通过分析开源程序库，挖掘库函数 API 的使用方法并提供查询接口，来为程序员使用库函数提供参考。Engler 等人[19]设计了一系列规则模板，通过分析程序提取 MUST 和 MAY 的两类规则，来检查程序中违反规则模板的缺陷。Maffort 等人[20]使用关联模式挖掘的方法从程序中挖掘程序的体系结构模式，并识别程序中违反体系结构模式的缺陷。

本文所提方法使用了程序中函数调用语句间的顺序关系，基于序列模式挖掘算法检测程序中与函数调用序列相关的疑似缺陷。PR-Miner[3]是与本文较为相似的工作，该方法提取程序中经常一起出现的元素，使用频繁项集挖掘技术将其作为一个项集来挖掘元素间的关联规则，并支持变量和函数间关联规则的识别。本文与该方法的主要区别在于本文在程序隐含规则的挖掘过程中引入了函数调用间的顺序关系这一约束信息，通过挖掘函数调用序列模式来提高缺陷检测能力以及减少误报，但本文方法未关注与变量关联规则相关的程序缺陷。

结束语 针对基于函数调用关联规则挖掘的缺陷检测方法未使用程序中函数调用先后顺序信息，从而导致缺陷挖掘结果误报较多的问题，本文提出了一种基于函数调用序列模式挖掘的缺陷检测方法，该方法自动检测程序中与函数调用模式相关且可疑度较高的疑似缺陷。基于此方法，在一组开源项目上进行实验，结果表明，上述方法能够有效发现程序中由于违反函数调用序列模式而导致的缺陷，减少了疑似缺陷误报数，降低了人工验证疑似缺陷的开销。

本文是关于使用程序结构及静态分析结果信息辅助程序缺陷挖掘研究的初步结果。在本文研究的基础上，我们计划通过使用控制流、数据流等更多静态分析结果信息和更加有针对性的缺陷挖掘算法，来进一步提高缺陷挖掘的精度。

参 考 文 献

[1] LI M, HUO X. Software Defect Mining Based on Semi-supervised Learning[J]. Journal of Data Acquisition and Processing, 2016, 31(1): 56-64. (in Chinese)
黎铭，霍轩. 半监督软件缺陷挖掘研究综述[J]. 数据采集与处理，2016，31(1)：56-64.

[2] LI Z, LU S, MYAGMAR S, et al. CP-Miner: a Tool for Finding Copy-Paste and Related Bugs in Operating System Code[C]// Conference on Symposium on Opearting Systems Design & Implementation. 2004: 289-302.

[3] LI Z M, ZHOU Y Y. PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code[C]// European Software Engineering Conference Held Jointly with, ACM Sigsoft International Symposium on Foundations of Software Engineering. Lisbon, Portugal, 2005: 306-315.

[4] LU S, PARK S, HU C, et al. MUVI: Automatically Inferring Multi-Variable Access Correlations and Detecting Related Semantic and Concurrency Bugs[C]// ACM Symposium on Operating Systems Principles 2007 (SOSP 2007). Stevenson, Washington, USA, 2010: 103-116.

[5] XIE T, PEI J. MAPO: Mining API Usages from Open Source Repositories[C]// International Workshop on Mining Software Repositories (MSR 2006). Shanghai, China, 2006: 54-57.

[6] QU W, JIA Y, JIANG M. Pattern Mining of Cloned Codes in Software Systems[J]. Information Sciences, 2014, 259(3): 544-554.

[7] HAN J, PEI J, KAMBER M. Data Mining: Concepts and Techniques[M]. Elsevier, 2011.

[8] HARRINGTON P. Machine Learning in Action[M]. Greenwich, CT: Manning, 2012.

[9] AGRAWAL R, SRIKANT R. Fast Algorithms for Mining Association Rules in Large Databases[C]// International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., 1994: 487-499.

[10] HAN J. Mining Frequent Patterns without Candidate Generation[J]. ACM Sigmod Record, 2000, 29(2): 1-12.

[11] YOUNG M, PEZZE M. Software Testing and Analysis: Process, Principles and Techniques[M]. John Wiley & Sons, 2007.

[12] MEI H, WANG Q X, ZHANG L, et al. Software Analysis : a Road Map[J]. Chinese Journal of Computers, 2009, 32(9): 1697-1710. (in Chinese)
梅宏，王千祥，张路，等. 软件分析技术进展[J]. 计算机学报，2009，32(9)：1697-1710.

[13] AGRAWAL R,SRIKANT R. Mining Sequential Patterns[C]// IEEE 29th International Conference on Data Engineering. 1995: 3-14.

[14] SRIKANT R,AGRAWAL R. Mining Sequential Patterns:Generalizations and Performance Improvements[C]// International Conference on Extending Database Technology:Advances in Database Technology. Springer-Verlag,1996:1-17.

[15] TAN L,ZHANG X,MA X,et al. AutoISES:Automatically Inferring Security Specifications and Detecting Violations[C]// Conference on Security Symposium. 2008:379-394.

[16] TAN L,YUAN D,KRISHNA G,et al. /* iComment:Bugs or Bad Comments? */[C]// ACM Symposium on Operating Systems Principles 2007 (SOSP 2007). Stevenson, Washington, USA,2007:145-158.

[17] TAN L,ZHOU Y,PADIOLEAU Y. aComment:Mining Annotations from Comments and Code to Detect Interrupt Related Concurrency Bugs[C]// Proceeding of International Conference on Software Engineering. 2011:11-20.

[18] YU X M,LIANG B,CHEN H,et al. Path-Sensitive Multi-Variable Access Correlations Mining and Related Source Code Defects Detection[J]. Pattern Recognition and Artificial Intelligence, 2012,25(4):691-698. (in Chinese)
于秀梅,梁彬,陈红,等. 路径敏感的源码关联变量模式挖掘及缺陷检测[J]. 模式识别与人工智能,2012,25(4):691-698.

[19] ENGLER D,CHEN D Y,HALLEM S,et al. Bugs as Deviant Behavior:a General Approach to Inferring Errors in Systems Code[J]. ACM Sigops Operating Systems Review,2010,35(5): 57-72.

[20] MAFFORT C,VALENTE M T,BIGONHA M,et al. Mining Architectural Patterns Using Association Rules[C]// International Conference on Software Engineering and Knowledge Engineering. 2013:375-380.

(上接第 220 页)

[28] MOLINARI D H,STAMBAUGH M A,CAIN P J. Apparatus for testing cellular base stations:U. S. Patent 6,308,065[P]. 2001.

[29] AMALFITANO D,FASOLINO A R,TRAMONTANA P. A gui crawling-based technique for android mobile application testing [C]// 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE,2011:252-261.

[30] WANG P,LIANG B,YOU W,et al. Automatic Android GUI Traversal with High Coverage[C]// 2014 Fourth International Conference on Communication Systems and Network Technologies (CSNT). IEEE,2014:1161-1166.

[31] CHUN B G,IHM S,MANIATIS P,et al. Clonecloud:elastic execution between mobile device and cloud[C]// Proceedings of the Sixth Conference on Computer Systems. ACM, 2011: 301-314.

[32] ENCK W,GILBERT P,HAN S,et al. TaintDroid:an information-flow tracking system for realtime privacy monitoring on smartphones[J]. ACM Transactions on Computer Systems (TOCS),2014,32(2):1-29.

[33] FELT A P,CHIN E,HANNA S,et al. Android permissions demystified[C]// Proceedings of the 18th ACM Conference on Computer and Communications Security. ACM,2011:627-638.

[34] ANAND S,NAIK M,HARROLD M J,et al. Automated concolic testing of smartphone apps[C]// Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM,2012:1-11.

[35] CHOI W,NECULA G,SEN K. Guided gui testing of android apps with minimal restart and approximate learning[J]. ACM SIGPLAN Notices,2013,48(10):623-640.

[36] MIRZAEI N,MALEK S,PĂSĂREANU C S,et al. Testing Android apps through symbolic execution[J]. ACM SIGSOFT Software Engineering Notes,2012,37(6):1-5.

[37] VAN DER MERWE H,VAN DER MERWE B,VSSER W. Verifying android applications using Java PathFinder[J]. ACM SIGSOFT Software Engineering Notes,2012,37(6):1-5.

[38] JESUS F. Smali,an assembler/disassembler for Android's dex format[OL]. http://code.google.com/p/smali.

[39] ZHENG M,LEE P P C,LUI J C S. ADAM:an automatic and extensible platform to stress test android anti-virus systems[M]// Detection of Intrusions and Malware,and Vulnerability Assessment. Springer Berlin Heidelberg,2013:82-101.

[40] APVRILLE A,STRAZZERE T. Reducing the Window of Opportunity for Android Malware Gotta catch'em all[J]. Journal in Computer Virology,2012,8(1/2):61-71.

[41] SPREITZENBARTH M,FREILING F,ECHTLER F,et al. Mobile-sandbox:Having a deeper look into android applications [C]// Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM,2013:1808-1815.

[42] BATYUK L,HERPICH M,CAMTEPE S A,et al. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications[C]// 2011 6th International Conference on Malicious and Unwanted Software (MALWARE). IEEE,2011:66-72.

[43] ZHENG C,ZHU S,DAI S,et al. Smartdroid:an automatic system for revealing ui-based trigger conditions in android applications[C]// Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. ACM,2012: 93-104.

[44] KOVACHEVA A. Efficient Code Obfuscation for Android [M]// Advances in Information Technology. Springer International Publishing,2013:104-119.

[45] BERTHOME P,FECHEROLLE T,GUILLOTEAU N,et al. Repackaging android applications for auditing access to private data[C]// 2012 Seventh International Conference on Availability,Reliability and Security (ARES). IEEE,2012:388-396.

[46] STEVENS R,GIBLER C,CRUSSELL J, et al. Investigating user privacy in android ad libraries[C]// Workshop on Mobile Security Technologies (MoST). 2012.