软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(3):408−416[doi: 10.3724/SP.J.1001.2011.03931]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

软件库调用规约挖掘*

钟 浩[1+], 张 路[2,3], 梅 宏[2,3]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100190)

[2](北京大学 信息科学技术学院 软件研究所,北京 100871)

[3](高可信软件技术教育部重点实验室(北京大学),北京 100871)

Mining Invocation Specifications for API Libraries

ZHONG Hao[1+], ZHANG Lu[2,3], MEI Hong[2,3]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[3](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

+ Corresponding author: E-mail: zhonghao@itechs.iscas.ac.cn

Zhong H, Zhang L, Mei H. Mining invocation specifications for API libraries. *Journal of Software*, 2011,22(3): 408−416. http://www.jos.org.cn/1000-9825/3931.htm

Abstract: Invocation Specifications of an API library are types of specifications that are able to describe the legal invocation sequences of methods. Client code should follow descriptions of invocation specifications to call upon methods provided by API libraries. If this procedure is not followed, defects can be introduced into client code, and reduce its integrity. Because invocation specifications define the properties that are trustworthy for a software system, they play a central role in the research of trustworthy software and model checking; however, due to the great efforts to write invocation specifications, most API libraries do not provide such specifications. To address the problem, researchers have proposed various approaches to mine invocation specifications automatically. In this paper update-to-date research of mining specifications is surveyed, and the potential direction of further research is discussed.

Key words: mining specifications; API library

摘 要: 软件库调用规约是一种描述软件库提供函数正确调用顺序的规约.客户代码应按此规约描述的内容调用函数,否则可能引入缺陷,从而降低软件的可信性.由于能够描述可信软件应该满足的性质,软件库调用规约在可信软件、模型检测等研究中扮演特殊的角色.但是,受制于编写规约的巨大代价,软件库通常并不提供已编写好的调用规约.为此,研究者提出了各种自动挖掘此种规约的方法.阐述了其中代表性的方法及其最新的研究进展,并在此基础上探讨了将来的研究方向.

关键词: 挖掘规约;软件库

---

* 基金项目: 国家自然科学基金(90718042, 90718016); 国家重点基础研究发展计划(973)(2007CB310802, 2009CB320703); 国家高技术研究发展计划(863) (2007AA010303, 2007AA010301)

本文根据第一作者在北京大学攻读博士学位期间所从事的软件库函数调用规约挖掘的研究工作基础上撰写而成.第一作者的现单位为中科院软件所互联网软件实验室.

收稿时间: 2010-03-04; 修改时间: 2010-04-27; 定稿时间: 2010-07-06

CNKI 网络优先出版: 2010-11-05 11:50, http://www.cnki.net/kcms/detail/11-2560.TP.20101105.1150.002.html

中图法分类号: TP311 文献标识码: A

1 调用规约挖掘的背景与意义

近年来,可信软件日益得到学术界和工业界的重视.软件库的调用规约能够形式化地描述其函数的正确调用顺序,在可信软件中扮演重要的角色.但是,受编写此种规约的代价等因素影响,一般软件库并不提供完整的形式化的调用规约.为此,研究者提出了各种方法自动地挖掘此种规约.这些研究与模型检测[1]和软件分析[2]的研究有着紧密的联系.其中,模型检测和软件分析着力于检查可信的软件系统是否满足某一性质,而软件库调用规约挖掘则着力于挖掘可信的软件系统应该采用什么样的规则调用软件库提供的函数.图 1 展示了软件库调用规约所扮演的角色.其中,J2SE 等软件库提供了大量可以复用的函数;程序员编写客户代码调用软件库提供的函数以完成某个编程任务;调用规约则描述客户代码应该按照什么样的规则调用软件库提供的函数.

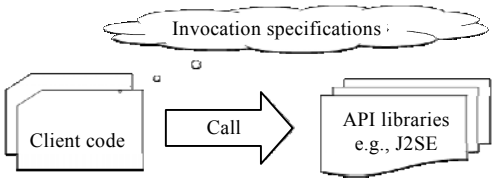


Fig.1 Role of specifications

图 1 调用规约扮演的角色

除了挖掘调用规约之外,也有研究者[3]研究如何挖掘程序中变量的合法取值范围,即不变式.不变式与调用规约存在互补的关系,它们分别描述了不同种类的调用规则.程序员在使用软件库时,需要同时满足不变式和调用规约描述的要求.Lorenzoli 等人[4]探索过将挖掘调用规约与挖掘不变式结合起来,他们提出的方法能够挖掘同时具有上述两种规约特性的规约.但除此之外,已有的挖掘不变式的研究与挖掘调用规约的研究都相对独立.本文仅关注与调用规约的有关研究,这些研究的目标是探索各种挖掘调用规约的自动化方法.其中,最早的一种方法可以追溯到 1998 年 Cook 和 Wolf[5]发表在《ACM Trans. on Software Engineering and Methodology》上的论文.在本文中,我们力求从输入、表示、算法和用途角度阐述调用规约挖掘的研究现状,并在此基础上提出此研究方向的存在的问题和未来发展的方向.

2 调用规约挖掘的输入

调用规约挖掘的输入主要如下:

(1) 客户代码

随着开源软件的飞速发展,开源社区提供了成千上万行可以免费获得的代码.这些代码中,很多是使用软件库的客户代码.由于客户代码展示了使用软件库的细节,通过分析客户代码可以挖掘到软件库的规约.客户代码使用软件库的细节可以通过静态分析的方法得到,也可以通过动态执行的方法得到.这两种方法各有优劣:静态分析的方法能够用比较小的代价获取软件库的使用信息,获取到的使用信息也比较全面.但在获取信息的精度上,静态分析的方法一般弱于动态分析的方法.一些需要实际执行才能获取的结果(如多态),用静态分析只能得到近似的结果.由于客户代码易于获得,从客户代码挖掘规约是目前的主流途径[5−33].

从客户代码获取的软件库使用信息一般为软件库函数的调用序列.目前,无论采用动态还是静态方法,都很难完全准确获得函数的调用序列.分析此信息所得具体技术,在相当程度上决定了获得的调用序列的可靠性,从而也决定了挖掘到的调用规约的准确度.研究者在调用序列的获取上也有一些探索.例如,Yang 等人[6]认为,一个软件库函数(如 m_1)的内部可能调用其他软件库函数(如 m_2),而这种调用关系造成分析获得的调用序列里大量出现 $m_1 \rightarrow m_2$ 的调用对.由于并不反映实际的软件库函数调用规则,此调用对于挖掘的结果有负面的影响.因此,他们提出的方法分析了函数间的调用关系,并将分析结果用于过滤挖掘到的候选规约.又如,Ammons 等人[7]将数据间的依赖关系用于分析函数的调用顺序.他们提出的方法首先动态执行客户代码;从获得的软件库调用序列中,他们仅选取存在数据依赖关系的片段用于挖掘.再如,Chen 等人[8]发现,在客户代码运行中,一个类可能有多个实例.如果通过动态方法获取函数调用序列,多个实例的函数调用可能混杂在一起,从而影响分析的结果.

为此,他们提出了相应的处理技术,以区分不同实例的调用.

另外,虽然客户代码较为丰富,但仍然不能覆盖某些不太常用软件库或者软件库的不太常用的一些函数.Thummalapenta 和 Xie[34]提出了一种可以自动分析软件库热点和冷点的方法.这里,热点指的是客户代码较多的软件库函数/类,冷点指的是客户代码较少的软件库函数/类.他们分析了 8 个常见的软件库,结果表明,所有这些软件库的冷点都多于热点,这些冷点不具备足够的客户代码用于挖掘规约.为了处理冷点软件库和冷点函数,有些研究者探索了用随机生成的测试用例代替客户代码以挖掘规约的途径[27−29].这些研究的缺点在于,随机生成的测试用例难以反映真实的软件库使用场景.为此,也有研究者试图建立较大规模的客户代码库,或者利用现有的大规模客户代码库来挖掘规约.例如,Google 公司搜集了开源社区的代码,建立了非常庞大的代码库.通过 Google 代码搜索(http://www.google.cn/codesearch),可以访问到这个库的内容.Thummalapenta 和 Xie[9]提出的 ParseWeb 工具用 Google 代码搜索返回的代码挖掘规约.此工具能够挖掘到质量更好的规约,其中一个重要原因是用于挖掘的客户代码规模相对较大.

(2) 注释和文档

自然语言描写的注释和文档包含了丰富的与软件使用规则有关的内容,分析注释和文档的内容也可以挖掘软件库规约.

注释和文档中可能包含显示描述规则的句子,例如在 J2SE 源代码中,类 javax.naming.ldap.StartTlsResponse 的注释为“setHostnameVerifier() must be called before negotiate() is invoked for it to have effect”.这个句子显式描述了 setHostnameVerifier 函数和 negotiate 函数之间的调用规则.注释和文档中也可能包含隐式描述规则的句子,例如在 JDBC 库的文档中,函数 java.sql.ResultSet.deleteRow 的描述为“Deletes the current row from this ResultSet object and from the underlying database”;函数 java.sql.ResultSet.close 的描述为“Releases this ResultSet object’s database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed”.虽然这两条描述里并没有明显包含与调用规约有关的内容,一个有经验的程序员能够根据上面的描述和使用资源的经验推断出这两个函数的调用规则.即,在调用 deleteRow 函数之后,需要调用 close 函数以关闭打开的数据库连接.

如上所述,自然语言描写的注释和文档里都包含丰富的可以用于挖掘规约的内容.但是,由于分析自然语言较为复杂,利用注释和文档作为输入的研究目前较少.

(3) 软件库代码

某些软件库是开源的,从软件库的源代码也可能挖掘出其调用规约.具体地说,软件库代码中的派生关系、函数间的调用关系、函数对内部状态的访问等都可能用于挖掘规约.一个直观的例子是,如果某个函数将其所在的类属性指向的内存释放,那么客户代码再直接调用其他使用此类属性指向内存片断的软件库函数,就可能抛出异常.如前所述,分析客户代码,以获得软件库函数的调用序列已经很难做得完全准确.而相比分析客户代码,分析软件库代码所需的分析技术则更为复杂.例如,要分析得到函数在什么条件下会影响到内部的状态,需要借助于复杂的代码分析技术.对于其中一些困难,已有的技术还无法很好地加以处理.由于所需的基础信息很难准确获得,目前利用软件库代码直接挖掘其调用规约的研究较为初步.

3 调用规约的表示

调用规约挖掘的结果是以各种形式描述的调用规约.目前,常见的调用规约表示形式如下.

(1) 常见调用序列

常见调用序列描述经常在客户代码中出现的函数调用顺序.一般来说,每条常见调用序列都对应一个区间为 0~1 的实数.这个实数为此常见调用序列出现的频率.例如,图 2 展示了 Yang 等人[6]提出的 Perracotta 挖掘到的两条规约.图中,矩形代表函数,边表示函数间的调用顺序,PAL 则表示相应的常见调用序列在用于挖掘的客户代码中出现的频率.挖掘以常见调用序列为形式的规约是目前规约挖掘的主流研究方向之一,其相关论文参见文献[6,9,12−14,18].

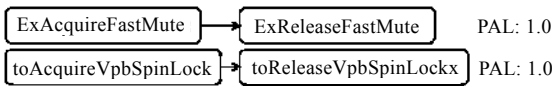
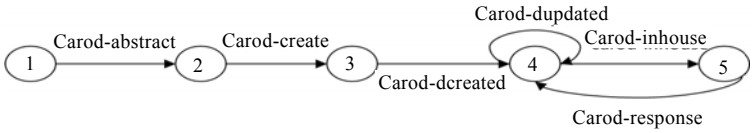


Fig.2 Specifications in the form of frequent call sequences

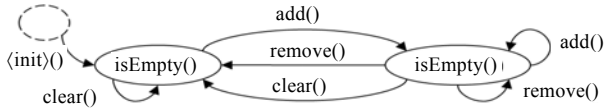
图 2 以常见调用顺序为形式的调用规约

(2) 自动机

自动机能够简洁地描述多个函数之间的调用规则.图 3(a)展示了 Cook 和 Wolf[3]提出的方法挖掘到的以自动机为形式的规约的局部.与绝大多数已有的方法一样,Cook 和 Wolf[3]提出的方法挖掘到的是匿名自动机,即状态没有名字,边则代表函数.匿名自动机在检查程序时比较方便,但不易于程序员理解.为此,Dallmeier 等人[10]提出了一种可以挖掘到命名自动机的方法.图 3(b)展示了 Dallmeier 等人[10]提出的方法为 Java 中的 Vector 类挖掘到的自动机.此方法能够挖掘出状态的名字,但不足之处在于需要人工参与.目前,挖掘以自动机为形式的规约也是目前的主流的研究方向之一,其相关论文参见文献[5,7,10,11,20,21,22].



(a)



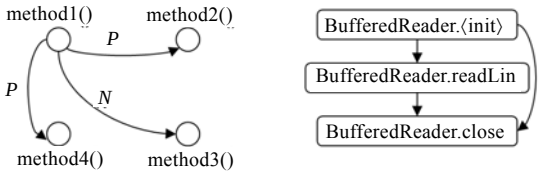
(b)

Fig.3 Specifications in the form of automata

图 3 以自动机为形式的调用规约

(3) 图

图可以描述较为复杂的规约.目前,只有少数研究能够挖掘以图为形式的规约.例如,图 4 展示了已有方法挖掘到的两种图:PRG 图和子图.其中,图 4(a)展示的是钟浩等人[36]提出的 JRF 工具挖掘到的 PRG 图.该图中的点表示函数,边则表示函数见的调用关系.例如,*P* 代表一个函数调用之前应该调用某个函数;*N* 则表示一个函数调用之后不能调用某个函数.图 4(b)展示的则是 Nguyen 等人[37]提出的方法挖掘到的子图.该图的点同样表示函数,边则表示函数间的控制流关系.他们方法挖掘到的子图是程序控制流图中经常出现的子图.



(a) (b)

Fig.4 Specifications in the form of graphs

图 4 以图为形式的调用规约

在 UML 建模语言中,状态图(自动机)和序列图都可以表示函数间的调用关系.如前所述,挖掘自动机为形式的规约是目前的主流研究之一,而挖掘序列图的研究则相对较少.Lo 和 Khoo[38]提出了一种能够挖掘以序列图为形式的调用规约的方法.图 5 展示了此方法挖掘到的调用规约.在挖掘序列图时,此方法利用了 Lo 等人[39]之

前提出的方法挖掘到的以常见调用序列为形式的调用规约.

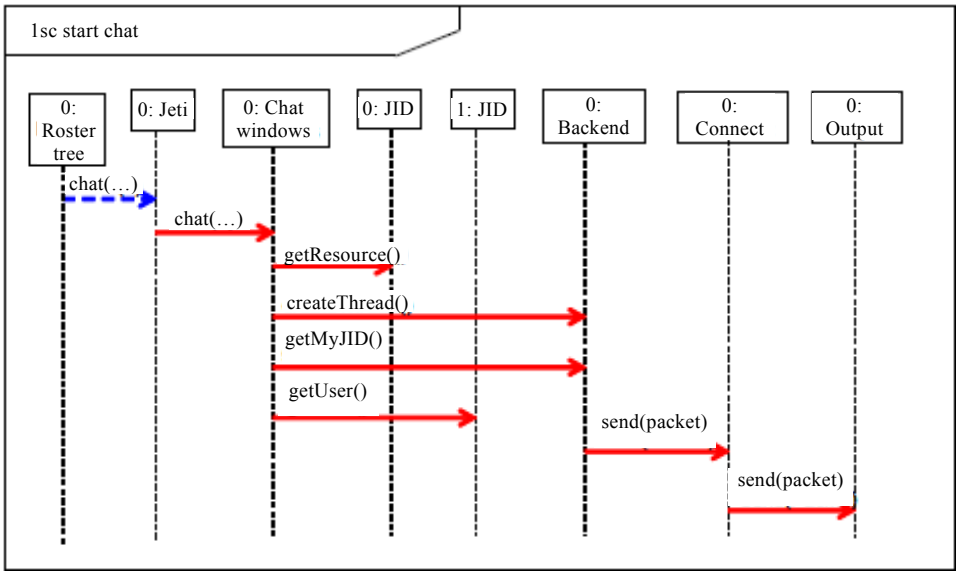


Fig.5 Specifications in the form of sequence diagrams

图 5 以序列图为形式的调用规约

(4) 时序逻辑

时序逻辑[40]是一种符号系统,用于描述并推理与时间相关的命题与规则.在模型检测[1]中,时序逻辑常常被用来描述系统应该满足的与时间有关的性质.常见的时序逻辑包括 LTL(linear temporal specification)和 CTL (computational tree logic)两类.例如, Wasylkowski 等人[17]开发的 JADET 工具能够挖掘出以常见调用序列为形式的规约.在此工具基础之上,Wasylkowski 和 Zeller[31]开发的 TIKANGA 工具能够将常见调用序列转化为以 CTL 为形式的规约,.

总之,目前的研究挖掘到的调用规约有常见调用序列、自动机、图和时序逻辑等形式.这些形式并非完全独立,常见调用序列可以看作是其他形式规约的基础.在一定程度上,不同形式的规约可以互相转换:首先,常见调用序列可以转换为自动机.Yang 等人[6]和 Gabel 等人[21]都认为,常见调用序列本质上是较为简单的自动机.Gabel 和 Su[22]则进一步提出一种将简单自动机拼接为较为复杂自动机的方法;其次,常见调用序列可以转换为图.例如,Lo 和 Khoo[38]提出的挖掘序列图的方法就是基于之前他们[39]提出的挖掘常见调用序列的方法;最后,常见调用序列可以转换为时序逻辑.Lo 和 Khoo[39]认为,Yang 等人[6]挖掘到的常见调用序列实质是 LTL 描述的规约, Wasylkowski 和 Zeller[31]提出的挖掘 CTL 的方法也是基于他们[17]之前提出的挖掘常见调用序列的方法.

4 调用规约挖掘的算法

根据用于挖掘的数据和期望挖掘到的规约形式,现有的挖掘算法主要分为如下几类:

(1) 基于出现频率的挖掘算法

基于出现频率的挖掘算法主要用于从客户代码挖掘规约,此类挖掘算法能够以挖掘常见调用序列为形式的调用规约.数据挖掘领域里,有很多现有的技术可以挖掘经常出现的项.因此,相当一部分调用规约挖掘方法都建立已有的数据挖掘算法之上.例如,Li 与 Zhou[14]开发的 PR-Miner 和 Livshits 与 Zimmermann[15]一起开发的 DynaMine,以及 Williams 和 Hollingsworth[16]一起开发的工具都利用了数据挖掘里的关联规则挖掘算法[46].又如,Ramanathan 等人[13]开发的 CHRONICLER 和 Wasylkowski 等人[17]开发的 JADET 都利用了数据挖掘中的序列挖掘算法[47].

为了更好地处理挖掘客户代码所特有的问题,也有研究者试图改进甚至提出新的挖掘算法.例如,Yang 等人[6]认为,相当部分的常见调用序列描述的是两个函数之间的关系.因此,他们提出的方法为挖掘常见函数对做了优化和改进.

(2) 基于文法推理的算法

基于文法推理的算法主要用于从客户代码挖掘规约,此类算法能够挖掘到以自动机为形式的规约.文法推理[48]的输入一般是一个正例样本集,有时候还包括一个反例样本集和其他的附加信息.语言文法推理的输出则是以自动机为形式的语言文法描述.例如,Cook 和 Wolf[5]就扩展了文法推理中经典的 Biermann 和 Feldman 算法[49]挖掘以自动机为形式的规约.

另外,也有研究者提出其他的算法来挖掘自动机.例如,Gabel 和 Su[21]认为,常见调用序列本质上是一种简单的自动机,并提出了一种将常见调用序列组合为自动机的算法.又如,Kremenek 等人[26]提出的算法预先定义好使用资源的自动机模板,然后利用客户代码的分析结果选择填入模板的具体函数.这样,他们提出的算法能够从客户代码挖掘到描述如何正确使用资源的,以自动机为形式的调用规约.

(3) 基于自然语言处理的算法

基于自然语言处理的算法主要用于从自然语言描述的文档挖掘规约.为了分析自然语言描写的内容,此类算法需要借助于自然语言处理以及数据挖掘中的各项技术.例如,Tan 等人[35]提出的 iComment 用一个分类器把包含函数调用关系的句子和不包含函数调用关系的句子区分开来.然后,iComment 用一组模板来挖掘函数间的规约.其中的一个模板为$\langle F_A\rangle$ must (NOT) be called before $\langle F_B\rangle$.通过模板,iComment 可以挖掘出注释中显示描述的规约.钟浩等人[42]提出的算法首先通过命名实体识别技术从函数的描述里分析出对应的动作和资源.然后通过比照一个预先定义的模版挖掘规约.这样,他们提出的方法就能够挖掘到函数文档里隐式描述的规约.

(4) 基于代码分析的算法

分析客户代码和分析软件库代码都需要借助于代码分析技术.但是就挖掘规约而言,分析客户代码的难度和涉及的代码分析技术的广度都比直接从软件库代码挖掘规约要低很多.分析客户代码需要获得软件库函数的调用顺序,而分析软件库代码则需要获得类的派生关系、函数调用关系、指针引用、对属性和局部变量的访问等多种信息.在本文中,基于代码分析的技术特指从软件库代码直接挖掘出规约的技术.

钟浩等人[36]提出的算法首先分析软件库函数内部操作和软件库类的派生关系;然后,JRF 利用一些基本的限制(如访问内存的内容之前必须首先分配内存)和一些的挖掘策略可以挖掘出包含调用顺序的规约.Long 和 Wang[41]提出的算法通过分析软件库代码挖掘其使用规则.他们提出的算法分析所有软件库提供的函数,如果某些函数访问的属性等有较多重叠,则认为这些函数是一起使用的.他们的算法并不能挖掘到与调用顺序有关的规则.就规约的准确度而言,钟浩等人[37]提出的算法还有待提高.就规约的表达能力而言,Long 和 Wang[41]提出的算法仍然较为初步.总之,要直接从软件库代码挖掘高质量的规约面临很多目前仍然难以克服的困难.

5 调用规约的用途

目前,调用规约的用途主要集中于以下两类:

(1) 检测程序中的缺陷

此类用途将调用规约与模型检测、静态分析和动态监控结合在一起:调用规约描述系统应当满足的性质,而上述方法则可以检查系统是否满足规约描述的性质.例如,Yang 等人[6]把挖掘到的调用规约送入静态代码分析工具 ESP[50]来检查 NTFS 文件系统的源代码,结果他们成功发现了一个得到微软开发团队确认的缺陷.又如,钟浩等人[42]将挖掘到的调用规约用于检查开源代码,结果发现了 35 个得到了开发者确认的缺陷.

(2) 辅助软件开发

规约描述了正确的软件库使用规则,潜在地能够帮助程序员理解软件库的用法.Thummalapenta 和 Xie[9]开发的 ParseWeb 尝试利用挖掘到的规约辅助软件开发.给定输入类型和输出类型,ParseWeb 能够返回可把输入类型转化为输出类型的常见调用序列.软件库函数在不同的编程上下文可能需要满足不同的调用规则.为此,钟浩

等人[51]提出了一种将聚类和序列挖掘结合起来的方法,此方法能够挖掘到与编程上下文敏感的规约.为了便于程序员理解规约描述的规则,钟浩等人[51]提出的方法进一步地将规约推荐和代码推荐结合起来,取长补短,从而达到了辅助程序员理解软件库用法的目的.

6 存在的问题和发展的方向

调用规约挖掘的研究历史较短,目前还有一些存在的问题.这些问题和可能的发展方向具体如下:

(1) 准确度不高

如前所述,绝大多数已有的方法都是基于客户代码分析的挖掘方法.Weimer 和 Mishra[52]指出,基于客户代码分析的方法挖掘到的调用规约的准确度普遍只有 1%~10%.他们认为,挖掘到的调用规约质量不高的一个重要原因是难以从客户代码获取准确的软件库使用细节.为此,钟浩等人[53]探讨了可能对调用规约挖掘造成负面影响的因素,并提出了相应的处理办法.但是,即使引入了此方法,挖掘到的调用规约的准确度仍然较低.一个假的规约如果用于检查缺陷,可能会导致很多虚假的报告;如果用于辅助软件开发,则可能误导程序员.为了更好地把挖掘到的规约应用于实际,提高规约准确度可能是将来亟待解决的问题.

(2) 依赖的数据源过于单一

绝大多数已有的方法依赖于单一的数据源,即已有的客户代码.综合利用多种数据源可能挖掘到质量更高的规约.但是,要结合不同的数据源并不容易.不同数据源得到的信息可能存在冗余、冲突等现象.如何综合利用这些不同的数据源、提出更好的规约挖掘方法,可能是未来将来潜在的发展方向.

(3) 用途较少

目前,调用规约的主要用途仍然局限在用于检测程序中的缺陷和辅助软件开发.规约在其他场景下也可能有用,但要挖掘在这些场景能起到作用的规约,目前还有很多工作需要做.例如,规约在测试中扮演重要的角色.但要把自动挖掘的调用规约用于测试,还需要在调用规约的表示、准确度等方面提升现有的规约挖掘研究.

7 总 结

调用规约描述了调用软件库应该满足的规则,在软件开发的多个环节都扮演着重要的角色.但是,受制于手工编写规约的巨大代价,大部分软件库都不提供完整的调用规约,或者根本不提供调用规约.为此,自动地挖掘软件库的调用规约正成为目前一个热点的研究方向.本文从调用规约挖掘的输入、表示、算法和用途几个方面阐述了当前调用规约挖掘的研究现状.在此基础上,本文还探讨了现有研究的不足和将来的发展方向.本文对于从事此项研究的学者和尝试在工业实践中利用调用规约挖掘的产业界人士具有一定的参考价值.

References:

[1] Lin HM, Zhang WH. Model checking: Theories, techniques and applications. Acta Electronica Sinica, 2002,30(12):1907−1912 (in Chinese with English abstract).

[2] Mei H, Wang QX, Zhang L, Wang J. Software analysis: A road map. Chinese Journal of Computers, 2009,32(9):1697−1710 (in Chinese with English abstract).

[3] Ernst MD, Perkins JH, Guo PJ, McCamant S, Pacheco C, Tschantz MS, Xiao C. The Daikon system for dynamic detection of likely invariants. Science of Computer Programming, 2007,69(1):35−45.

[4] Lorenzoli D, Mariani L, Pezzè M. Automatic generation of software behavioral models. In: Proc. of the 13th ICSE. 2008. 501−510. http://portal.acm.org/citation.cfm?id=1368157

[5] Cook J, Wolf A. Discovering models of software processes from event-based data. ACM Trans. on Software Engineering and Methodology, 1998,7(3):215−249. [doi: 10.1145/287000.287001]

[6] Yang J, Evans D, Bhardwaj D, Bhat T, Das M. Perracotta: Mining temporal API rules from imperfect traces. In: Proc. of the 28th ICSE. 2006. 282−291. http://portal.acm.org/citation.cfm?id=1134325&dl= [doi: 10.1145/1134285.1134325]

[7] Ammons G, Bodok R, Larus JR. Mining specifications. In: Proc. of the 29th POPL. 2002. 4−16. http://portal.acm.org/citation.cfm?id=503275 [doi: 10.1145/503272.503275]

[8] Chen F, Rosu G. Parametric trace slicing and monitoring. In: Proc. of the 15th TACAS. 2009. 246−261. http://www.springerlink.com/content/p270100k362x2w86/ [doi: 10.1007/978-3-642-00768-2_23]

[9] Thummalapenta S, Xie T. Parseweb: A programmer assistant for reusing open source code on the Web. In: Proc. of the 21st ASE. 2007. 204−213. http://portal.acm.org/citation.cfm?id=1321631.1321663 [doi: 10.1145/1321631.1321663]

[10] Dallmeier V, Lindig C, Wasylkowski A, Zeller A. Mining object behavior with ADABU. In: Proc. of the WODA. 2006. 17−24. http://portal.acm.org/citation.cfm?id=1138918 [doi: 10.1145/1138912.1138918]

[11] Shoham S, Yahav E, Fink S, Pistoia M. Static specification mining using automata based abstractions. In: Proc. of the ISSTA. 2007. 174−184. http://portal.acm.org/citation.cfm?id=1273487 [doi: 10.1145/1273463.1273487]

[12] Engler DR, Chen DY, Chou A. Bugs as deviant behavior: A general approach to inferring errors in systems code. In: Proc. of the 8th SOSP. 2001. 57−72. http://portal.acm.org/citation.cfm?id=502041 [doi: 10.1145/502034.502041]

[13] Ramanathan MK, Grama A, Jagannathan S. Path-Sensitive inference of function precedence protocols. In: Proc. of the 29th ICSE. 2007. 240−250. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4222586&tag=1 [doi: 10.1109/ICSE.2007.63]

[14] Li ZM, Zhou YY. PR-Miner: Automatically extracting implicit programming rules and detecting violations in large software code. In: Proc. of the 10th ESEC/13th FSE. 2005. 306−315. http://portal.acm.org/citation.cfm?id=1081755&dl= [doi: 10.1145/1081706.1081755]

[15] Livshits VB, Zimmermann T. Dynamine: Finding common error patterns by mining software revision histories. In: Proc. of the Joint 10th ESEC/13th FSE. 2005. 296−305. http://portal.acm.org/citation.cfm?id=1081754&dl=

[16] Williams CC, Hollingsworth JK. Recovering system specific rules from software repositories. In: Proc. of the 2nd MSR. 2005. 1−5. http://portal.acm.org/citation.cfm?id=1082983.1083144 [doi: 10.1145/1083142.1083144]

[17] Wasylkowski A, Zeller A, Lindig C. Detecting object usage anomalies. In: Proc. of the 6th ESEC and the 14th FSE. 2007. 35−44. http://portal.acm.org/citation.cfm?id=1287632 [doi: 10.1145/1287624.1287632]

[18] Weimer W, Necula G. Mining temporal specifications for error detection. In: Proc. of the 11th TACAS. 2005. 461−476. http://www.springerlink.com/content/48gf5k4nrja0f4p4/

[19] Acharya M, Xie T, Pei J, Xu J. Mining API patterns as partial orders from source code: From usage scenarios to specifications. In: Proc. of the 6th ESEC/14th FSE. 2007. 25−34. http://portal.acm.org/citation.cfm?id=1287630&dl= [doi: 10.1145/1287624.1287630]

[20] Lo D, Khoo SC. SMArTIC: Towards building an accurate, robust and scalable specification miner. In: Proc. of the 5th ESEC/13th FSE. 2006. 265−275. http://portal.acm.org/citation.cfm?id=1181775.1181808 [doi: 10.1145/1181775.1181808]

[21] Gabel M, Su Z. Symbolic mining of temporal specifications. In: Proc. of the 30th ICSE. 2008. 51−60. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4814116

[22] Gabel M, Su Z. Javert: Fully automatic mining of general temporal properties from dynamic traces. In: Proc. of the 7th ESEC/15th FSE. 2008. 339−349. http://portal.acm.org/citation.cfm?id=1453150 [doi: 10.1145/1453101.1453150]

[23] Henzinger TA, Jhala R, Majumdar R. Permissive interfaces. In: Proc. of the 10th ESEC/13th FSE. 2005. 31−40. http://portal.acm.org/citation.cfm?id=1081706.1081713 [doi: 10.1145/1081706.1081713]

[24] Reiss SP, Renieris M. Encoding program executions. In: Proc. of the 23rd ICSE. 2001. 221−230. http://portal.acm.org/citation.cfm?id=381497

[25] Whaley J, Martin MC, Lam MS. Automatic extraction of object-oriented component interfaces. In: Proc. of the ISSTA. 2002. 218−228. http://portal.acm.org/citation.cfm?id=566212 [doi: 10.1145/566172.566212]

[26] Kremenek T, Twohey P, Back G, Ng A, Engler D. From uncertainty to belief: Inferring the specification within. In: Proc. of the 7th OSDI. 2006. 259−272. http://portal.acm.org/citation.cfm?id=1298471

[27] Alur R, Černỳ P, Madhusudan P, Nam W. Synthesis of interface specifications for Java classes. In: Proc. of the 32nd POPL. 2005. 98−109. http://portal.acm.org/citation.cfm?id=1047659.1040314 [doi: 10.1145/1040305.1040314]

[28] Sankaranarayanan S, Ivanči F, Gupta A. Mining library specifications using inductive logic programming. In: Proc. of the 13th ICSE. 2008. 131−140. http://portal.acm.org/citation.cfm?id=1368107 [doi: 10.1145/1368088.1368107]

[29] Gowri M, Grothoff C, Chandra S. Deriving object type states in the presence of inter-object references. In: Proc. of the OOPSLA. 2005. 77−96. http://portal.acm.org/citation.cfm?id=1103845.1094818 [doi: 10.1145/1094811.1094818]

[30] Thummalapenta S, Xie T. Alattin: Mining alternative patterns for detecting neglected conditions. In: Proc. of the 24th ASE. 2009. 283−294. http://portal.acm.org/citation.cfm?id=1747526 [doi: 10.1109/ASE.2009.72]

[31] Wasylkowski A, Zeller A. Mining temporal specifications from object usage. In: Proc. of the 24th ASE. 2009. 295−306. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5431762 [doi: 10.1109/ASE.2009.30]

[32] Lo D, Maoz S. Mining hierarchical scenario-based specifications. In: Proc. of the 24th ASE. 2009. 359−370. http://portal.acm.org/citation.cfm?id=1747491.1747532 [doi: 10.1109/ASE.2009.19]

[33] Pradel M, Gross TR. Automatic Generation of object usage specifications from large method traces. In: Proc. of the 24th ASE. 2009. 371−382. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5431756 [doi: 10.1109/ASE.2009.60]

[34] Thummalapenta S, Xie T. SpotWeb: Detecting framework hotspots and coldspots via mining open source code on the Web. In: Proc. of the 23rd ASE. 2008. 327−336. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4639336 [doi: 10.1109/ASE.2008.43]

[35] Tan L, Yuan D, Krishna G, Zhou Y. iComment: Bugs or bad comments? In: Proc. of the 21st SOSP. 2007. 145−158. http://portal.acm.org/citation.cfm?id=1323293.1294276

[36] Zhong H, Zhang L, Mei H. Inferring specifications of object oriented APIs from API source code. In: Proc. of the 15th APSEC. 2008. 221−228. http://portal.acm.org/citation.cfm?id=1488103 [doi: 10.1109/APSEC.2008.54]

[37] Nguyen TT, Nguyen HA, Pham NH, Al-Kofahi JM, Nguyen TN. Graph-Based mining of multiple object usage patterns. In: Proc. of the 7th ESEC/FSE. 2009. 383−392. http://portal.acm.org/citation.cfm?id=1595767 [doi: 10.1145/1595696.1595767]

[38] Lo D, Maoz S, Khoo SC. Mining modal scenario-based specifications from execution traces of reactive systems. In: Proc. of the 22nd ASE. 2007. 465−468. http://portal.acm.org/citation.cfm?id=1321710 [doi: 10.1145/1321631.1321710]

[39] Lo D, Khoo SC, Liu C. Efficient mining of iterative patterns for software specification discovery. In: Proc. of the KDD. 2007. 460−469. http://portal.acm.org/citation.cfm?id=1281192.1281243 [doi: 10.1145/1281192.1281243]

[40] Huth M, Ryan M. Logic in Computer Science. Cambridge: Cambridge University Press, 2004.

[41] Long F, Wang X, Cai Y. API hyperlinking via structural overlap. In: Proc. of the ESEC/FSE. 2009. 203−212. http://portal.acm.org/citation.cfm?id=1595727 [doi: 10.1145/1595696.1595727]

[42] Zhong H, Zhang L, Xie T, Mei H. Inferring resource specifications from natural language API documentation. In: Proc. of the 24th ASE. 2009. 307−318. http://portal.acm.org/citation.cfm?id=1747528 [doi: 10.1109/ASE.2009.94]

[43] Rescher N, Urquhart A. Temporal Logic. New York: Springer-Verlag, 1971.

[44] Clarke E, Grumberg O, Peled D. Model Checking. MIT Press, 1999.

[45] Lo D, Khoo SC. Software specification discovery: A new data mining approach. In: Proc. of the NGDM. 2007. http://en.scientificcommons.org/42486510

[46] Agrawal R, Imielínski T, and Swami A. Mining association rules between sets of items in large databases. In: Proc. of the ICMD. 1993. 207−216. http://portal.acm.org/citation.cfm?id=170036.170072 [doi: 10.1145/170035.170072]

[47] Agrawal R, Srikant R. Mining sequential patterns. In: Proc. of the 7th ICDE. 1995. 3−14. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=380415 [doi: 10.1109/ICDE.1995.380415]

[48] Parekh R, Honavar V. Grammar Inference, Automata Induction, and Language Acquisition. Handbook of Natural Language Processing. Marcel Dekker, 1998.

[49] Biermann A, Feldman J. On the synthesis of finite state machines from samples of their behavior. IEEE Trans. on Computer, 1972,21(6):592−597. [doi: 10.1109/TC.1972.5009015]

[50] Das M, Lerner S, Seigle M. ESP: Path-Sensitive program verification in polynomial time. In: Proc. of the PLDI. 2002. 57−68. http://portal.acm.org/citation.cfm?id=512529.512538 [doi: 10.1145/512529.512538]

[51] Zhong H, Xie T, Zhang L, Pei J, Mei H. MAPO: Mining and recommending API usage patterns. In: Proc. of the 23rd ECOOP. 2009. 318−343. http://www.springerlink.com/content/m342264m3h326774/

[52] Weimer W, Mishra N. Privately finding specifications. IEEE Trans. on Software Engineering, 2008,34(1):21−32. [doi: 10.1109/TSE.2007.70744]

[53] Zhong H, Zhang L, Mei H. Early filtering of polluting method calls for mining temporal specifications. In: Proc. of the 15th APSEC. 2008. 9−16. [doi: 10.1109/APSEC.2008.53]

附中文参考文献:

[1] 林惠民,张文辉.模型检测:理论、方法与应用.电子学报,2002,30(12):1907−1912.

[2] 梅宏,王千祥,张路,王戟.软件分析技术进展.计算机学报,2009,32(9):1697−1710.

钟浩(1979－),男,重庆人,博士,助理研究员,主要研究领域为软件工程,挖掘软件工程数据.

梅宏(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程.

张路(1973－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件复用,程序理解.