C 程序隐式规则自动提取与反例检测

禹 振,苏小红,王甜甜,马培军
(哈尔滨工业大学计算机科学与技术系,黑龙江哈尔滨 150001)

摘 要: 提出一种通用且高效的隐式规则自动提取与反例检测方法,使用频繁闭合项集挖掘技术挖掘包含多种程序元素的编程模式,然后由编程模式产生编程规则;引入正序规则的概念,以避免从同一个编程模式中产生多个冗余规则.在此基础上,提出一种高效的反例检测算法检测违反规则的程序片段.实验结果表明,该方法能够自动提取程序中存在的隐式编程规则,并快速有效地检测违反规则的反例.

关键词: 频繁闭合项集挖掘; 程序规则提取; 反例检测; 软件缺陷检测; 静态分析
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2013)02-0248-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.02.007

Automatically Extracting Implicit Programming Rules and Detecting Violations from C Programs

YU Zhen, SU Xiao-hong, WANG Tian-tian, MA Pei-jun
(*School of Computer Science and Technology, Harbin Institute of Technology, Harbin, Heilongjiang* 150001, *China*)

Abstract: A general and efficient method is proposed to automatically extract the rules and detect violations to the rules extracted. Closed frequent itemset mining is applied to mine programming patterns. Then these patterns are used to generate programming rules. The concept of Positive Order Rule is introduced to avoid generating redundant rules from the same programming pattern. Based on these efforts, we also propose an efficient violations detection algorithm to detect program segments that are not consistent with the extracted rules. The experiment results on large software source code indicate that this method can automatically extract lots of implicit programming rules and also can efficiently detect the code segments that violate the extracted rules.

Key words: frequent closed itemset mining; programming rules extracting; violations detecting; software defects detecting; static analysis

1 引言

大型程序中存在着大量隐式编程规则,但它们中的大多数由于过分分散和隐晦,而且数量众多,而难以在文档中逐一说明,然而隐式编程规则可能表达了程序的内在特性和特定需求,如果程序员忘记或者未注意到这些规则,就很容易将缺陷引入程序.因此,很有必要开发一种能够自动提取这些隐式规则并自动进行反例检测的工具.

已有研究工作主要专注于某一类规则的提取,如函数-函数规则[1,2]或者变量-变量规则[3].文献[1]使用频繁闭合序列挖掘技术提取 API 错误处理规范.与文献[1]类似,文献[2]基于频繁闭合序列技术挖掘面向对象语言如 Java 和 C++ 中的异常处理规范.MUVI[3]利用频繁闭合项集挖掘技术推断程序中变量间的访问关联关系,并检测相应的语义和并发缺陷.PR-Miner[4]也使用频繁闭合项集挖掘技术,但能挖掘出更多类型的规则,包括函数-函数,函数-变量,以及变量-变量关联规则.然而,PR-Miner 挖掘出来的规则中绝大多数是逆序规则和冗余规则,应该被消除.

针对这些问题,本文采用抽象语法树、Token 流和频繁闭合项集挖掘技术相结合的方法,同时引入正序规则的概念,提出一种改进模型,既能自动挖掘出多种类型的编程规则并检测反例,又能避免产生大量的逆序和冗余规则.

收稿日期:2011-05-18;修回日期:2012-09-13
基金项目:国家自然科学基金(No.60673035);高等学校博士学科点专项科研基金(No.20092302110040)

2　相关概念定义

本文提出的方法能从软件源码中挖掘三种类型的规则,即函数-函数规则,变量-变量规则及函数-变量规则,下面给出它们的定义和实例.

定义 1　函数-函数规则:由两个或多个函数构成,规则的左部或者右部至少包含一个函数,这样的规则称为函数-函数规则.

图 1 给出了文献[4]在 PostgreSQL 中提取的一条函数-函数规则:如果为 HeapTuple 类型的变量调用了 SearchSysCache()函数,那么必须在以后的某个时候为该变量调用 ReleaseSysCache()函数.这是因为 SearchSysCache()为 HeapTuple 类型变量分配内存,而 ReleaseSysCache()负责释放它.这条规则在 PostgreSQL 中出现了 209 次.如果某个程序片段违反了它,则有可能造成内存泄漏.

```
PostgreSQL-8.0.1/src/backend/catalog/dependency.c
1733  getRelationDescription(StringInfo buffer, Oid rehd)
1734  {
1735      HeapTuple relTup;
          ......
1740      relTup = SearchSysCache(...);
          ......
1796      ReleaseSysCache(relTup );
1797  }
```

图1　PostgreSQL中的一条函数-函数规则

```
Linux-1.0/fs/ext2/namei.c
705   int ext2_unlink(...)
706   {
          ......
708       struct inode * inode;
          ......
755       inode->i_nlink --;
756       inode->i_dirt =1;
          ......
764   }
```

```
Linux-1.0/fs/minix /namei .c
305   int minix _mkdir
          (  struct inode* dir, ...)
306   {
307       ......
365       dirt->i_nlink ++;
366       dir->i_dirt=1;
          ......
371   }
```

(*a*) ext2_unlink中的V-V规则　　(*b*) minx -mkdir中的V-V规则

图2　Linxu-1.0中一条变量-变量规则

定义 2　变量-变量规则:由至少两个结构体变量或者结构体的域成员构成.规则的左部或者右部至少包含一个变量,这样的规则称为变量-变量规则.

图 2 显示了本文方法在 Linux-1.0 中发现的一条变量-变量类型的规则,它出现了 23 次.该规则指出如果文件的链接数目发生变化,则其 dirt 位应被设置成 1,以便将文件内容写回磁盘.故 *inode*→*i_nlink* 和 *inode*→*i_dirt* 应该同时被访问或者修改.

定义 3　函数-变量规则:函数-变量规则由函数或者变量构成,它是函数-函数规则和变量-变量规则的超集,但还包括函数与变量之间的混合耦合关系.

```
PostgreSQL-9.1 beta 1\contrib\dblink\dblink.c
1586  static char **
1587  get_pkey _attnames  (... )
1588  {
1591      SysScanDesc scan;
1592      HeapTuple indexTuple;
1609      scan = systable_beginscan(.....)
1612      while (HeapTupleIsValid (indexTuple =
                          systable_getnext(scan)))
          .....
1631      systable_endscan(scan);
1635  }
```

图3　PostgreSQL-9.1中一条函数-变量规则

图 3 显示一条函数-变量规则,它表达了四个函数与两个变量之间的耦合关系.函数 *systable_beginscan*()给 *SysScanDesc* 类型变量 *scan* 赋值,然后在 while 循环中调用 *systable_getnext*()不断获取 *HeapTuple* 索引元组 *indexTuple*,*HeapTupleIsValid*()测试 *indexTuple* 是否有效.操作的最后,调用 *systable_endscan*()释放资源.

图 4 展示了本文方法在 Linux－2.0 中提取出的一条函数-函数规则以及违背该规则的一个反例.图 4(*a*)中的规则表达了进行中断屏蔽以及中断恢复的正确顺序:首先将中断标志保存到 flags 中,然后清空当前中断标志寄存器,接着进行一系列操作,最后将 flags 中的内容恢复到寄存器.然而,图 4(*b*)中的程序片段遗忘了对 cli()的调用,它可能在后续操作期间不能有效地屏蔽中断请求.

3　隐式编程规则提取与反例检测模型

本文提出的基于抽象语法树与频繁闭合项集的隐式编程规则提取和反例检测模型如图 5 所示.

该模型的基本思路为:首先对源程序进行词法和语法分析,建立抽象语法树;然后在抽象语法树上,以函数定义为基本单位,进行过程内分析,提取感兴趣的语法成分,如结构体变量名、函数调用名等;接着使用冲突率较低的字符串哈希算法,将收集到的语法成分

```
Linux-2.0/arch/m68k/atari/atasound.c
38  static void atari_nosound(.....)
39  {
44      save_flags(flags);;
45      cli();
46      sound_ym.rd_data_reg_sel = 7;
47      tmp = sound_ym.rd_data_reg_sel;
48      sound_ym.wd_data=tmp| 0x39;
49      restore_flags(flags);
50  }
```

```
Linux-2.0\drivers\isdn\isdn_net.c
1974  int isdn_get_setcfg(.....)
1975  {
2040      save_flags(flags);
          // cli() is missing!
2041      if((i=isdn_get_free_channel (
                ..... chidx))<0){
2047          restore_flags (flags);
2048          return -EBUSY;
2049      }
2128  }
```

(*a*) atari_nosound中的规则　　(*b*) isdn_get_setcfg中的反例

图4　Linxu-2.0中一条编程规则(出现88次)及其反例

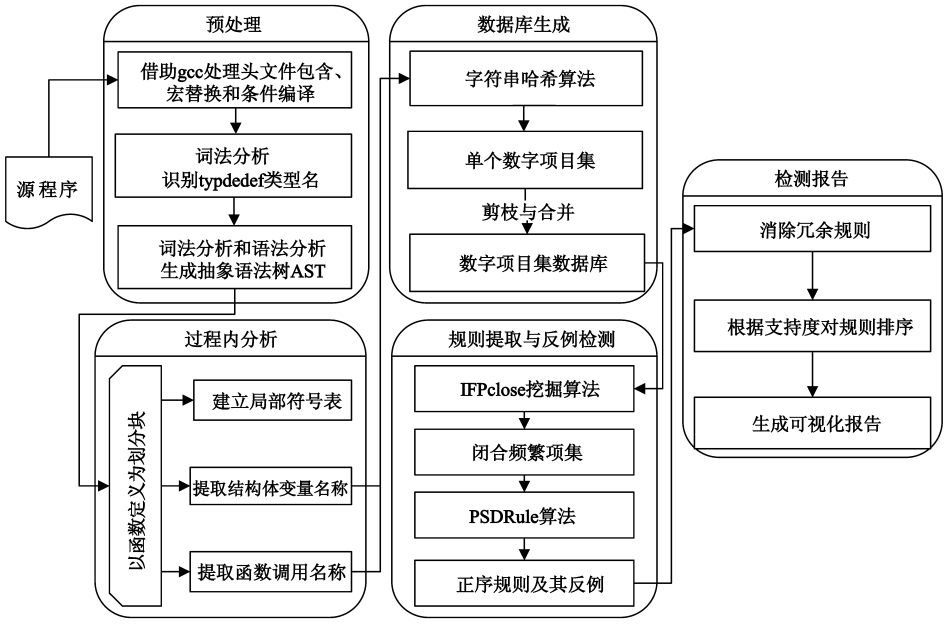

图5 隐式编程规则提取与反例检测模型

映射成数字(也称数字项目),这样一个函数定义内的所有被选择的语法成分对应的数字项目就构成了一个数字项目集.剪除掉那些只含有单个元素的数字项目集,合并数字项目集内部的重复数字项目,剩下的数字项目集的集合就构成数字项目集数据库.先使用频繁闭合项集挖掘算法(Improved FPclose,IFPclose)挖掘程序中的编程模式,然后使用正序规则生成算法(Postive Order Rule,PSDRule)从编程模式生成编程规则,并在生成规则的同时进行反例检测.与 PR-Miner 闭规则方法[4]不同的是,正序规则方法不仅能生成置信度为 100% 的规则,还能消除由闭规则方法产生的绝大部分冗余和逆序规则.最后,根据支持度大小对规则进行降序排序,生成可视化报告.

3.1 选取感兴趣的语法成分

我们只考虑结构体变量和函数调用这两种语法成分.相对于那些简单变量(如 int,char,float 等),它们之间的关联关系更普遍也更重要[3].另外,我们观察到包含在同一条规则中的结构体变量可能含有不同的变量名.如图 2 所示,在函数 *ext2_unlink* 和 *minix_mkdir* 中,示例规则使用了不同的变量名 *inode* 和 *dir*.如果单纯根据字面值将它们映射到不同的数字,则可能漏检该规则.因此我们使用结构体变量的类型名替代变量本身,如上述两个变量都被同一类型名 *inode* 代替.

3.2 数字项目数据库生成

本文使用哈希函数 *lh_strhash*[3],实现从字符串字面值到数字串的映射.一个函数定义体内的所有被选择的语法成分对应的数字项目就构成一个数字项目集.我们剪除那些只具有单个元素的数字项目集,它们在后续的规则生成阶段没有意义,将其包含在项目集数据库中只会增加频繁闭合项集挖掘算法运行的时空开销.同时,因为 IFPclose 算法只能处理含有互异项目的数字项目集,故我们对项目集中的重复项目进行合并,使得保留下来的项目各不相同.这样就得到了最后的数字项目集数据库.

3.3 改进的频繁项集挖掘算法

通过将源程序映射成数字项目集数据库,我们将规则提取问题转化为频繁项集挖掘问题.为解决该问题,人们提出了很多挖掘算法[12].本文参考文献[6~10],提出了 FPclose[9]的改进算法 IFPclose.FPclose 使用分治策略深度优先递归搜索条件 FP-tree,直到 FP-tree 只含有单条路径时,才产生闭合频繁项集(Closed Frequent Itemset,CFI)候选.这导致它不能及时发现已经出现在条件模式基中的 CFI 候选,并可能漏检某些 CFI.IFPclose 在深度优先递归搜索的过程中,同时考虑条件模式基的支持度和 FP-tree 的结构问题.当发现条件模式基的支持度大于当前条件模式树中元素的最大支持度时,则条件模式基必定是一个 CFI 候选.另外,当条件模式树是单路径时,IFPclose 的处理策略也不同于 FPclose.其算法如图 6 所示.

IFPclose 算法只挖掘频繁闭合项集,而不是生成所有频繁项集.频繁闭合项集是一个频繁项集,它的支持度大于其超频繁项集.只挖掘频繁闭合项集既能显著减少需要生成的项集的数目,同时又能保留频繁项集的所有信息.

3.4 规则生成和反例检测

对于从数字项目集数据库中挖掘出来的一个频繁

```
算法：IFPclose (fpt,cft,base)
输入：fpt,一个FP-tree
      cft,一个CFI-tree,全局数据结构,初始为空
      base,条件模式基,初始为空
输出：经过更新的cft
1.  if base.support> the max support of nodes in fpt.head
2.      insert base into cft
3.  if fpt contains only a single path
4.      if fpt has only a single element a
5.          let belta = a ∪ base,
                and belta. support = smallest support in belta
6.          insert belta into cft
7.      else fpt has only a single path containing more than one node
8.          if nodes in fpt . head have all the same support, such as sp
9.              let belta = fpt. head ∪ base, and belta. support = sp;
10.             insert belta into cft
11.         else nodes in fpt. head don't have the same support
12.             goto 14;
13. else fpt has more than one path
14.     for each node a_i in fpt.head
15.         let belta = a_i ∪ base ,and belta.support = a . support
16.         construct a_i's conditional FP-tree cfpt
17.         if cfpt is not empty then IFPclose (cfpt, cft, belta);
18.         else insert belta into cft
```

图6　IFPclose算法

项集而言,我们称与该频繁项集对应的语法成分的集合为程序的编程模式,它表明这些语法成分相互关联或者经常一起被使用.由 IFPclose 算法已经得到频繁闭合项集,即编程模式后,需要解决的问题是如何从编程模式产生编程规则.

文献[4]提出闭规则方法.闭规则包含了具有相同支持度、支持项集和置信度的其他数量众多的子规则,如果只产生闭规则,则可以大大减少需要产生的规则的数目,同时又能向审查人员呈现出有益的规则.闭规则的内部表示形式为

$I: s \mid \{C_1: s_1 \mid s_1 > s\}, \{C_2: s_2 \mid s_2 > s\}, \cdots \{C_m: s_m \mid s_m > s\}$

其中 I 是频繁闭合项集,s 是其支持度,$C_1, C_2, \cdots C_m$ 是 I 的子集,且其相应支持度 $s_1, s_2, \cdots s_m$ 都大于 I 的支持度 s.称这种表示形式为频繁闭合项集 I 的闭规则压缩形式,它表达了从 I 可以产生的所有闭规则.对于从频繁闭合项集 I 产生出来的一个规则 $X \Rightarrow Y$,如果 X 是 $C_1, C_2, \cdots C_m$ 中的某个 C_i,则该规则称为闭规则,规则的置信度可直接得到 s/s_i;否则,即 X 与 $C_1, C_2 \cdots C_m$ 都不等,则规则 $X \Rightarrow Y$ 的置信度为 100%.因此只要能将每个频繁闭合项集 I 表示成这种格式,我们就能容易地产生所有的闭规则.

这样就把规则生成问题转化为如何求得频繁闭合项集 I 的所有支持度 s_i 大于 I 的支持度 s 的子集 C_i 的问题.因为 C_i 的支持度 s_i 大于 s,所以 C_i 必定包含在其他频繁闭合项集 J 中,且这样的 J 至少存在一个.包含 C_i 的频繁闭合项集可能有多个,我们要找到具有最大支持度的那个频繁闭合项集 K,这样 C_i 的支持度就是 K 的支持度.这样如果某一个项集 W 存在于至少两个频繁闭合项集中,则 W 必定是某个频繁闭合项集 I 的 C_i,因此我们再次调用 IFPclose 算法对第一次调用 IFPclose 算法挖掘出来的 CFI 集合进行挖掘(置最小支持度为 2),得到 CFI 之间的所有可能公共子集的集合 *CommonSub*.其元素 cs 为三元组 $<F, s, E>$:F 是公共子集本身;s 是其支持度,定义为 cs 的具有最大支持度的支持项集的支持度;E 是其支持项集集合.闭规则压缩格式生成算法如图 7 所示.

```
算法：GenCondensedFormat (I)
输入：频繁闭合项集集合I ={I_k|1≤k≤n},
      其中I_k是四元组 <F_k, S_k, E_k, ComSub_k>,
      ComSub_k表示I_k的所有支持度大于 S_k 的子集组成的集合
输出：以闭规则压缩形式表示的频繁闭合项集集合I
1.  mine common closed frequent sub - itemsets CommonSub from I
2.    CommonSub = IFPclose ( {F_i|i=1,2,...n}, 2);
3.    where CommonSub = {cs_i|1≤i≤m}, and
      cs_i has 3 fields< F_i^cs, s_i^cs, E_i^cs >
4.  for i=1,2,...m
5.      Denote cs_i's support itemsets E_i^cs has t_i elements{I_1^i, I_2^i, ...I_n^i}
6.      for j =1, 2, ... t_i
7.          if cs_i.support > I_j^i.support
8.              insert cs_i into I_j^i's ComSub
```

图7　闭规则压缩格式生成算法

利用频繁闭合项集的压缩格式表示,我们能在生成规则的同时进行反例检测.然而 PR-Miner 只产生置信度小于 100% 的闭规则,而忽略掉置信度为 100% 的规则.该方法有两个缺点:100% 规则是合理的隐式规则,却没有提取出来;实际中,审查人员不可能从错综复杂的闭规则中推导出 100% 规则,虽然这在理论上是可行的.此外,文献[4]产生的规则绝大多数是冗余和逆序规则,应该被消除.因此我们引入正序规则的概念.

定义 4　正序规则:设有频繁项集 $I = \{i_1, i_2, \cdots i_m\}$ 是二元组 $< im_k, ix_k >$,其中 im 是项目本身,ix 是以自然数标记的项目索引.若规则 $X \Rightarrow Y$(其中 $X = \{i_{j1}, i_{j2}, \cdots i_{jp}\}$, $Y = \{i_{u1}, i_{u2}, \cdots i_{uq}\}$, $X \cap Y = \varnothing$, $X \cup Y = I$)是正序规则,则有 $ix_{j1} \leqslant ix_{j2} \leqslant \cdots \leqslant ix_{jp}$ 与 $ix_{u1} \leqslant ix_{u2} \leqslant \cdots \leqslant ix_{uq}$ 成立,且不存在 $i_g \in X$ 与 $i_h \in Y$,使得 $ix_g > ix_h$,否则,称之为逆序规则.

正序规则生成及反例检测算法如图 8 所示.在反例检测中,对于一条规则,我们首先找到规则左部的具有最大支持度的支持项集,该支持项集实际上是频繁闭合项集,设置该频繁闭合项集的支持项集集合为规则左部的支持项集集合 *supsets_left*;然后对 *supsets_left* 中的每个元素,在规则的支持项集 *supsets_main* 中进行查找,如果找不到,这说明该项集只包含了规则的左部,而没有同时包含规则的右部,该项集就是规则的反例项集.

4　实验结果与分析

本文将 Linux－1.0, Linux－2.0, Http－2.2.2, MySQL－5.0.92源代码作为输入,进行了规则提取和反例检

```
算法：PSDRule (CFIS)
输入：具有压缩格式表示的频繁闭合项集集合 CFIS={cfi_k|1≤k≤n}，其中 cfi_k 是四元组
      <F_k,S_k,E_k,ComSub_k>，ComSub_k 表示 cfi_k 的所有支持度大于 S_k 的子集组成的集合，
      其中元素 cs 的形式为三元组 <F^cs, S^cs, E^cs>
输出：正序规则集合 Rules
1.  construct the rule's array Rules, initially empty
2.  for each cfi in CFIS
3.      denote cfi.F has n elements, ascend elements in cfi.F according to the line number of elements
4.      for the first w items in cfi.F, w from 1 to n-1
5.          for each cs in cfi.ComSub
6.              if left-candi == cs
7.                  produce a new rule rule; make rule.mainCFI = cfi;
8.                  rule.left = cs; rule.right = cfi-cs;
9.                  rule.sup = cfi.sup; rule.supsets = cfi.supsets; rule.conf = cfi.sup/cs.sup;
10.                 DetectRuleViolations (rule); insert rule into Rules
11.         check that left-candi is a subset of some cs or not. If it is, goto 4
12.         if left-candi is not a subset of any cs, then produce a new rule rule whose confident is 1
            make rule.viosets empty, rule.mainCFI = cfi, rule.sup = cfi.sup, rule.supsets = cfi.supsets,
            rule.left = cs, rule.right = cfi-cs, rule.conf = 1.0
13.         insert rule into Rules

子例程：DetectRuleViolations (rule)
  输入：闭规则 rule 是七元组 <mainCFI, left, right, sup, conf, supsets, viosets>，其中 mainCFI 表示与
        rule 关联的频繁闭合项集，left 表示规则的左部，right 表示规则的右部，sup 是规则的支持度，
        conf 为规则的置信度，supsets 表示规则的支持项集集合，viosets 表示规则的反例项集集合
  输出：经过更新的 rule
1.  denote maxsup_left as the itemset having maximum support among itemsets supporting left,
    denote supsets_left as sets of itemset supporting maxsup_left,
    denote supsets_main as sets of itemset supporting mainCFI
2.  for each ims_left in supsets_left
3.      check that ims_left is in supsets_main or not
4.      if ims_left is not in supsets_main
5.          insert ims_left into rule.viosets
```

图8 规则生成与反例检测算法

测实验.实验设置最小支持度 15,最小置信度 0.9;实验环境 AMD Athlon 2.6GHZ CPU,2GB 内存,Windows XP 操作系统.实验结果如表 1 所示.其中 #ItemSet 表示原始项集集合经过剪枝和合并后剩下的项集数目,#Total 表示挖掘出的规则与反例总数,#Confirm 表示经人工分析后确认的规则与反例数目,#Time 表示两次调用 IFP-close 算法以生成频繁闭合项集压缩格式和进行规则提取和反例检测的总时间,#Space 表示最大内存消耗.图 2、图 3 和图 4 展示了本文检测到的一部分规则和反例.

表 1 4 个开源项目的实验结果

程序名称	代码规模 (LOC)	项集数目 (# ItemSet)	# Total		# Confirm		时间消耗 # Time(s)	内存消耗 # Space(MB)
			# Rule	# Vio	# Rule	# Vio		
Linux – 1.0	110K	680	45	23	9	0	3.98	11.4
Linux – 2.0	431K	952	29	20	8	3	3.80	15.8
Http – 2.2.2	200K	2907	36	31	21	4	3.80	15.8
MySQL – 5.0.92	764K	8168	383	88*	6*	0*	294.50	59.3

注:表中的 * 表示限于时间和精力,只对规则总数的前 30 条进行了统计和分析

相较于 PR-Miner[4],本文方法只产生正序规则.这不仅消除了闭规则方法[4]产生的大量冗余规则和逆序规则,而且能够生成 PR-Miner 不能产生的置信度为 100%的规则.分别用本文正序规则方法和文献[4]的 PR-Miner 闭规则方法对四个项目的源代码进行实验,结果如表 2 所示.从表中可以看出,与闭规则方法相比,正序规则方法在减少所生成的规则的数目的同时,却增加了检验出的真实规则的数目;此外,正序规则方法还提取出了数条置信度为 100% 的规则,避免了 PR-Miner 闭规则方法对这些规则的漏检.

另外,本文还采用人工注入规则与反例的方法,对 Linux – 1.0 和 Linux – 2.0 源代码进行了测试.其中在 Linux – 1.0 中注入 10 条规则,每条规则关联着 0,1 或 2 个反例,共 11 个反例;在 Linux – 2.0 中注入 10 条规则,每条规则都关联着 1 个反例,共 10 个反例.人工注入的 20 条规则中,有 6 条变量-变量规则,14 条函数-函数或者函数-变量规则.实验参数设置与表 1 相同.从表 3 给出的人工注入实验结果看,本文方法能够检测出所有

规则和反例,即没有漏检,但误检较高,检测出的 44 条规则中仅有 20 条是真正的规则.这是因为正序规则方法虽然能够生成置信度为 100% 的规则,但同时也生成了一些冗余规则,尽管如此,相比于 PR-Miner 生成的冗余规则,数量还是大大降低了.

表 2　正序规则方法与 PR-Miner 方法实验结果对比

程序名称	PR-Miner 闭规则方法					本文正序规则方法				
	# Total		# Confirm		100% 规则	# Total		# Confirm		100% 规则
	# Rule	# Vio	# Rule	# Vio		# Rule	# Vio	# Rule	# Vio	
Linux – 1.0	214	292	6	0	0	45	23	9	0	3
Linux – 2.0	120	188	4	3	0	29	20	8	3	4
Http – 2.2.2	49	75	9	4	0	36	31	21	4	10
MySQL – 5.0.92	1679	272*	4*	0*	0*	383	88*	6*	0*	2*

注:表中的 * 表示限于时间和精力,只对规则总数的前 30 条进行了统计和分析

表 3　人工注入规则和反例的实验结果

程序名称	人工注入		检测结果		人工确认		漏检情况	
	#规则	#反例	#规则	#反例	#规则	#反例	#规则	#反例
Linux – 1.0	10	11	26	11	10	11	0	0
Linux – 2.0	10	10	18	10	10	10	0	0

5　结论

本文对已有的规则提取和反例检测模型进行了改进,针对传统方法不能产生置信度为 100% 的规则和产生的规则中存在大量冗余规则和逆序规则的问题,引入正序规则的概念,在产生置信度为 100% 规则的同时,过滤掉冗余规则和逆序规则.通过对实际开源代码进行分析,本文方法能够找到更多有效的规则和真实的反例.而且,人工注入实验表明,本文方法能够检测出所有规则和反例,漏检率为 0.本文方法的缺点是只能挖掘单个函数范围内的规则,无法挖掘跨函数的规则.未来的工作是使用动态分析(如 MOPS[11])的方法,获取大量的程序运行轨迹,再从这些运行轨迹中挖掘跨函数规则.

参考文献

[1] Acharya M, Xie T. Mining API error-handling specifications from source code [A]. Proceedings of the 12th International Conference on Fundamental Approach to Software Engineering [C]. Heidelberg: Springer-Verlag, 2009. 370 – 384.

[2] Thummalapenta S, Xie T. Mining exception-handling rules as sequence association rules [A]. Proceedings of the 31st International Conference on Software Engineering [C]. Washington D C: IEEE Computer Society Press, 2009. 496 – 506.

[3] Lu S, Park S, Zhou Y Y. MUVI: automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs [A]. Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles [C]. New York: ACM Press, 2007. 103 – 116.

[4] Li Z M, Zhou Y Y. PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code [A]. 13th ACM SIGSOFT Symposium on the Foundations of Software Enginerring [C]. New York: ACM Press, 2005. 306 – 315.

[5] Chang R Y, Podgurski A. Discovering programming rules and violations by mining international dependences [J]. Journal of Software Maintenance and Evolution: Research and Practice, 2011, 23(3): 160 – 175.

[6] Pei J, Han J W, Mao R Y. CLOSET: an efficient algorithm for mining frequent closed itemsets [A]. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery [C]. New York: ACM Press, 2000 21 – 30.

[7] Wang J Y, Han J W, Pei J. CLOSET + : searching for the best strategies for mining frequent closed itemsets [A]. Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining [C]. New York: ACM Press, 2003. 236 – 245.

[8] Graphne G, Zhu J F. High performance mining of maximal frequent itemsets [A]. ISAM03 Workshop on High Performance Data Mining: Pervasive and Data Stream Mining [C]. New York: ACM Press, 2003. 135 – 143.

[9] Graphne G, Zhu J F. Efficiently using prefix-tress in mining frequent itemsets [A]. Proceedings of the 12th ACM SIGSOFT Symposium on Operating Systems Design and Implementation [C]. New York: ACM Press, 2003. 110 – 119.

[10] Han J W, Pei J, Yin Y W. Mining frequent patterns without candidate generation [A]. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data [C]. New York: ACM Press, 2000. 1 – 12.

[11] Chen H, Wagner D. MOPS: An infrastructure for examining security properties of software [A]. Proceedings of the 9th ACM Conference on Computer and Communications Security

[C]. New York: ACM Press, 2002. 235 – 244.
[12] 刘晓东,刘大有. 数据挖掘专利综述[J]. 电子学报, 2003, 31(12A): 1989 – 1993.
Liu Xiaodong, Liu Dayou. Data Mining Patent Summarization [J]. Acta Electronica Sinica, 2003, 31(12A): 1989 – 1993. (in Chinese)

作者简介

禹 振 男,1987 年生. 博士研究生. 研究方向为软件缺陷检测.
E-mail: yuzhen_3301@yahoo.com.cn

王甜甜 女,1980 年生. 博士. 讲师. 主要研究方向:程序分析、软件缺陷检测、软件测试等.
E-mail: sweetwtt@126.com

苏小红 女,1966 年生. 教授、博士生导师. 中国计算机学会高级会员. 主要研究方向:软件缺陷检测、软件重构、、信息融合、目标检测与跟踪等.
E-mail: sxh@hit.edu.cn

马培军 男,1963 年生. 教授、博士生导师. 主要研究方向:空间计算、信息融合、软件工程、目标检测与跟踪等.
E-mail: ma@hit.edu.cn