ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.2, February 2010, pp.209−217
doi: 10.3724/SP.J.1001.2010.03782

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

一种路径敏感的静态缺陷检测方法*

肖 庆[1+], 宫云战[1], 杨朝红[1,2], 金大海[1], 王雅文[1]

[1](北京邮电大学 网络与交换技术国家重点实验室,北京 100876)

[2](装甲兵工程学院 信息工程系,北京 100072)

Path Sensitive Static Defect Detecting Method

XIAO Qing[1+], GONG Yun-Zhan[1], YANG Zhao-Hong[1,2], JIN Da-Hai[1], WANG Ya-Wen[1]

[1](State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

[2](Department of Information Engineering, Academy of Armored Force Engineering, Beijing 100072, China)

+ Corresponding author: E-mail: 2722976@qq.com

Xiao Q, Gong YZ, Yang ZH, Jin DH, Wang YW. Path sensitive static defect detecting method. *Journal of Software*, 2010,21(2):209−217. http://www.jos.org.cn/1000-9825/3782.htm

Abstract: This paper presents a new path sensitive algorithm for static defect detecting running in polynomial time. In this method, property state conditions are represented by abstract domain of variables, and infeasible paths can be identified when some variables' abstract value range is empty. This method avoids the combination explosion of full path analysis by merging the conditions of identical property state at join points in the CFG (control flow graph). This algorithm has been implemented as part of a defect testing tool called DTS (defect testing system). Practical test results show that this method can reduce false positive.

Key words: defect detecting; static analysis; path sensitive; dataflow analysis; program analysis

摘 要: 提出一种多项式复杂度的路径敏感静态缺陷检测算法.该方法采用变量的抽象取值范围来表示属性状态条件,通过属性状态条件中的变量抽象取值范围为空来判断不可达路径.在控制流图(control flow graph,简称 CFG)中的汇合节点上合并相同属性状态的状态条件,从而避免完整路径上下文分析的组合爆炸问题.该算法已应用于缺陷检测系统 DTS(defect testing system).实际测试结果表明,该方法能够减少误报.

关键词: 缺陷检测;静态分析;路径敏感;数据流分析;程序分析

中图法分类号: TP311 **文献标识码**: A

软件代码中的缺陷是导致软件故障和漏洞问题的重要原因.基于缺陷的软件测试技术可以分为动态检测技术和静态检测技术.静态检测技术不运行被测程序,主要通过各种静态分析方法来发现程序中的缺陷.从可计算性理论的角度来看,静态分析是一个不可判定问题.Rice定理[1]表明,静态分析不能完美地确定一般程序的任何非平凡属性.静态分析的不可判定性实际上意味着任何自动化的静态分析系统,针对一个程序的非平凡属性

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2007AA010302, 2009AA012404 (国家高技术研究发展计划(863))

Received 2009-06-11; Revised 2009-09-11; Accepted 2009-12-07

(例如是否存在运行时错误),不可能做到既是可靠的又是完备的.可靠的(sound)静态分析意味着,如果分析结果没有报告某类运行时错误,则程序中肯定不存在某类运行时错误,也就是说没有漏报(false negative).完备的(complete)静态分析意味着,如果分析结果报告了某类运行时错误,则程序中肯定存在某类运行时错误,也就是说没有误报(false positive).大量的误报会使人对分析工具失去信心.

静态分析的方法有很多,从路径抽象和近似的角度可以划分为路径敏感(path-sensitive)方法和路径不敏感方法(path-insensitive).路径敏感方法考虑分支间的组合关系,能够区分控制流图上的不同路径信息.与路径敏感方法相比,路径不敏感方法由于分析更加粗糙将会引入更多的误报.最直接和详细的路径敏感分析方法是完整路径分析.完整路径分析准确记录每条不同路径上的"程序执行状态",将导致指数复杂度甚至无限的状态空间.本文提出了一种多项式复杂度的路径敏感算法,并将其应用于缺陷检测系统 DTS(defect testing system).实际测试结果表明,该方法能够减少误报的产生.本文首先给出一个简单的例子说明路径不敏感分析将导致误报.然后,详细讨论数据流分析的 3 种典型解:IDEAL,MOP(meet over all paths)和 MFP(maximal fixed point),指出不可达路径判断不准确和数据流信息"过早"聚合是导致路径分析精确性损失的原因.本文采用变量的抽象取值范围来表示属性状态条件,通过属性状态条件中的变量抽象取值范围为空来判断不可达路径,并基于相同属性状态的属性状态条件合并提出了一种多项式复杂度的路径敏感算法.最后,给出了 10 个大型开源程序的测试数据和结论.

1 一个误报的例子

程序的"时序安全属性(temporal safety properties)"是描述"某些坏的事情不会发生"的一类属性.通常,时序安全属性规定了一系列有序的事件,要求在程序中不能发生这些事件.如果静态分析工具在程序中发现了违反该安全属性的情况,则报告一个反例(counter example)[2].例如,资源申请后必须释放,否则将造成一个资源泄漏缺陷.该属性可用有限状态机来描述,如图 1 所示.

对于路径不敏感分析,由于无法获取精确的路径上下文信息,静态分析工具在实际检测过程可能会产生误报.图 2 为一个可能会产生误报的例子.

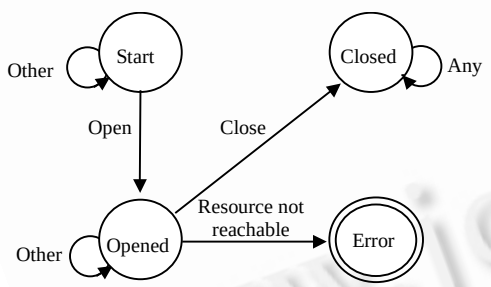

Fig.1 Finite state machine of resource leak property

图 1 资源泄漏属性有限状态机

```
void foo (boolean dump){
        FileOutputStream f=null;
        If (dump)
                f=new FileOutputStream("test.txt");   /* open */
        if (p)
                x=0;
        else
                x=1;
L1:     if (dump)
                f.close();   /* close */

L2:     }
```

Fig.2 An example of false positive

图 2 一个产生误报的例子

上例中,路径不敏感分析不考虑不同的路径信息,属性状态机在 $L1$ 位置上可能的状态集合为{start,opened},最终在 $L2$ 位置处的可能状态集合为{start,opened,error},报告一个"反例",这是一个误报.

从函数入口到达程序 $L1$ 位置共有 4 条可达路径,路径敏感分析将分别跟踪这 4 条路径的路径信息.路径信息可由变量取值表示,则在 $L1$ 位置,属性状态机沿不同路径的可能到达的状态集合为

{{start:*dump*=false,*p*=true,*x*=0,*f*=null},{start:*dump*=false,*p*=false,*x*=1,*f*=null},
{opened:*dump*=true,*p*=true,*x*=0,*f*=notnull},{opened:*dump*=true,*p*=false,*x*=1,*f*=notnull}}.

从 $L1$ 到 $L2$ 的条件判断为 *dump*,由于状态集合中到达 start 状态的两条路径都要求 *dump*=false,因此 start 状态将无法传递到 $L1$ 处判断条件的真分支中.同理,opened 状态将无法传递到假分支中.变量取值矛盾表明,该状态在一条不可达路径上传递,该状态应该从可能状态集合中删除.则在 $L2$ 位置,属性状态机可能到达的状态

集合为

{{start:*dump*=false,*p*=true,*x*=0, *f*=null},{start:*dump*=false,*p*=false,*x*=1, *f*=null},
{closed:*dump*=true,*p*=true,*x*=0, *f*=notnull},{closed:*dump*=true,*p*=false,*x*=1, *f*=notnull}}.

上例表明,路径敏感方法由于可以消除不可达路径,从而减少了误报的产生.

2 路径不敏感分析

传统的基于迭代求解的数据流分析方法是路径不敏感的.数据流问题是一个元组$\langle L,\wedge,F,G,FM\rangle$[3]:

1. L 为需要传播和计算的值集合.
2. $\wedge$为 L 上定义的聚合操作,$\langle L,\wedge\rangle$形成一个半格,包含$\perp$和$\top$,$\wedge$操作用于控制流汇合节点上将不同分支的值进行聚合.
3. F 为 L 到 L 上的单调转换函数集合,代表程序语句对值的影响.
4. $G=(V,E,entry,exit)$为控制流图,V 为节点集合,E 为边的集合,$entry$ 和 $exit$ 分别为控制流图中的唯一入口和唯一出口.
5. $FM:E\to F$ 将 E 中的边映射到 F 中的转换函数.

映射FM也能被扩展到路径上,对于G中路径$p=[e_0,e_1,\ldots,e_n]$,则

$$f_p:L\to L:\ f_p=FM(p)=FM(e_n)\circ FM(e_{n-1})\circ\ldots\circ FM(e_0),FM(e_0),FM(e_1),\ldots,FM(e_n)\in F,$$

因此,f_p仍为单调函数.

数据流问题存在不同的解,其中典型的解包括:

1. IDEAL:理想解,也称真实解.对程序入口到某程序点的所有实际可执行路径的尾端值取聚合(meet)操作,则得到理想解.求理想解是一个不可判定问题.
2. MOP(meet over all paths):称为全路径聚合解.对程序入口到某程序点的所有路径的尾端值取聚合操作,则得到 MOP 解.
3. MFP(maximal fixed point):最大不动点解.所谓的最大不动点解是对控制流图上节点的数据流方程进行不断迭代最终收敛而得到的解.例如,前向(forward)可能(may)数据流计算方程有如下形式:

$$In(s)=\bigcup_{s'\in pred(s)}Out(s'),$$

$$Out(s)=Gen(s)\cup(In(s)-Kill(s)).$$

传统的基于迭代方法求得的即为MFP解.求MFP解有较低的复杂度,通常在实际中先对控制流节点排序再进行计算,可以认为其复杂度在$O((N+E)H)$~$O((N+E)NH)$之间[3].其中,N为节点数,E为边数,H为$\langle L,\wedge\rangle$的高度.数据流分析的MFP解可以认为是完整路径分析的一个不精确的保守近似,虽然避免了路径组合爆炸问题,但却牺牲了路径敏感性.MFP,MOP,IDEAL三者之间的关系为MFP≤MOP≤IDEAL,其中,≤关系代表后者更加精确(前者更加保守).当F中的转换函数满足分配性时,MFP=MOP.对上述 3 个解之间关系的直观理解如图 3 所示.

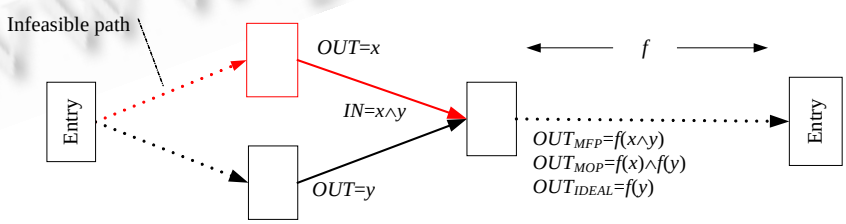

Fig.3 Comparison of three kinds of solution of data flow problem

图 3 数据流问题的 3 个解的对比

MFP 和 MOP 的差异在于,MFP 在控制流汇合节点将不同分支的数据流信息“过早”地进行聚合;而 MOP 将不同路径进行分别考虑,在最后才将不同路径的数据流信息进行聚合.因为 L 为半格,有$(x\wedge y)\le x,(x\wedge y)\le y$;又$f$为单

调函数,则 $f(x\wedge y)=f(x\wedge y)\wedge f(x\wedge y)\le f(x)\wedge f(y)$,即 MFP≤MOP.当 f 满足分配率时,$f(x\wedge y)=f(x)\wedge f(y)$,即 MFP=MOP.

MOP 和 IDEAL 的差异在于,MOP 考虑了控制流图上的所有路径;而 IDEAL 只针对那些实际执行路径,将不可达路径排除在外.如图 3 所示,MOP 解为 $f(x)\wedge f(y)$,IDEAL 解为 $f(y)$.因为 L 为半格,所以有 $f(x)\wedge f(y)\le f(y)$,即 MOP≤IDEAL.

3 DTS 采用的路径敏感分析方法

下面讨论的属性都属于时序安全属性,并采用有限状态机描述.DTS 缺陷检测将属性状态机应用于程序分析过程中,计算每个程序位置上状态机的可能属性状态集合,如果可能属性状态集合中包含 error 状态,则报告一个可能缺陷.

如果把可能属性状态集合看作数据流要分析的值,可能属性状态集合上的并运算为数据流的聚合运算,则图 1 所示的属性检查问题即为一个数据流问题,其目标为得到该问题的 IDEAL 解.而经过前述数据流问题的讨论可知,精确求 IDEAL 解不现实,只能考虑近似保守解.基于迭代的数据流分析得到的 MFP 解与 IDEAL 解相比,其精确性的损失有两个来源:1) 不可达路径判断不准确;2) 对于不满足分配率的转换函数来说,在控制流汇合节点将不同分支的数据流信息“过早”地进行聚合.因此,提高数据流分析的精确性,可以从这两个角度进行考虑.

为了既能减少误报又避免路径组合指数爆炸问题,本文提出一种基于数据流分析的路径敏感分析算法.我们先介绍一些概念:

定义 1. 在程序 P 的执行过程中,程序的执行状态可由二元组$\langle \ell,\rho\rangle$表示,其中,ℓ代表当前的程序执行位置,ρ代表该状态下的环境,程序环境记录了程序中当前每一个变量 X 的值.在环境ρ下,变量 X 的取值记作$\rho(X)$.

我们采用变量的抽象取值区间来表示变量的可能取值范围[4].

定义 2. 程序通过路径 S 执行到位置ℓ,则 S 上的谓词和赋值操作对ℓ处环境ρ中各变量的可能取值范围进行了限定.我们将 S 在ℓ处限定的变量取值范围集合称作 S 在ℓ处的路径条件,记作 $R(S,\ell)$.

例如,在图 4(a)中,对于*dump*和*flag*==1 均取真值的路径S_1,在$L1$ 处的路径条件为

{*dump*[true], *flag*[1,1], *f* [notnull]}.

```
    f=null;
    if (dump){
        f=open(...);
        flag=1;
    }
    else
        flag=0;
L1: if (flag==1)
        close(f);
```

(a)

```
    f=null;
    if (dump)
        f=open(...);
    if (dump)
        flag=1;
    else
        flag=0;
    if (flag==1)
        close(f);
```

(b)

```
    f=null;
    if (dump)
        flag=1;
    else
        flag=0;
L1: if (dump)
        f=open(...);
L2: if (flag==1)
        close(f);
L3:
```

(c)

Fig.4 Three examples[5]

图 4 3 个例子[5]

定义 3. 程序通过路径 S 执行到位置ℓ,属性状态机的状态沿 S 进行传递和变化.在ℓ处到达状态σ,将 $R(S,\ell)$记录在σ上,称为属性状态条件,包含条件的属性状态表示为$\{\sigma:R(S,\ell)\}$.

下面讨论的属性状态都是包含条件的,例如,在图 4(a)中,对于*dump*和*flag*==1 均取真值的路径S_1,在$L1$ 处资源泄漏状态机的属性状态为{opened:*dump*[true],*flag*[1,1],*f*[notnull]}.

属性状态条件是到达该属性状态的相关变量可能取值集合.它标记了能够到达当前属性状态的程序环境信息.属性状态条件可用于不可达路径判断.

对属性状态条件的不同处理方式对应不同的复杂度:

1) 如果所有属性状态条件都取空集,该方法就退化为针对可能属性状态集合的传统数据流分析,虽然具有较低的复杂度,但会有较高的误报.

2) 如果记录属性状态条件,并在所有控制流汇合节点上都不允许属性状态条件合并,该方法就成为变相的完整路径分析.

3) 记录属性状态条件,在所有控制流汇合节点上,将相同属性状态的属性状态条件进行合并是前两种方法的折衷.实际上,程序员在编程时经常通过谓词条件来标记状态信息,选择以属性状态为单位合并路径信息符合程序员的编程习惯[5].因此,该方法具有较低的复杂度,也能降低误报.DTS即采用该方法.下面用该算法来分析图4所示的3个例子的资源泄漏缺陷检测.

考虑图4(a),能够到达程序*L*1位置的可能属性状态集合为

{{opened:*dump*[true], *flag*[1,1], *f* [notnull]},{start:*dump*[false], *flag*[0,0], *f* [null]}}.

*L*1位置判断条件的真分支对变量*flag*限定取值范围为[1,1],假分支对变量*flag*限定取值范围为[−∞,0]∪[2,+∞].将真假分支对*flag*变量的限定范围分别与到达*L*1位置前各状态的状态条件中的*flag*取值范围进行交运算:

$$[1,1]\cap[1,1]=[1,1] \tag{1}$$

$$[1,1]\cap[0,0]=\varnothing \tag{2}$$

$$([-\infty,0]\cup[2,+\infty])\cap[1,1]=\varnothing \tag{3}$$

$$([-\infty,0]\cup[2,+\infty])\cap[0,0]=[0,0] \tag{4}$$

其中,公式(2)和公式(3)得到∅,表明对应的属性状态条件与当前判断谓词取值限定相矛盾,即意味着该属性状态已经历的路径和当前分支组合后为不可达路径.图4(b)中的情形与图4(a)类似.属性状态条件信息可以用于排除不可达路径,从而降低误报.但是考虑图4(c)情况,程序*L*1位置可能属性状态集合为

{{start:*dump*[true,false], *flag*[0,1], *f* [null]}}.

程序*L*2位置可能属性状态集合为

{{start:*dump*[false], *flag*[0,1], *f* [null]},{opened:*dump*[true], *flag*[0,1], *f* [notnull]}}.

*L*2位置的判断条件为*flag*==1,将其真假分支对*flag*的限定范围与上述状态的状态条件中的*flag*取值范围进行交运算,然后考虑状态转换.最终,在*L*3处可能属性状态集合为

{{start:*dump*[false], *flag*[0,1], *f* [null]},{error:*dump*[true], *flag*[0,0], *f* [notnull]},
{closed:*dump*[true], *flag*[1,1], *f* [notnull]}}.

这会造成一个误报,其原因是,属性状态条件记录的是所有可能到达该属性状态的相关变量取值范围集合,而程序中可能存在不同的路径到达当前属性状态,属性状态条件合并了这些路径上的路径条件信息.如前所述,在控制流汇合节点将不同分支的数据流信息“过早”地进行聚合会导致不精确.图4(c)中到达*L*1位置前,真假分支传递来的都为start状态,属性状态条件合并不同分支上的路径条件信息(*dump*[true,false], *flag*[0,1], *f* [null]),该属性状态条件丢失了*dump*和*flag*间的组合约束关系,造成后续不可达路径判断不准确形成误报.避免该误报的方法即设法阻止控制流汇合节点上不同分支的数据流信息“过早”地进行聚合,但这需要以提高分析复杂度为前提.

在所有控制流汇合节点上允许相同状态的条件进行合并的数据流迭代算法如下:

算法 1. DTS路径敏感分析算法.

in[*n*]:节点*n*执行之前的可能属性状态集合.

out[*n*]:节点*n*执行之后的可能属性状态集合.

kill[*n*]:节点*n*删除或转换了的可能属性状态集合.

gen[*n*]:节点*n*由于转换而得到的新可能属性状态集合.

entry:控制流入口节点.

pred(*n*):节点*n*的前驱集合.

merge:汇合节点集合.

输入:控制流图和采用状态机描述的待测属性.

输出:每个控制流节点的可能属性状态集合.

for each $n \in$ N do *in*[*n*] := ∅;

change := true;
while *change* do begin
  change := false;
  for each $n \in$ N do begin
    if $n = entry$ then
      $in[n] = \{\text{start}\}$;
    else
      $in[n] := \bigcup_{p \in pred(n)} out[p]$;
    if $n \in merge$ then
      $in[n] = \left\{ \sigma : \bigcup_{\sigma_i = \sigma} R_i \mid \sigma, \sigma_i \in in[n] \right\}$;
    $oldout := out[n]$;
    $out[n] = gen[n] \cup (in[n] - kill[n])$;
    if $out[n] \neq oldout$ then *change* := true
  end
end

允许相同状态的条件进行合并的方法的复杂度可进行如下估计:假设控制流图中节点个数为N,边数为E,属性状态机状态数为D,程序中相关变量个数为V,用于表示变量取值的偏序抽象语义域上基本操作(交、并、补、判相等,判是否矛盾等操作)的最大复杂度为Q.内层for循环计算复杂度由下述因素共同决定:节点数和边数、每个节点可能出现的最多状态数D、状态条件中包含的变量数V、表示变量取值的偏序抽象语义域上基本操作最大复杂度Q.因此,内层for循环的最大复杂度为$O((N+E)DVQ)$.外层while循环的终止条件为所有节点的$out[n]$集合不发生变化,而while循环的每次迭代只能使$in[n]$和$out[n]$集合变大或者其状态条件发生变化,假设用于表示状态条件的各抽象语义域的最多增大次数为H,则while循环的迭代次数上限为$NDVH$.综上所述,算法在最坏情况下复杂度为$O((N+E)ND^2VQH)$.在实际中采用对节点先进行排序的方法进行计算,实际中的属性状态机状态个数D为常量且通常不超过 10 个,因此,在实际中该算法的复杂度可以认为在$O((N+E)VQH)$~ $O((N+E)NVQH)$之间.

4 实验结果

为了分析路径敏感算法消除误报的效果,我们针对路径不敏感(传统数据流分析)和本文提出的路径敏感方法进行缺陷检测对比实验 1.分析扫描的对象为 10 个大型 Java 开源软件(选取标准为 sourceforge 排名靠前且能编译通过),扫描的目标为资源泄漏和空句柄引用这两类缺陷.我们对扫描结果进行了人工确认,实验结果见表 1.

Table 1 Results of comparison experiment 1

表 1 对比实验 1 结果

Program	Number of files	Number of lines	Number of confirmed defects	Path-Insensitive method			DTS' path-sensitive method			Number of reduced false positives
				Time (s)	Number of reported defects	Number of false positives	Time (s)	Number of reported defects	Number of false positives	
areca-7.1.1	426	68 090	43	106	67	24	112	64	21	3
aTunes-1.8.2	306	52 603	20	55	22	2	52	22	2	0
Azureus_3.0.5.2	2 720	575 220	129	802	363	234	880	340	211	23
cobra-0.98.1	449	70 062	3	97	12	9	99	11	8	1
freecol-0.7.3	343	110 822	109	92	121	12	99	121	12	0
freemind-0.8.1	509	102 112	47	351	76	29	369	69	22	7
jstock-1.0.4	165	38 139	26	168	35	9	180	33	7	2
megamek-0.32.2	535	212 453	137	256	249	112	309	227	90	22
robocode-1.6	233	53 408	12	58	53	41	62	30	18	23
SweetHome3D-1.8	154	59 943	17	114	31	14	122	29	12	2
Total	5 840	1 342 852	543	2 099	1 029	486	2 284	946	403	83

采用路径不敏感方法总体分析时间为 2 099s,路径敏感方法总体分析时间为 2 284s,总体分析时间增加了8.81%.采用路径不敏感方法的误报数为 486,采用路径敏感方法的误报数为 403.与路径不敏感方法相比,本文描述的路径敏感方法排除了总体误报数的(486−403)/486×100%=17.08%.从上述结果可以看出,与路径不敏感分析算法相比,本文提出的路径敏感分析算法只增加了较少的分析时间,但能够有效地减少误报.

Das等人提出了一种在属性状态上增加程序执行符号状态信息,并利用这些执行符号状态信息来排除不可达路径的多项式复杂度路径敏感方法.在他们实现的工具ESP中,采用常量传播格来表示执行符号状态信息[5].常量传播格是一种表示变量取值信息的简单而粗略的方法.而DTS采用抽象取值范围来表示变量取值信息[4],可以认为是常量传播格的精化.我们针对常量传播格表示和抽象取值范围表示进行了缺陷检测对比实验 2,分析扫描对象及目标与实验 1 相同,实验结果见表 2.

Table 2 Results of comparison Experiment 2

表 2 对比实验 2 结果

Program	Number of files	Number of lines	Number of confirmed defects	Constant propagation lattice			Abstract value range			Number of reduced false positives
				Time (s)	Number of reported defects	Number of false positives	Time (s)	Number of reported defects	Number of false positives	
areca-7.1.1	426	68 090	43	110	64	21	112	64	21	0
aTunes-1.8.2	306	52 603	20	55	22	2	52	22	2	0
Azureus_3.0.5.2	2 720	575 220	129	862	344	215	880	340	211	4
cobra-0.98.1	449	70 062	3	100	11	8	99	11	8	0
freecol-0.7.3	343	110 822	109	102	121	12	99	121	12	0
freemind-0.8.1	509	102 112	47	359	71	24	369	69	22	2
jstock-1.0.4	165	38 139	26	174	34	8	180	33	7	1
megamek-0.32.2	535	212 453	137	278	227	90	309	227	90	0
robocode-1.6	233	53 408	12	60	30	18	62	30	18	0
SweetHome3D-1.8	154	59 943	17	114	29	12	122	29	12	0
Total	5 840	1 342 852	543	2 214	953	410	2 284	946	403	7

采用常量传播格表示比采用抽象取值范围表示多了 7 个误报.我们分析了这 7 个误报以后发现,其主要发生在条件谓词中包含数值型变量情况下.例如下例:

文件:Azureus_3.0.5.2_source\org\gudy\azureus2\ui\swt\views\configsections\ConfigSectionFile.java

缺陷类型:空句柄引用

```
…
194:    BooleanParameter zeroNew=null;
…
198:    if ( userMode>0) {
199:       //zero new files
200:       zeroNew=new BooleanParameter(gFile, sCurConfigID,
201:                                                    “ConfigView.label.zeronewfiles”);
…
205:    }
…
222:    if (userMode>0) {
…
232:       zeroNew.setAdditionalActionPerformer(new ExclusiveSelectionActionPerformer(btnIncremental));
…
```

采用常量传播格表示变量取值信息,在第 232 行将会报告一个空句柄引用缺陷,这是一个误报.*userMode*>0对变量取值的限定是一个区间范围,而用常量传播格表示其取值信息为⊤,丢失了精度.

5 相关工作

静态缺陷检测过程中引起误报的原因有很多,从数据流分析上考虑,其主要原因有两个:1) 不可达路径判断不准确,导致不精确;2) 在控制流汇合节点将不同分支的数据流信息“过早”地进行聚合,导致不精确.其中,前者代表IDEAL和MOP的差距,后者代表MFP和MOP的差距.不同学者提出了许多尝试提高数据流分析精度的方法,其中尝试改进MFP和MOP的差距的包括:Bodik和Anik提出一种值重命名方案,通过在数据流分析过程中综合值名称空间来提高数据流分析的精度,这实际上是针对那些转换函数不满足分配率的数据流分析,试图通过MFP分析得到MOP信息[6];Ammons和Larus提出一种从控制流图中分离出一个有限路径集合进行数据流分析的方法,通过该方法来提高数据流分析的精度,其本质上是通过减少分析路径数求MOP解,以提高精度[7];Thakur和Govindarajan首先分析哪些控制流汇合节点会降低数据流分析精度,然后通过重构这些节点处的控制流图,最后在重构的控制流图上进行分析以提高精度,其本质上也是试图通过MFP分析得到MOP 信息[8].

尝试改进IDEAL和MOP差距的包括:Holley和Rosen提出一种通过增加一个有限的谓词集合来提高数据流分析精度的方法,将控制流上的每条边与该谓词集合上的一个关系相关联,如果某条路径的关系合成得到空集,则被认为是不可达的[9];Bodik等人提出一种低代价的基于检测静态边关联的查找不可达路径方法,用于提高定义使用分析的精度[10];Tu和Padua通过在SSA汇合节点上增加控制谓词,提出一种广泛性的SSA方法来提高数据流分析精度[11];Das等人在模型检查的属性状态上增加程序执行符号状态信息,通过跟踪属性状态和程序执行符号状态间的关联关系,并将其用于排除不可达路径来提高数据流分析的精度[5];Fischer等人通过在数据流分析半格中元素上增加谓词信息来提高数据流分析精度.实际上,这些谓词将程序路径集合进行了划分,从另一个角度也可以认为是记录了不同的路径信息,该信息可用于不可达路径判断.谓词的选取基于一种反例导向的抽象精化技术(counterexample guided abstract refinement)[12].

在上述方法中,Das[5]方法与我们的方法相似.但是,我们引入了抽象解释思想,采用变量的抽象取值来表示属性状态条件,不可达路径就体现为属性状态条件中某个变量抽象取值范围为空,属性状态条件相关计算更加精确、灵活.

6 小 结

本文分析了路径不敏感造成静态分析不精确的原因,并提出了一种多项式复杂度的路径敏感分析方法.该方法采用变量的抽象取值范围来表示属性状态条件;通过属性状态条件中变量取值为空来判断不可达路径;通过在控制流汇合节点上进行相同属性状态的属性状态条件合并来降低计算复杂度.通过对 10 个大型 Java 开源项目的分析,表明该方法能够减少误报.下一步工作包括:研究选择哪些控制流汇合节点和哪些属性状态进行属性状态条件合并,以更好地求得复杂度和精度的平衡;当前属性状态条件中记录了相关变量的取值范围,实际上并不是所有变量都会影响后续的不可达路径判断;研究如何减少属性状态条件中变量数可以使算法的时间和空间开销更小.

References:

[1] Rice HG. Classes of recursively enumerable sets and their decision problems. Trans. of the American Mathematical Society, 1953, 74(2):358−366.

[2] Ball T, Rajamani SK. Automatically validating temporal safety properties of interfaces. In: Dwyer M, ed. Proc. of the 8th Int’l SPIN Workshop on Model Checking of Software. Berlin, Heidelberg: Springer-Verlag, 2001. 103−122.

[3] Aho AV, Lam MS, Sethi R, Ullman JD. Compilers Principles, Techniques, and Tools. 2nd ed., New York: Addison-Wesley, 2006. 626−632.

[4] Yang ZH, Gong YZ, Xiao Q, Wang YW. The application of interval computation in software testing based on defect pattern. Journal of Computer-aided Design & Computer Graphic, 2008,20(12):1630−1635 (in Chinese with English abstract).

[5] Das M, Lerner S, Seigle M. ESP: Path-Sensitive program verification in polynomial time. In: Knoop J, Hendren LJ, eds. Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation. New York: ACM Press, 2002. 57−68.

[6] Bodik R, Anik S. Path-Sensitive value-flow analysis. In: MacQueen DB, Cardelli L, eds. Proc. of the 25th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. San Diego: ACM Press, 1998. 237−251.

[7] Ammons G, Larus JR. Improving data-flow analysis with path profiles. In: Berman AM, ed. Proc. of the ACM SIGPLAN '98 Conf. on Programming Language Design and Implementation. New York: ACM Press, 1998. 72−84.

[8] Thakur A, Govindarajan R. Comprehensive path-sensitive data-flow analysis. In: Soffa ML, Duesterwald E, eds. Proc. of the 6th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization. New York: ACM Press, 2008. 55−63.

[9] Holley LH, Rosen BK. Qualified data flow problems. In: Abrahams P, Lipton R, Bourne S, eds. Proc. of the 7th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. New York: ACM Press, 1980. 68−82.

[10] Bodik R, Gupta R, Soffa PL. Refining data flow information using infeasible paths. In: Jazayeri P, Schauer H, eds. Proc. of the Software Engineering Notes ESEC/FSE'97. New York: ACM Press, 1997. 361−377.

[11] Tu P, Padua D. Gated SSA-based demand-driven symbolic analysis for parallelizing compilers. In: Valero M, ed. Proc. of the 1995 ACM Int'l Conf. on Supercomputing. New York: ACM Press, 1995. 414−423.

[12] Fischer J, Jhala R, Mujumdar R. Joining data flow with predicates. In: Wermelinger M, Gall HC, eds. Proc. of the 13th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Lisbon: ACM Press, 2005. 227−236.

附中文参考文献:

[4] 杨朝红,宫云战,肖庆,王雅文.基于缺陷模式的软件测试中的区间运算应用.计算机辅助设计与图形学学报,2008,20(12):1630−1635.

肖庆(1979－),男,湖南祁东人,博士生,讲师,主要研究领域为软件测试,程序分析.

金大海(1974－),男,博士,主要研究领域为软件测试.

宫云战(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试,软件工程.

王雅文(1983－),女,博士生,CCF 学生会员,主要研究领域为软件测试,程序分析.

杨朝红(1976－),男,博士,副教授,CCF 高级会员,主要研究领域为软件测试,软件工程.