基于动态协同双向映射的相似执行路径生成方法

郭 曦[1],王 盼[2,3]

(1.华中农业大学信息学院计算机科学系,湖北武汉 430070;2.武汉电力职业技术学院,湖北武汉 430079;
3.武汉大学电气工程学院,湖北武汉 430072)

摘 要: 相似执行路径的生成是代码分析和检测的基础性工作之一,现有的方法通常以程序的行为序列或结构为分析对象,通过改变关键谓词的取值等方法来进行分析,但由于缺乏必要的引导信息导致生成的相似路径的有效性较低,另外由于路径的谓词集合较长而难以求解也降低了分析的精度.提出基于动态协同双向映射的分析方法,通过对程序控制流图的表示形式进行扩展,结合后向符号分析的方法生成候选路径的最弱前置条件,并以此为引导信息使用编辑距离的方法通过改变距离因子的取值来生成有针对性的相似路径集合.实验结果表明,与现有的方法相比,该方法的准确性和效率有明显的优势.

关键词: 静态分析; 控制流图; 最弱前置条件; 相似执行路径

中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2014)11-2168-06

电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2014.11.007

A Technique of Similar Execution Paths Generation Based on Dynamic Synergy Bidirectional Mapping

GUO Xi[1], WANG Pan[2,3]

(1. *Department of Computer Science, College of Informatics, Huazhong Agriculture University, Wuhan, Hubei* 430070, *China*;
2. *Wuhan Electric Power Technical College, Wuhan, Hubei* 430079, *China*;
3. *School of Electrical Engineering, Wuhan University, Wuhan, Hubei*, 430072, *China*)

Abstract: Similar execution paths generation is one of the fundamental tasks in code analysis and detection. The current methods usually target to the program behavior or program structure, and change the value of key predicates, but these methods has a low effectiveness due to the lack of the necessary guidance information, Meanwhile, the predicates set has a large size and usually hard to solve, thus it will reduce the analyze precision as well. A technique of similar execution paths generation based on dynamic synergy bidirectional mapping is proposed in this paper. According to extend the shape of Control Flow Graph and use the backward symbolic analysis, the weakest precondition of the candidate path is generated, which can be used as the guidance information to generate pointed similar execution paths set according to the edit distance via changing the distance factor. The experimental results show that this method has the advantage of precision and anti-inference.

Key words: static analysis; control flow graph; weakest precondition; similar execution path

1 引言

为了保证软件开发的质量,软件测试作为必要的阶段约占软件开发和维护成本的 60%左右,其中程序调试是最耗时、代价最为昂贵的任务之一.程序执行路径分析是调试过程中的发掘程序行为特征的重要方法,该过程需要理解程序的功能、结构、语义以及执行路径的特点,故高效地分析程序执行路径并构建程序的行为特征对于提高软件产品的安全性具有重要意义.

程序分析中的一个重要问题是路径可行性判定问题[1],即通过使用符号执行[2]和约束求解等分析方法生成初始变量的取值范围,使得程序能够沿着指定的路径执行,同时对于不能验证的属性,通过生成反例的方法来判断是否存在对应的执行路径.由于这些分析工具通常使用抽象的分析方法生成反例程序,故源代码的抽象程度会对反例生成的效果产生直接的影响.最弱前置条件[3]能够分析程序的执行语义并建模,是一种后向符号分析方法,能够减小抽象操作对分析精度造成的影响.但由于大规模程序的路径约束条件的解空间与程序分支呈指数关系,容易产生状态空间爆炸的问题,从而对

收稿日期:2014-02-10;修回日期:2014-03-14;责任编辑:马兰英
基金项目:国家自然科学基金(No.61103220, No.91118003);湖北省自然科学基金(No.2014CFB144)中央高校基本科研业务费专项资金(No.0900206154);武汉大学搏士短期出国(境)研修专项经费

程序性质的分析带来干扰.

上述问题导致程序路径分析过程中容易产生不可达或者冗余的路径,同时最弱前置条件在计算过程中需要对循环变量进行计算,对程序可能陷入死循环的状态的情况进行预处理.目前的路径生成方法由于缺乏必要的路径引导信息,导致生成较多无效或者冗余的路径集合,故需要对路径条件和检测点之间关联关系进行深入研究.本文针对以上问题,以程序路径的可满足性为研究对象,提出动态协同双向映射的路径生成方法,具体的过程是:首先以程序的控制流图中的循环结构为分析对象,对循环体中的节点设置相互正交的标签变量以区分执行路径在循环提中的节点序列,使得每条执行路径都具有惟一的路径编码;然后在距离因子 k 的作用下,计算指定节点的最弱前置条件,不断回溯到控制流图的起始节点,在该过程中通过对路径编码在标签变量的作用下进行逆运算,以还原执行路径在循环结构中的节点执行序列,最后通过符号执行工具在最弱前置条件的作用下生成与目标执行路径的编辑距离不超过 k 的路径集合.该方法可有效减少在符号执行分析过程中由于缺乏路径引导信息而导致的死循环问题,可更加高效地生成有针对性的路径集合.

2　相关工作

程序执行路径的生成与程序输入有紧密的联系,对于程序 P 及 P 中的一条执行路径 l,设 P 的输入空间为 S,路径生成问题为:对于程序输入 $i\in S$,i 对应的测试用例 t 在 P 中所生成的执行路径.程序执行路径的生成可以转化为测试数据的问题,由于通过人工的方式构建测试数据的效率较低,目前测试数据生成已经进入自动化时代.其中主要有基于动态的方法和基于静态方法[4],动态的方法需要通过输入数据使目标程序能够实际运行,故该方法生成的测试数据是确定的,主要有线性式分析法[5],该方法将判断语句转化为布尔型的赋值语句,由于该方法在求解非线性的路径约束条件过程中可能会产生局部极值,故存在不完备性.文献[6]采用动态数据流分析的方法来定位分支条件对应的谓词变量,该方法虽然对于线性路径约束条件具有完备性,但是该方法每次只能分析一个分支谓词变量,故需要频繁地进行迭代操作.文献[7]采用松弛迭代法分析输入和谓词函数之间的依赖关系,通过求解建立的输入变量线性方程组而产生输入数据,由于方法采用线性函数来逼近非线性函数,故当谓词中存在非线性函数时,需要进行多次迭代以产生新的输入数据.文献[8]采用进化的分析方法对搜索空间进行缩减以生成路径覆盖的测试数据,但是对于路径的选择仍然采用人工的分析方式.静态分析方法对路径谓词约束进行求解,从而获得输入数据生成问题的全部解.符号执行是最常用的静态分析方法,它通过将符号引入到程序的输入中来建立约束系统,但是符号执行的方法对于循环变量、数组下表等结构不能准确地进行分析,同时由于符号执行在分析过程中需要对复杂的代数进行运算以缩减状态空间,这些都限制了其使用范围.文献[9]通过最弱前置条件引导符号执行的方法来对程序所具有的性质进行验证,但未对产生最弱前置条件进行约简.文献[10]通过最弱前置条件的方法来检测并定位程序中的错误语句,其分析过程与本文方法中的距离因子 $k=0$ 情况类似,由于循环结构的执行次数可以通过改变 k 的取值来调整,故具有更好适应性.

对程序中循环结构的分析是本文方法中需要着重研究的内容.循环结构的分析即程序的终止性判定问题,它从可计算性的角度可归结为停机问题,从而是不可判定的,故不存在一个通用的算法能够检测一个给定的程序中是否存在停机现象.目前的研究工作主要集中在程序的终止性和非终止性分析两个方面.对于终止性分析,常用的方法有循环不变式和 Ranking 函数的方法,包括线性阶和多项式阶函数[11,12],文献[13]提出了抽象精化的方法来分析程序的终止性.对于非终止性分析,文献[14]提出了一种死循环检测方法,但是该方法不能处理诸如嵌套循环等复杂控制结构的程序.文献[15]提出基于 lasso(由前缀和环组成)的方法来分析程序的非终止性,但是该方法需要枚举出程序中的 lasso,同时文中也指出 lasso 不能检测出所有的循环结构.

3　控制流图中循环结构的预处理

定义 1　控制流图(Control Flow Graph,CFG).控制流图是由基本块(Basic Block)和边组成的有向图,每一个基本块被抽象成控制流图中的一个节点.控制流图采用形式化的方法可以表示为 $<N,E,s,f>$.其中 N 是 CFG 中节点的集合,$E\subseteq N\times N$ 是基本块之间边的集合,若程序的执行路径中的节点序列中存在节点 B_i 到 B_j 间的流向,则存在一条边 $e=(B_i,B_j)\in E$,b_{in} 和 b_{out} 分别表示控制流图的入口基本块和出口基本块.

本文所讨论的控制流图具有单一入口和单一出口,对于其它类型的控制流图,可以添加额外的节点以转化为具有单一入口和单一出口节点的控制流图.

定义 2　CFG 路径的可行性.对于节点序列 $s=<n_1,n_2,\cdots,n_k>$,其中 $(n_i,n_{i+1})\in E(1\leqslant i\leqslant k)$,$k=|s|$ 为序列的长度,若 $n_i=s$,$n_k=f$,则该序列是一条完整的执行路径.称路径 l 是可行的,当且仅当存在程序的一组输入,使得程序能够沿着指定的节点序列 s 执行,表示为 $N\times N\rightarrow\{T,F\}$ 的计算过程,否则称 l 是不可行

的路径.

定义 3 路径条件(Path Condition,PC).在控制流图中只可能在分支结构和循环结构的条件判定语句处通过调整谓词的取值产生不同的路径,路径条件是由程序中的分支结构或循环结构对应的约束条件构成的一阶逻辑公式,即分支结构在控制流迭代过程中产生的数据流值.

在 CFG 中,出了入口节点外,其它节点均有入边,即存在从其它节点指向该节点的边.对于循环结构的入口节点,由于存在多条入边,即在满足循环条件的基础上循环体会不断执行从而产生新的路径集合.若 CFG 中某个节点存在多条入边的时候,则表明程序中存在循环结构或者分支结构.本文方法在分析过程中需要对程序中的循环结构进行处理,用以获取循环体中节点的执行顺序和次数.当检测到 CFG 中存在环结构的时候,即程序中有循环体,对于循环体中的每一个节点,同时设置特定的码片向量,同时对每条入边设定不同的标签变量,这样获取的执行路径在循环体中节点的连接序列可以保证不同的循环次数对应于不同的路径编码(执行路径在循环体中尾节点处的值),即可以利用不同的路径编码来区分不同的执行路径.循环体中不同节点的码片向量必须设定为互不相同,且相互正交.本文的方法需要通过回溯以获取程序执行序列对应的输入集合,故在回溯过程中通过路径编码和码片向量进行逆运算以区分节点的不同入边,从而还原程序的执行过程.

设循环体中某一节点的码片向量为 $\boldsymbol{S}$(-1,-1,-1,1,1,-1,1,1),另一节点的码片向量为 $\boldsymbol{T}$(-1,-1,1,-1,1,1,1,-1).由于 $\boldsymbol{S}$ 和 $\boldsymbol{T}$ 是相互正交的两个向量,故其规格化内积为 0:

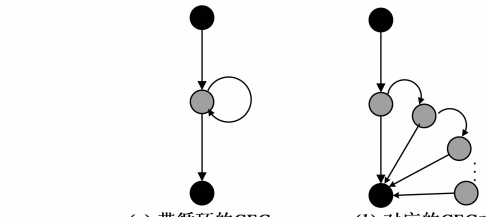
$$\boldsymbol{S}\cdot\boldsymbol{T}\equiv\frac{1}{m}\sum_{i=1}^{m}S_iT_i=0$$

同时对每条入边以二进制的方式设定标签变量,这样当执行序列从一个节点经过另一个节点的时候,采用如下方式进行编码:标签变量中的数据比特依次和该节点码片向量进行乘积操作,当编码位为 1 的时候直接将该码片向量传递给下一个执行路径中的节点;当编码位为 0 的时候,传递的是该码片向量的反码;当执行序列不经过某一条边时,则不传送任何向量.同时,码片向量与自身的规格化内积为 1,而与自身反码的规格化内积为-1,这种性质可以为路径的回溯操作提供依据.

$$\boldsymbol{S}\cdot\boldsymbol{S}=\frac{1}{m}\sum_{i=1}^{m}S_iS_i=\frac{1}{m}\sum_{i=1}^{m}S_i^2=\frac{1}{m}\sum_{i=1}^{m}(\pm1)^2=1$$

$$\boldsymbol{S}\cdot\bar{\boldsymbol{S}}=\frac{1}{m}\sum_{i=1}^{m}S_i\bar{S}_i=\frac{1}{m}\sum_{i=1}^{m}-1=-1$$

当进行回溯操作的时候,通过路径编码和节点的码片向量进行规格化内积操作,即通过计算 CFG 图中边的标签变量还原出程序的执行路径.在 CFG 图中使用节点序列表示程序执行的路径,通过以上方法建立起路径编码和执行路径之间的对应关系,可以区分节点的不同入边和不同的出边.同时由于回溯是从 CFG 的出口基本块开始,故可以通过入边来区分执行路径,且通过一次回溯就可以还原出程序的执行路径.在控制流图中使用带回边的节点集合表示循环结构的执行路径,这些节点构成了一个环结构.图 1 表示了一个带循环结构的控制流图片段,其中原始的 CFG 结构如图 1(*a*)所示,图 1(*b*)为消除了循环结构的控制流图,记为 CFG^{∞}.该方法使得程序的每条执行路径都有惟一的节点序列与之对应,即拥有惟一的路径编码.在实际分析过程中,由于循环的终止性是不可判定的,即在实验过程中程序的执行次数是有限的,即控制流图可以表示为 CFG^n,显然 CFG^n 中的节点集合是 CFG^{∞} 的子集.



(*a*) 带循环的CFG (*b*) 对应的CFG^{∞}

图1 循环结构过程间控制流图及其CFG^{∞}形式

以下给出 CFG 节点的码片向量和边的标签变量插入算法.

算法 1:CFG 节点的码片向量和边的标签变量插入算法

```
Input: CFG, loop
Output: CFG with chip vector and label variables
  label variable of loop_entry←0;
  foreach node n in loop do
    m←number of entry edges of n;
    n←chipVector;
    if m > 1 then
    i←0;
      foreach InEdge ie of n do
      label variable of ie←i;
        i + + ;
      endfor
    endif
  endfor
    CFG with chip vector and label variables
```

由于本文的方法需要对循环体进行抽象操作,同时获取程序的执行路径,故对循环结构的分析精度会对相似执行路径的生成产生直接的影响.循环分析主

要有线性循环分析和非线性循环分析,由于线性循环分析已有较多的研究,故本文主要针对非线性循环的执行条件进行分析.

4　相似路径的生成

定义 4　最弱前置条件(Weakest Precondition, WP).对于程序中的一条语句,若从入口基本块 b_{in} 到该语句是路径可行的,且该语句执行后满足结果断言(即后置条件),最弱前置条件是保证这些条件发生的最小前提条件.WP 由一组谓词公式组成:WP(S, R),其中 S 为 CFG 中语句集合,R 是 S 执行后的结果断言集合.

算法 2 为 CFG 中各节点对应的最弱前置条件的生成过程.对于 CFG 中指定节点对应的语句 s 及其后置条件 φ_{post},通过该算法可以获得该节点的符号状态的集合 ST,并对结果进行合并和约简操作.

算法 2　最弱前置条件生成算法.

```
Input: CFG, φ_post
Output: weakest precondition
1   var F: |Formula| ← Statement;
2   var List: pair of(Statement, Formula);
3   ∀ s∈Statement, F(s) ← ∅;
4   List ← |(ExitNode, φ_post)|;
5   while List ≠ ∅ do;
6     (s', φ_post) ← select from List;
7     while(s, s') ∈ E of CFG do;
8       φ_pre ← merge(F(s), simplify(WP(s, φ_post)));
9       if φ_pre ≠ false then
10          F(s) ← F(s) ∪ φ_pre;
11          List ← List ∪ (s, φ_pre);
12        endif
13    end while
14  endwhile
15  return F(EntryNode)
```

最弱前置条件在计算过程中的两个主要的规则表示如下:

$$\mathrm{WP}(x=e, Q) = Q[e/x];$$

$$\mathrm{WP}(\mathrm{if}(c)\, s_1 \text{ else } s_2, Q) = (c \rightarrow \mathrm{WP}(s_1, Q)) \wedge (\neg c \rightarrow \mathrm{WP}(s_2, Q));$$

由于最弱前置条件具有析取性质,故可以实现诸如 $\mathrm{WP}(s, \varphi_1 \vee \varphi_2) = \mathrm{WP}(s, \varphi_1) \vee \mathrm{WP}(s, \varphi_2)$ 的约简操作,它是状态集合中被选取的条件的析取范式.为了量化程序执行路径之间的相似程度,本文使用计算路径之间编辑距离(edit distance)的方法来生成与目标路径近邻的执行路径集合,同时引入因子 k 调整路径编辑距离的值.对于程序 $Prog$ 所有的可行路径集合 $Paths$,设目标执行路径为 l,则与 l 的编辑距离不超过 k 的路径集合 L 表示如下,显然 $L \subset Paths$,其中函数 D 用来计算编辑距离.

$$L = \{\varepsilon \mid \varepsilon \in Paths \wedge D(l, \varepsilon) \leqslant k\}$$

对于 CFG 中节点的路径条件的集合,它是由待分析语句 s,语句 s 处的路径条件 pc,程序可行路径 $path$ 和距离因子 k 通过笛卡尔乘积经 SMT 求解器计算得出:

$$\{(s, pc, path, k) \in Prog \times PC \times Paths \times k \rightarrow PC\}$$

对 CFG 图最弱前置条件操作后,通过自底向上的回溯过程得到包含入口基本块的路径集合 L,对于此过程所得到的最弱前置条件集合 WP,它是各条回溯路径所得到最弱前置条件的析取:

$$\mathrm{WP}(Prog, PC, Paths, k) = \vee_{l \in L} \mathrm{WP}(l, \varphi)$$

由于距离因子 k 的取值可以控制相似路径的生成,过小的 k 取值因生成的路径数量较少,生成的近邻路径也较少,通过回溯所得到的输入域较实际有效的输入域有较大的局限性;若不断增加 k 的取值,可以生成更多的执行路径,但此时的路径条件集合和最弱前置条件集合也相应地增大,从而对约束求解器的性能提出较高的要求,故需要在路径的规模和分析的精度之间进行权衡.在距离因子 k 的影响下,最弱前置条件的计算如下所示:

$$\begin{cases} \mathrm{WP}(Prog, PC, Paths, k) = \mathrm{WP}(s, pc, path), & k = 0 \\ \lim\limits_{k \to \infty} \mathrm{WP}(Prog, PC, Paths, k) = \mathrm{WP}(Prog, PC), & k \to \infty \\ \mathrm{WP}(Prog, PC, Paths, k) = \vee_{l \in L} \mathrm{WP}(l, \varphi), & k \in (0, \infty) \end{cases}$$

5　实验与分析

本文采用符号执行和近邻最弱前置条件相结合的分析方法来生成相似执行路径,通过为 CFG 中循环结构中的节点加入相互正交的标签变量方式为每一条执行路径构建具有惟一的路径编码作为该路径的标识,通过分析本文方法在处理循环结构和相似路径生成的效率来进行实验,实验的整体分析流程如图 2 所示.

我们以 WALA 作为实验分析平台,并使用其提供的 T.J.Watson Libraries 进行程序的过程内分析.在前端(Front End)的功能主要是进行代码的预处理,通过修改 GCC 语法扫描器中的 parse.y 文件中的语义动作,通过对源文件的扫描生成对应的抽象语法树(AST),并对 AST 的结构进行划分以获取程序的基本块,然后构建程序的调用图与控制流图,在前端还需对程序中的循环结构进行检测和分析,通过循环展开的方法为后端生成检测路径提供必要的信息.后端(Rear End)的主要功能是使用本文的方法对程序进行最弱前置条件分析,并使用符号执行的方法在距离因子的作用下生成待验证的路径集合,通过分析路径的可满足性来检验所产生的路径的可行性.在后端分析过程中使用 CVC3 工具对路径谓词集合进行计算和精化,保留可求解的路径谓词集合.本文使用的符号执行工具是 Java PathFinder,

它以 CVC3 工具生成的路径谓词集合为前置条件引导程序在 CFG 中沿着指定的路径执行以生成有针对性的路径集合.本文的基准测试程序来自 DaCapo[16]和常见的开源 JAVA 程序,其中包含有大量的循环语句,故可以较好地满足本文方法的所需要的分析条件.

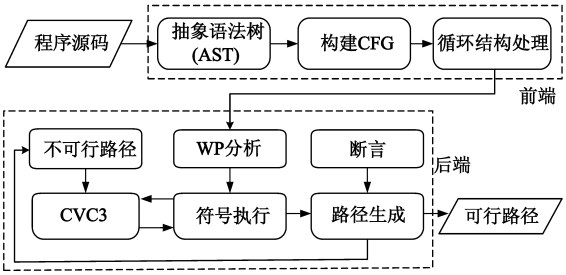


图2 整体分析流程

执行路径的生成主要通过遍历 CFG 的方法生成待检测路径的集合,在此过程中需对循环次数进行分析.虽然可以对程序的循环终止性进行分析,但是在实际的分析过程中往往需要指定循环次数以预防因内存资源耗尽而导致的异常现象,我们可通过调整距离因子 k 的取值来间接控制循环的次数.由于本文的方法是对传统最弱前置条件分析方法的近似,故距离因子 k 的取值对路径的生成有直接影响,即 k 的取值越大,可生成的执行路径也越多,同时时空开销也相应地增长.

在相似路径生成方面,传统的符号执行分析方法由于缺少必要的路径引导信息,从而需遍历整个程序状态空间,这种粗粒度的分析方法往往造成很大的开销,且生成大量不可满足的路径,导致路径可行性的分析精度相对较低.最弱前置条件作为后向符号分析方法从 CFG 图中待分析的节点出发,结合后置条件与本文方法中生成的路径编码进行回溯操作,生成 CFG 中各节点的最弱前置条件.这样符号执行工具在路径生成过程中由于有各节点的最弱前置条件作为引导信息,可高效地生成与目标路径相近邻的可行路径集合.在实验过程中,主要分析的是本文方法在生成相似执行路径时的效率,考察指标为加入距离因子后生成可行路径的效率,主要从循环结构断点数目与 k 值之间的关系,以及可行路径生成的数量的两个方面展开讨论.

现有的循环结构处理方法一般通过设置循环的上限,然后使用循环展开的方式来进行分析,但是对于循环次数相对较多的情形,由于没有中断处理过程,可能出现路径编码的值溢出的情况,同时一条执行路径可能包含大量重复的路径片段,故在实验过程中我们引入断点机制,用以检测路径片段与执行路径编码的对应关系.当编码值出现溢出的情况,则使用新设置的一组相互正交的标量变量继续进行路径编码,同时在路径的回溯过程中,应从当前出现溢出的节点开始回溯分析,一直到上一个出现溢出的节点;若不存在这样的上一个溢出节点,则将入口基本块作为回溯的终点.在实验过程中,我们通过距离因子 k 设定循环执行次数,同时与之对应的执行路径为迭代路径.距离因子 k 和断点数量之间的对应关系如图 3 所示,它表示在某一 k 值情况下具有最多迭代路径的断点个数.实验过程中,k 的取值一般不超过 20,同时由于迭代路径的数量和 k 的取值呈指数关系,若 k 值继续增加可能导致分析的开销过大而出现资源不足的情况.另外需要注意的是,最弱前置条件在分析过程中路径谓词的数量会随着 k 值的增大而增多,从而使分析的效果趋于仅使用符号执行工具的效果.

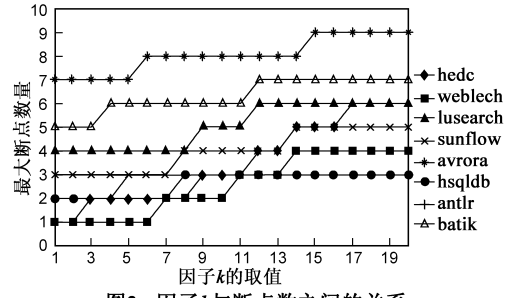


图3 因子k与断点数之间的关系

图 4 表示生成可行的相似路径数量的对比实验.我们采用符号执行工具 Java PathFinder 生成执行路径,实验对象为 Java PathFinder 加入本文方法后生成可行路径与仅使用 Java PathFinder 所生成的路径数量.对于每一个测试基准程序,仅使用 Java PathFinder 所生成的可行路径占所生成的所有路径的百分比用黑色的柱状图表示;加入 Java PathFinder 后所对应的百分比使用灰色的柱状图表示;不可行的路径百分比用带斜线的柱状图.采用本文的方法后,由于 Java PathFinder 在生成可行路径的时候,由于在分支结构和循环结构中有距离因子以及最弱前置条件的引导,可以生成更有针对性的相似路径集合,提高分析的精度与效率.

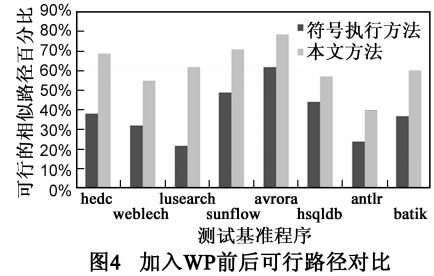


图4 加入WP前后可行路径对比

6 结论及将来工作

本文针对符号执行在分析过程中由于缺乏路径引导信息而生成大量无效路径的问题提出一种相似路径

生成方法,首先对程序控制流图的表示形式进行改进,通过对 CFG 中循环体中节点设置相互正交的标签编码,同时使用循环展开的方法消除 CFG 中的循环结构,从而不同的执行路径具有不同的路径编码,这样最弱前置条件在分析过程中可以依据路径编码来还原程序的执行路径.符号执行工具依据最弱前置条件所生成的引导信息可以生成相似的执行路径.实验分析表明本文的方法可以高效地针对指定的执行路径生成与之近邻的可行路径集合,在程序调试、程序分析过程中有重要的应用价值.

将来的工作可从以下两个方面展开:(1)由于本文的方法还不能够处理递归程序,故对于因函数调用而在调用图中出现的环,需分析针对环结构所生成的路径的可行性,从而可以分析函数递归调用而导致的死循环;(2)对于并行程序中的循环结构,由于需要对并行执行线程之间所有的循环条件组合进行分析,如何对本文的方法进行扩展从而支持具有此类结构的程序是下一步需要研究的内容.

参考文献

[1] 张健.精确的程序静态分析[J].计算机学报,2008,31(9):1549 – 1553.
Zhang Jian. Sharp static analysis of programs[J]. Chinese Journal of Computers, 2008, 31(9): 1549 – 1553. (in Chinese)

[2] King J. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385 – 394.

[3] Dijstra E. A discipline of programming[R]. Englewood Cliffs: Prentice Hall, 1976.

[4] 单锦辉,王戟,齐治昌.面向路径的测试数据自动生成方法述评[J].电子学报,2004,32(1):109 – 113.
J HShan, J Wang, Z C Qi. Survey on path-wise automatic generation of test data[J], Acta Electronica Sinica, 2004, 32(1): 109 – 113. (in Chinese)

[5] Miller W, Spooner D L. Automatic generation of floating point test data[J]. IEEE Trans on Software Engineering, 1976, 2(3): 223 – 226.

[6] Korel B. Automated software test data generation[J]. IEEE Trans on Software Engineering, 1990, 16(8): 870 – 879.

[7] Gupta N, et al. Automated test data generation using an iterative relaxation method[A]. Proceedings of the ACM SIGSOFT Sixth Int. Symposium on the Foundations of Software Engineering[C]. Orlando, Florida, USA, 1998. 231 – 244.

[8] 张岩,巩敦卫.基于搜索空间自动缩减的路径覆盖测试数据进化生成[J].电子学报,2012,40(5):1011 – 1016.
Y Zhang, D W Gong. Evolutionary generation of test data for path coverage based on automatic reduction of search space [J]. Acta Electronica Sinica, 2012, 40(5): 1011 – 1016. (in Chinese)

[9] Zhongxian G, Earl T, David J. Has the bug really been fixed [A]. Proceedings of the 32nd Conference on International Conference on Software Engineering(ICSE 2010)[C]. Cape Town, South Africa, 2010: 55 – 64

[10] He H, Gupta N. Automated debugging using path-based weakest preconditions[A]. Proceedings of the 7th Fundamental Approaches to Software Engineering(FASE 2004)[C]. Barcelona, Spain, 2004: 267 – 280

[11] Bradley A, Manna Z, Sipma H. Linear ranking with reachability[A]. Proceedings of the 17th International Conference on Computer Aided Verification(CAV 2005)[C]. Edinburgh, Scotland, U K, 2005: 491 – 504.

[12] Bradley A, Manna Z, Sipma H. The polyranking principle [A]. Proceedings of the 32nd International Colloquium on Automata, Language and Programming(ICALP 2005)[C]. Lisbon, Portugal, 2005: 1349 – 1361.

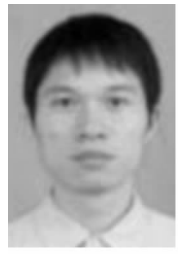
[13] Cook B, et al. Abstraction refinement for termination[A]. Proceedings of the 12th International Symposium on Static Analysis(SAS 2005)[C]. London, UK, 2005: 87 – 101.

[14] Velroyen H. Automatic nonermination analysis of imperative programs[D]. Chalmers University of Technology, Goteborg, 2007.

[15] Gupta A, et al. Proving non-termination[A]. Proceedings of the 35th Annual ACM SIGPLANSIGACT Symposium on Principles of Programming Languages(POPL 2008)[C]. San Francisco, California, USA, 2008: 147 – 158.

[16] Blackburn M, Garner R, Hoffman C, Khan M. The DaCapo benchmarks: Java benchmarking development and analysis[A]. Proceedings of the 21st International Conferences on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2006)[C]. Portland, USA, 2006: 169 – 190.

作者简介



郭　曦(通信作者)　男,1983 年出生于湖北,博士,华中农业大学讲师.主要研究方向:信息安全、软件分析、软件测试等.
E-mail: seyeyesx@163.com

王　盼　女,1987 年出生于河南,硕士,武汉电力职业技术学院讲师,武汉大学电气工程学院博士生.主要研究方向:电力电子功率变换、新能源等.
E-mail: wangpan6712063@163.com