# 基于程序行为特征比较的谓词切换排序①

苏 欣, 缪 力

(湖南大学 信息科学与工程学院, 长沙 410082)

**摘 要**: 谓词切换(Predicate Switching)通过动态改变程序中的谓词判断语句状态观察程序运行结果的变化, 分析可能与错误相关的关键谓词判断语句, 从而实现辅助错误定位. 谓词判断语句排序算法决定了谓词切换定位关键谓词判断语句的效率. 已有的排序算法如 LEFS 算法定位效率较低; PRIOR 算法虽然提高了定位效率, 但必须首先做程序动态切片找寻与错误相关的谓词判断语句集合, 而后建立程序依赖图以定义谓词判断语句的优先级, 这个过程需要花费大量的时间, 且算法复杂度较高. 在这两种算法基础上提出一种新的改进排序算法, 首先通过对比成功和失败的测试用例在运行中所展现出来不同程序行为特征, 以此定义谓词判断语句的优先级, 然后对不同优先级别的谓词根据执行先后顺序进行反向排序. 基于基准测试集 Siemens Suite 的程序进行了实验, 结果表明本文的排序算法与 LEFS 算法相比定位效率更高, 与 PRIOR 算法相比减少定义谓词优先级的耗费, 且算法更易于实现.

**关键词**: 谓词切换; 动态切片; 程序特征

## Predicate Switching Ordering Based on Branch Spectra Comparison

SU Xin, MIAO Li

(College of Information Science and Engineering, Hunan University, Changsha 410082, China)

**Abstract**: Predicate Switching locates and analyses the critical predicate to locate fault through dynamically changing the branch predicate outcomes at runtime and observes the changes in the program results. Predicate Switching Ordering algorithm determines the efficiency of locating the critical predicate. Existing ordering algorithms, such as the LEFS (Last Executed First Switched Ordering, LEFS) algorithm with lower efficiency and the PRIOR (Prioritization-based Ordering, PRIOR) algorithm with higher efficiency, the latter time consuming and complexity of algorithm, are defective. In this paper, we propose a novel and improved algorithm, which first determines the priority of the predicate by comparing different program spectra between the success and failure of test cases, then orders priorities of predicate, according to the reverse order of the predicate executions. To evaluate it, we tested our technique on Siemens Suite. The results have shown that the ordering algorithm is more efficient in locating the critical predicate than the LEFS, and less consuming but more implemented than the PRIOR.

**Key words**: predicate switching; dynamic slicing; program spectra

在程序开发与维护阶段, 有 50%-80%左右的时间和工作量花费在减少程序错误的调试工作中[1], 定位程序中的错误是调试过程中最困难也是最关键的一部分. 为了尽可能提升错误定位的效率和准确性, 采用自动化的方法进行错误定位是当今研究的热点. 目前关于错误定位的研究主要通过观察预先设计好的测试用例在运行过程中所反应出来的程序行为, 采用自动化方法进行分析和推测, 减少程序员搜索错误的语句范围. 这方面的技术主要包括 Dynamic Program Slicing[2], Delta Debugging[3]和谓词切换[4]等, 其中的

① 基金项目:湖南大学“青年教师成长计划”

收稿时间:2012-04-11;收到修改稿时间:2012-05-22

Delta Debugging 和谓词切换两者同属于对比定位法, 该方法特点是对比成功与失败测试用例在运行中所展现出来不同的程序行为, 动态的改变用例相应的状态, 进而观察最终输出结果的变化, 以此辅助定位程序错误位置.

随着程序规模的增大以及程序结构的复杂, 程序运行过程中需要考虑的状态数量变得非常巨大, 搜索程序所有状态需要花费大量的时间. 为减少进行搜索所需要的时间, Xiangyu 在 06 年提出谓词切换[5], 该方法将状态的改变模拟成谓词判断分支的改变, 提高错误定位的效率. 谓词判断语句排序算法决定了谓词切换定位关键谓词判断语句的效率, Xiangyu 提出了两种排序算法 LEFS(Last Executed First Switched Ordering) 和 PRIOR(Prioritization-based Ordering). LEFS 基于一个假设: 错误的语句往往与程序的最终输出较为接近, 所以最后执行的最先修改; PRIOR 则通过程序动态控制依赖图, 根据每个谓词判断语句与最终输出结果的依赖关系紧密程度定义其优先级, 与输出结果依赖关系最密切的条件判断语句优先级别最高. 实验证明 PRIOR 要比 LEFS 定位效率更高, 但 PRIOR 排序需要耗费大量的系统资源及时间进行预处理工作, 依据程序动态切片找寻与错误相关的谓词判断语句集合以及建立程序依赖图定义谓词判断语句的优先级等. 该排序算法的算法复杂度较高, 消耗时间较多, 难以适应实际工作应用.

本文提出了一种新的排序算法 BSC(Branch Spectra Comparison Ordering),首先通过对比成功和失败的测试用例在运行中所展现出来不同程序行为特征, 以此定义谓词判断语句的优先级, 然后对不同优先级别的谓词根据执行先后顺序进行反向排序. 通过实验证明, BSC 与 LEFS 算法相比定位效率更高, 与 PRIOR 算法相比减少了定义谓词优先级的耗费, 且算法更易于实现.

## 1 谓词切换

谓词切换的思想是当失败的测试用例运行至某一个谓词判断语句时, 如果原来该谓词判断语句的状态为 true, 则动态地将运行路径改为 false 分支, 如果状态为 false, 则动态改为运行 true 分支. 通过动态路径的改变使得原来失败的测试用例最终得到与预期相同的结果, 相应的谓词判断语句被称为关键谓词判断语句. 通过分析关键谓词判断语句, 实现辅助定位程序中的错误.

谓词切换定义了一套选择规则, 选择程序中的谓词判断语句进行动态修改, 再根据最终结果判断所选择的谓词判断语句是否为关键谓词判断语句. 该选择规则如下:

每次只挑选一个谓词判断语句进行动态修改. 为了降低由于修改程序中多个谓词判断语句的状态组合导致的极高复杂度, 避免谓词判断语句组合所带来的巨大计算量, 仅仅考虑单个谓词判断语句改变所带来的影响.

谓词切换的谓词判断语句选择排序的规则, 即排序算法有两种: LEFS 和 PRIOR.

谓词切换的算法, 描述如图 1 所示:

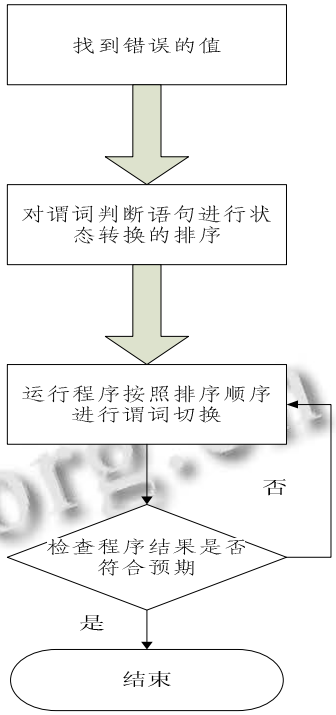


图 1 谓词切换算法

第一步: 找到错误的值

检查程序执行找到错误的值

第二步: 对谓词判断语句进行状态转换的排序

按照下面的方法运行程序 a.产生谓词判断语句的轨迹, 识别所有被执行的谓词判断语句运行轨迹 b.按照 LEFS 和 PRIOR 算法对谓词判断语句进行排序.

第三步: 反复运行程序, 检查运行结果

每一次运行都按照第二步的排序顺序改变相应的

谓词判断语句的状态，观察程序运行结果，如果程序运行结果符合预期则表明找到关键谓词判断语句并且终止查找.

## 2 程序行为特征

程序在每一次运行过程中都会表现出来的程序执行轨迹以及语句频率被统称为程序行为特征. 为了揭示程序行为特征与程序错误之间的关系，M.J.Harrold 做了大量的实验. 实验结果表明，程序出现异常的 Program Spectra[6]不一定表明该程序发生了错误，但是错误的程序运行一般都会显示出异常的 Program Spectra. 通过程序行为特征的分析比较获得与错误相关的程序信息被越来越多的用于辅助程序错误定位技术中. M. J. Harrold 对 Program Spectra 进行了相关的归纳总结[7]:

BS(Branch Spectra)记录程序中谓词判断分支的集合.

CPS(Complete-path spectrum) 记录程序本次运行中的完整路径包括程序中的所有循环.

PS(Path spectra) Path spectra 记录了程序中所经过无循环路径的集合.

DDS(Data-dependence spectra) 记录程序运行过程中变量的定义-使用对的集合.

Output 记录本次程序运行产生的输出结果.

ET(Execution trace)记录程序运行的轨迹，即记录本次程序运行过程中经过的所有语句.

因为本文的排序算法是对谓词判断语句进行排序，为需要分析的对象为谓词判断语句状态和执行路径，因此本文选取程序行为特征中的 Branch Spectra 进行分析比较.

## 3 BSC排序算法

本文的排序算法将程序行为特征分析运用到谓词切换排序中，M.J.Harrold 通过实验证明了异常的 Program Spectra 往往与程序的错误有关联，也就是说我们在比较成功测试用例和失败测试用例的 Program Spectra 中，两者有区别的程序特征与程序的错误关联程度较大，本文通过 Program Spectra 中的 BS 获得程序中的谓词判断语句执行情况和相关运行轨迹，对比成功和失败的测试用例，分析两者运行过程中的谓词判断语句状态，两者之间状态不同的谓词判断语句与错误的关联程度一般来说要高于状态一致的谓词判断语句. 基于此，本文提出一种新的谓词切换排序，该排序算法如下，分为三个步骤:

步骤一: 比较运行一个失败测试用例和一个成功测试用例的 BS，被循环多次执行的谓词根据执行先后次数进行标示区分，在执行次数相同的前提下将失败测试用例执行的与成功测试用例中的谓词判断语句进行对比后并分为三个集合，在成功测试用例中状态不同的谓词判断语句放入 A 集合; 在成功测试用例中没有执行的谓词判断语句放入 B 集合; 在成功测试用例中执行了但状态相同的放入 C 集合.

步骤二: 设置优先级为 A>B>C.

步骤三: 按照谓词判断语句的执行先后顺序分别对 A, B, C 三个集合中的条件判断语句进行反向排序，排序后根据优先级生成包含三者的集合.

以一个小程序举例说明如图 2 所示:

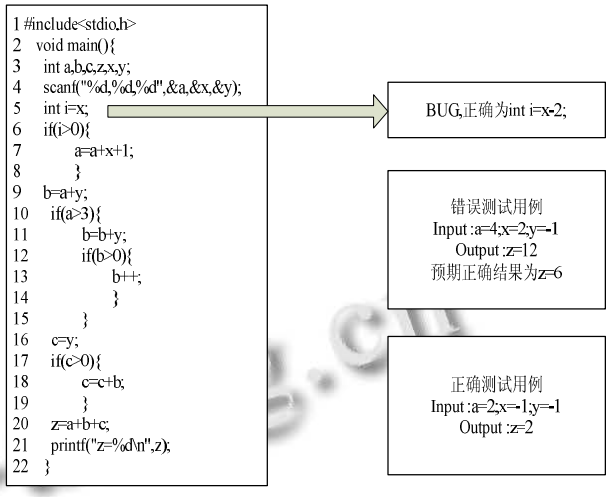


图 2 程序示意图

步骤一: 根据 BS，失败测试用例中运行谓词判断语句的集合为[6,10,12,17]，分别状态为[true,true,true,false]; 成功测试用例中运行的谓词判断语句为[6,10,17]状态为[false,false,false]观察比较两者之间的状态，将两者条件状态发生改变的谓词判断语句放在 A 集合为[6,10];成功用例中没有执行的谓词判断语句放在 B 集合为[12];两者条件状态相同判断语句放在 C 集合为[17].

步骤二: 设置优先集为 A>B>C.

步骤三: 将三个集合分别按照执行先后顺序反向排序，排序后按照优先顺序得到结果为(10,6,12,17).

根据 LEFS 得出来的排序集为(17,12,10,6)对比本算法得出来的(10,6,12,17). 本算法花费了 2 步就找到了关键谓词判断语句 5, 而 LEFS 则花了 4 步, 本算法的找寻效率明显要优于之前的 LEFS.

## 4 实验及结果

为了验证本文算法的定位效率, 本文用脚本文件完成了目录的创建, 对比, 用 JAVA 文件完成按照指定排序算法对谓词判断语句的排序以及排序文件的生成. 谓词判断语句状态的动态改变本文使用了工具 Nionka, Nionka[8]是 Petar Tsankov[9]在设计完成的,通过改变程序的谓词判断语句的状态, 使程序强制按某条路径执行的工具. 本文采用错误定位技术研究领域得到广泛使用的基准程序集 Siemens Suite, 分别用 LEFS 排序算法和 BSC 排序算法对 Siemens Suite[11]预先植入错误的程序进行寻找关键谓词判断语句的实验, 并根据统一的评价标准, 对比两者找寻关键条件判断语句的效率. 实验结果如下表 1:

表 1 Siemens Suite 实验结果

| Program | LEFS | BSC |
|---|---|---|
| print_tokens2[v2] | 44 | 36 |
| print_tokens2[v6] | 34 | 32 |
| print_tokens2[v7] | 47 | 46 |
| replace[v2] | 18 | 2 |
| schedule[v4] | 10 | 5 |
| totinfo[v3] | 15 | 13 |
| totinfo[v4] | 6 | 7 |

在 7 个实验程序中, 对 6 个实验程序本文算法都能比 LEFS 提高搜索效率, 其中 replace[v2]用 LEFS 需要 18 步, 而本文算法仅仅需要 2 步. 根据表中的数据, LEFS 平均通过 24.8 步找到关键选择判断语句, 本算法平均仅通过 20.1 步找到关键条件判断语句. 实验结果证明, 在大多数测试集的测试中本文的排序方法的定位效率要明显优于 LEFS. 唯一在效率低于 LEFS 算法的是 totinfo[v4], 这是因为该程序数据集选取的正确和错误测试用例中不存在状态不同的条件判断语句, 因此导致本算法在找寻效率上失去相对 LEFS 优势. 本文用 BSC 排序算法对 Xiangyu 实验中采用的 grep 2.5 和 tidy 等测试集进行了相关实验并与 Xiangyu 的实验数据进行对比, 比较结果表 2:

表 2 grep2.5 等测试集实验结果

| Program | PRIOR | BSC |
|---|---|---|
| grep 2.5 | 56 | 60 |
| tidy | 1 | 4 |
| bc-1.06 | 2 | 8 |
| tar-1.13.25 | 3 | 5 |

实验数据表明 BSC 定位效率十分接近于 PRIOR 的定位效率. 由于 PRIOR 算法并未给出排序的时间, 且本文实验的工具环境不同于 PRIOR, 因此难以直接通过实验与 PRIOR 算法对比. 一个算法的评价主要从算法复杂度来考虑. 在时间复杂度方面, LEFS 和 BSC 均只进行一次谓词排序循环, 排序循环算法的时间复杂度为 O(n), 因此两种排序算法的时间复杂度为 O(n). PRIOR 算法需要进行两次基于动态依赖图的动态切片操作, 构造动态程序依赖图的算法的时间复杂度为 O(n3), 因此 PRIOR 算法的时间复杂度为 O(n3). 由于三个算法步骤中所需要的额外空间为固定值, 因此, 三个算法的空间复杂度等同于该算法的时间复杂度. BSC 在算法复杂度上等于 LEFS, 明显低于 PRIOR 算法.

## 5 结语

本文结合程序行为特征提出了一种新的谓词切换排序方法, 通过对成功测试用例和失败测试用例谓词判断语句状态的差别定义谓词判断语句的优先级, 根据优先级进行谓词切换排序. 通过实验验证, BSC 在定位关键谓词判断语句的效率上要优于 LEFS, 接近于 PRIOR, 且算法复杂度较 PRIOR 低, 更易于实现. 在测试中, 如果成功测试用例在运行过程中几乎不存在和失败测试用例状态不同的谓词判断语句, BSC 在找寻效率上和 LEFS 差不多甚至低于 LEFS, 所以如何选择合适的正确测试用例是未来要解决的一个问题. 本文下一步的工作重点是利用更大的实验程序来验证我们方法的有效性, 并且更进一步完善该排序算法, 并且解决选择合适的正确测试用例的问题.

## 参考文献

1 Collofello S, Woodfield SN. Evaluating the effectiveness of reliability-assurance techniques. Journal of Systems and Software, 1989,9(3):191−195.

2 Korel B, Laski J. Dynamic Program Slicing. Information

(下转第 207 页)

图像、10 幅图像、15 幅图像时平均查准率和查全率. 从表中数据可以看出, 在查全率约在 20%至 25%范围内, 则查准率约在 85%至 95%范围内, 在查全率约在 35%至 45%范围内, 则查准率约在 75%至 85%范围, 在查全率约在 55%至 65%范围, 则查准率约在 70%至 80%范围内.

表 1 不同类别图像查全率、查准率比较

| 参数 / 种类 | 5 | | 10 | | 15 | |
|---|---|---|---|---|---|---|
| | **查全率** | **查准率** | **查全率** | **查准率** | **查全率** | 查准率 |
| Apple | **0.23** | **0.93** | **0.43** | **0.86** | **0.60** | 0.80 |
| bone | **0.25** | **0.98** | **0.50** | **0.96** | **0.68** | 0.91 |
| bottle | **0.21** | **0.86** | **0.42** | **0.83** | **0.58** | 0.77 |
| carriage | **0.20** | **0.87** | **0.36** | **0.73** | **0.53** | 0.71 |
| glass | **0.23** | **0.93** | **0.43** | **0.87** | **0.61** | 0.82 |
| misk | 0.23 | 0.92 | 0.40 | 0.80 | 0.60 | 0.80 |

## 3 结论

角点是图像的重要特征, 根据角点在图像圆形分块中的不同位置, 及相邻角点之间的间隔度数, 来确定图像的特征描述符, 用这样的描述符可以完成基于相邻角点间隔度数的图像检索. 由于采用圆形分块, 及相邻角点的间隔度数建立直方图的描述符, 使得其具有目标形状的缩放、平移和旋转等形变具有很好的鲁棒性. 通过实验证明, 如果图像轮廓较稳定, 且图像内部没有纹理的, 则可以达到 70%以上的查准率.

## 参考文献


1 Zhang DS, Lu GJ, Review of shape representation and description techniques. Pattern Recognition, 2004,37,1–19.

2 Kim H, Kim J, Region-based shape descriptor invariant to rotation, scale and translation. Signal Process Image Communication, 2000,16,(5):87–93.

3 周明全,耿国华,韦娜.基于内容图像检索技术.北京:清华大学出版社,2007.

4 王涛,刘文印,孙家广.傅立叶描述子识别物体的形状.计算机研究与发展,2002,39(12):1 714–1719.

5 陈武凡.小波分析及其在图像处理中的应用.北京:科学出版社,2002.

6 Abbasi S, Mokhtarian F, Kittler J. Curvature scale space image in shape similarty retrieval, Multimedia System, 1999, 7:467–476.

7 王建琦,邓雁萍,李介谷.一种改进的角点检测算法.上海交通大学学报,2004,34(7):913–916.

8 Sajjanhar A. 2003 Spatial information in histograms for shape representation, Lecture Notes in Computer Science 2690, 855–859.


(上接第 201 页)


Processing Letters. 1988,3(29):155–163.

3 Zeller RH. Simplifying and Isolating Failure-inducing Input. IEEE Trans. on Software Engineering, 2002, 28(2): 183–200.

4 Zhang XY. Fault Location via Precise Dynamic Slicing[MS Thesis]. Department of Computer Science on University of Arizona, 2006: 71–80.

5 Zhang XY, Gupta N, Gupta R. Locating Faults through automated Predicate Switching. Proc. of the 28th int. conf. on Software Engineering. 2006: 272–281.

6 Reps T, Ball T, Das M. The use of program profiling for software maintenance with applications to the year 2000 problem. ACM Software Engineering Notes, 1997,22(6): 432–439.

7 Harrold MJ, Rothermel G, Wu R. An empirical investigation of program spectra. Proc of the 1998 ACM SIGPLAN- SIG SOFT Workshop on Program Analysis for Software Tools and Engineering. 1998: 109–119.

8 NIONKA[2011-05-10] http://www.cc.gatech.edu/~orso/nionka/nionka-x86.php.

9 Tsankov P, Jin W, Orso A. Execution Hijacking: Improving Dynamic Analysis by Flying off Course. Proc. of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. 2011:200–209.

10 Valgrind [2011-03-08].http://valgrind.org/.

11 Do H, Elbaum SG, Rothermel G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Software Engineering: An international Journal, 2005,10(4):405–435.