面向自动修复并融合失效场景的缺陷定位方法

李 昂 毛晓光 雷 晏
(国防科学技术大学计算机学院 长沙 410073)

摘 要 为了应对日益增长的软件修复开销,研究高效的软件自动修复技术成为学术界和工业界的共识。缺陷定位作为自动修复技术的前端,是实现快速准确自动修复的关键,其精度直接影响自动修复的性能。然而,初步研究表明,现有缺陷定位技术缺乏对自修复需求的考虑,对自修复算法支持有限。有必要研究面向自修复的高精度自动化缺陷定位技术,以提升自修复性能。因此,提出了失效场景的缺陷定位方法来应对该问题。提出的方法首先采用程序切片技术,构造出与失效相关的场景;然后对失效场景的各个元素实施可疑值度量;最后将可疑值度量化的场景交给自动修复技术实施修复。初步实验结果表明,本缺陷定位方法能有效提升自动修复性能。

关键词 自动修复,缺陷定位,失效场景,程序切片,可疑值度量
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.12.023

Fault Localization Using Failure-related Contexts for Automatic Program Repair

LI Ang MAO Xiao-guang LEI Yan
(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract In face of the high cost consumed by program repair in the software life-cycle, researchers from both academia and industry are trying to develop an effective automatic program repair technique to address this problem. Being an essential part of program repair, fault localization plays an important role in repair performance. However, preliminary studies have shown that existing fault localization approaches do not take into account the features of automatic repair, and therefore restrict repair performance. It is vital to design fault localization approaches for automatic repair to improve its performance. To address this issue, this paper proposed a fault localization approach using failure-related contexts for improving automatic program repair. The proposed approach first utilizes program slicing technique to construct a failure-related context, then evaluates the suspiciousness of each element in the context of being faulty, and finally presents this context and its elements with different suspiciousness to automatic program repair techniques for performing repair on faulty programs. The preliminary experimental results demonstrate that the proposed approach is effective to improve automatic repair performance.

Keywords Automatic repair, Fault localization, Failure-related context, Program slicing, Suspiciousness evaluation

1 引言

随着软件规模与复杂度不断增加,即使采用先进的开发方法和工具,软件缺陷也不可避免地存在。已有研究[5,10]表明,涉及缺陷修复的软件测试、调试和维护活动的成本巨大,大约占整个项目成本的50%~75%,甚至有时可以平均占据整个软件项目总成本的90%。为了改变上述现状来提升修复缺陷的效率,近年来研究人员致力于实现软件修复的自动化,并提出了很多软件自动修复技术,例如JAFF[11,12]、基于类契约的自修复技术[13,14]和Par[15]等。其中,最典型且影响最广泛的先进自动修复技术,是美国弗吉尼亚大学所提出的GenProg[2,3]。GenProg扩展了遗传算法的概念,将其应用到了缺陷自动修复上,开发出一个高效的自动修复C程序的开源工具GenProg。该小组基于大规模实验,证明GenProg可以成功自动修复libtiff、php、python和wireshark这些典型开源软件中105个软件缺陷中的55个[2]。这表明GenProg具有较好的修复能力和实际使用价值。尽管GenProg表现出不错的修复能力,但是GenProg在效率和效力两个方面仍不够理想,需要进一步解决和改进。例如GenProg在实验中使用具有强计算能力的Amazon的EC2云计算设施,整个实验仍然耗费了大量的资源和时间,一个成功修复平均要花费96分钟[2]。

自动修复活动一般由3个部分组成:缺陷定位、补丁生成和补丁有效性验证。作为前端,缺陷定位主要提供引起软件失效的缺陷位置的相关信息,其精度直接影响着补丁生成和补丁有效性验证的效率,从而最终影响自动修复性能。本小组之前的研究[16]发现,现有缺陷定位研究主要是从人工修复的角度来进行设计和评估,而缺乏对自修复需求的考虑,从而导致传统性能良好的缺陷定位技术,对自修复的支持反而不够。有必要研究面向自修复的缺陷定位技术,来提升自动修

到稿日期:2015-02-13 返修日期:2015-03-30 本文受国家自然科学基金(61379054)资助。
李 昂(1992-),男,硕士生,主要研究领域为软件缺陷定位与自动修复;毛晓光(1970-),男,博士,教授,博士生导师,主要研究领域为高可信软件、软件开发方法、软件维护等,E-mail:xgmao@nudt.edu.cn;雷 晏(1985-),男,博士生,主要研究领域为软件缺陷定位与自动修复。

复性能。由于现有缺陷定位技术从整个程序的所有语句出发，可疑语句集合中存在许多和缺陷无关的语句，这些无关语句会给有效补丁生成带来干扰，从而导致自修复过程中产生大量的无效补丁，消耗大部分计算资源。这极大地制约了补丁生成效率，影响着后续补丁有效性验证效率，降低了整个自修复过程的性能。

基于上述分析和发现，本文提出失效场景的缺陷定位方法，以构造失效场景，将补丁生成的语句变异缩小到失效场景内，给补丁生成提供更多指导性信息，以此总体提升自修复性能。本文方法首先借助程序切片技术[17-19]，从单个失败测试用例的错误输出出发，构造出与失效相关的场景；然后，基于频谱的缺陷定位 SFL[1]（Spectrum-based Fault Localization）对失效场景所包含的各个语句实施可疑值度量；最后，将可疑值度量化的场景交给自动修复技术实施修复。由于 GenProg 是自修复领域最先进和影响力最为广泛的技术，本文选择将缺陷定位方法应用在 GenProg 上，即替换原有 GenProg 的缺陷定位方法。初步实验结果表明，本缺陷定位方法能有效减少无效补丁数生成，并且大幅度降低时间开销，从而显著地提升自修复的性能。

2 融合失效场景的缺陷定位方法

2.1 失效场景的构建

程序切片技术最早由 Weiser M[17] 提出，主要用于辅助编程人员调试。程序切片技术通过使用数据或控制依赖关系，能找出那些能直接或者间接影响程序输出值的语句，这些语句组成一个切片。当一个程序失效时，它所产生的错误信息往往能够为修复程序提供有价值的线索。本小组通过查找与缺陷相关的关键变量，对其进行针对性的逆向跟踪，即通过对其进行切片处理，从而确定与其相关的操作以及包含的语句。

一个程序的失效场景是由与已知缺陷相关的所有语句构成的一个语句集合。它能够标识缺陷产生的位置范围，从而为缺陷定位模块缩小搜索区域。通过运用失效场景，可以得到更为精确的缺陷定位结果，同时花费更少的时间。本文通过静态程序切片技术来构建失效场景，首先假设有一个与失效相关的变量集合 $V=\{V_1,V_2,\cdots,V_n\}$，以及一个在程序中这些变量对应的输出位置的集合 $S=\{S_1,S_2,\cdots,S_n\}$，从而可以得到相应的静态切片的集合 $P=\{P_1,P_2,\cdots,P_n\}$，其中 $P_i=(S_i,V_i)$。将切片集合中所有的元素进行并集运算，便可得到相对应的失效场景：$F=P_1\cup P_2\cup\cdots\cup P_n$。

2.2 融合失效场景的缺陷定位方法

假设将错误输出作为切片的兴趣点，那么此时切片就能找出那些直接或者间接影响错误输出值的语句集合，即能找到与失效相关的语句集合；而且切片还能呈现出这些语句在数据和控制依赖关系上相互影响和作用的原理。基于这些特点，本文提出了融合失效场景的缺陷定位方法（Fault Localization Using Failure-related Contexts，FLFC），即以错误输出作为兴趣点，以程序切片技术作为手段，构建出失效场景，并进一步细分场景内各语句影响力，为后续自修复阶段提供有力支持。下面给出 FLFC 工作原理的具体步骤。

步骤 1　使用程序切片技术构造失效场景。这步从失败测试用例中随机选取一个失败测试用例，采用程序切片技术，对该失败测试用例的错误输出实施逆向切片。此切片能显示失败测试用例错误执行的数据和控制流动，即能展现各个语句如何作用而导致了该测试用例运行失败，发生了程序失效。因此，本文方法以此切片作为失效场景。

步骤 2　用可疑值来衡量出各个语句对错误输出的影响力。这步会采用 SFL[1] 技术来度量出失效场景内各个语句的可疑值，以可疑值为标准来细分出各个语句对错误输出的影响力，即语句可疑值越高，该语句对错误输出的影响力越强。首先，假设程序 P 的语句集合 $S=\{s_1,s_2,\cdots,s_M\}$，在测试用例集 $T=\{t_1,t_2,\cdots,t_N\}$ 中运行，T 至少包含一个失败测试用例集（见图 1）。因此，$M=|S|$，$N=|T|$。图 1 所示 $N\times(M+1)$ 的矩阵是 SFL 的输入，称为 SFL 输入矩阵。元素 x_{ij} 等于 1 时，表示语句 s_j 被测试用例 t_i 的运行所执行，否则 x_{ij} 等于 0。SFL 输入矩阵最右边的结果向量 r 表示测试用例的运行结果。元素 r_i 等于 1 时，表示测试用例 t_i 的运行结果失败，否则 t_i 等于 0。基于图 1 所定义的矩阵，SFL 给每个语句定义如下 4 个变量：$a_{00}(s_j)$ 表示未执行语句 s_j 的成功测试用例数；$a_{01}(s_j)$ 表示未执行语句 s_j 的失败测试用例数。$a_{10}(s_j)$ 表示执行语句 s_j 的成功测试用例数，$a_{11}(s_j)$ 表示执行语句 s_j 的失败测试用例数。基于这 4 个变量，SFL 提出很多可疑值度量公式，典型的有 Ample、Arithmetic_mean、Cohen、Jaccard、M2 等。

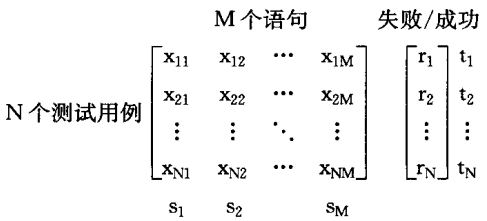

图 1　SFL 输入矩阵

步骤 3　将可疑值度量化的失效场景输入到自动修复后续补丁生成等活动中，实施自动修复。这步将失效场景以及包含的各个语句对应的可疑值输入到自动修复的补丁生成中，指导补丁生成。FLFC 可以直接替换 GenProg 原有的缺陷定位方法模块，然后运行采用本文缺陷定位方法的 GenProg 来评价本文缺陷定位方法的自动修复性能。

2.3 具体实现

FLFC 主要通过在缺陷定位模块前增加一个程序切片模块，对失败测试用例的错误输出实施逆向切片，从而构造失效场景。具体实现时，程序切片模块使用的是专门用于处理 C 程序的开源工具 Frama-C，SFL 模块选择的是 Jaccard 方法。

基于 FLFC 的 GenProg 的具体实现过程如图 2 所示。

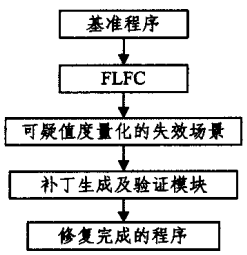

图 2　基于 FLFC 的 GenProg

3 实验

3.1 基准程序集和评价指标

目前已完成了 GenProg 基准程序集中 5 个基准程序的初步实验，相比于 SFL 技术，GenProg 的缺陷定位方法过于粗

粒度，而且之前的研究[16]已经表明 SFL 比 GenProg 的缺陷定位方法对修复性能的提升更好。其中，该研究还发现基于 SFL 的 Jaccard 缺陷定位方法对自修复的支持最好，而且 FLFC 需采用 SFL 的可疑值度量公式来度量失效场景内的语句可疑值。因此本文缺陷定位方法采用 Jaccard 可疑值度量公式来进行失效场景内的可疑值度量，并与基于 SFL 的 Jaccard 方法比较。具体来说，FLFC 与基于 SFL 的 Jaccard 方法分别应用到 GenProg 中，替换 GenProg 原有的缺陷定位方法，比较分别应用 FLFC 和 SFL 技术后 GenProg 的修复性能。

本实验从效力和效率两个方面来评价修复性能。在效力评价上，本实验采用无效补丁数 NCP[16]（Number of Candidate Patches），即在搜索到有效补丁之前已经生成的无效补丁数。显然，较少的 NCP 意味着更好的自动修复效力。在效率评价上，本实验采用修复时间，即从开始到找到有效补丁的整个修复过程所花费的时间。较少的修复时间意味着时间花销更低，具备更好的效率。

3.2 结果与分析

针对选取的 5 个基准程序进行了实验分析，从基于箱形图（Boxplot）的修复时间和 NCP 分析及基于 Wilcoxon 符号秩检验（Wilcoxon-Signed-Rank Test）的统计比较两个方面来进行实验结果分析。

图 3 和图 4 利用箱线图分别展示了本次实验在修复时间和 NCP 两个评价指标下，FLFC 和 Jaccard 对自修复的效果。从这两个图可以看出，FLFC 应用到自修复技术后其修复时间和 NCP 相比于原有最优方法 Jaccard，在各个基准程序上大幅度地降低。具体来说，应用 FLFC 后，Genprog 平均修复时间大幅减少，平均降幅为39.74%，具备了更优的效率；同时，NCP 值相比原来也相应减少，平均降幅为 38.37%，避免产生过多冗余补丁，获得了更好的自动修复效力。

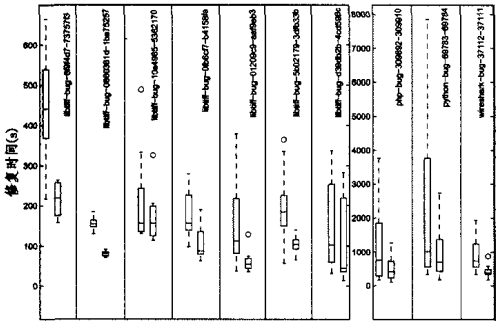

图 3 基于修复时间的箱线图

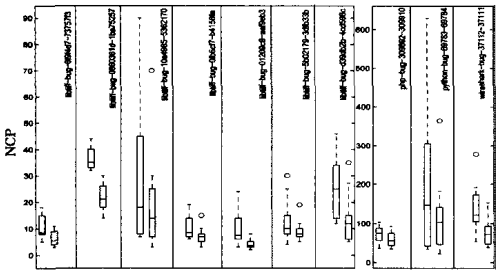

图 4 基于 NCP 的箱线图

实验同时采用了更为严谨科学的对比方法，即 Wilcoxon 符号秩检验方法，来进行具有统计意义的分析比较。在给定的显著性水平 σ 下，可以使用双尾（2-tailed）的 p 值和单尾（1-tailed）的 p 值来得出结论。每个检验会分别使用双尾和单尾，显著性水平 σ 为 0.05。给定一个基准程序，基于 FLFC 的 GenProg 在该程序上运行 20 次实验的修复时间或者 NCP 值作为 $F(x)$，而 $G(y)$ 为基于 Jaccard 的 GenProg 在该程序上运行 20 次实验的修复时间或者 NCP 值。

表 1 显示了采用 Wilcoxon 符号秩检验方法，FLFC 与 Jaccard 在修复时间和 NCP 上的对比。可以看出，在统计学意义上，FLFC 在修复时间和 NCP 上都显著小于 Jaccard，在所有基准程序上取得了更好（BETTER）的结果。因此，基于箱线图和 Wilcoxon 符号秩检验方法两个方面的结果，本文提出融合语义场景的缺陷定位方法能显著提升自修复性能。

表 1 基于修复时间和 NCP 的 Wilcoxon 符号秩检验

基准程序		双尾检验	右单尾检验	左单尾检验	结论
libtiff-bug-6f9f4d7-73757f3	修复时间	5.06E-03	9.98E-01	2.96E-03	BETTER
	NCP	7.80E-03	9.97E-01	4.54E-03	BETTER
libtiff-bug-0860361d-1ba75257	修复时间	5.06E-03	9.98E-01	2.96E-03	BETTER
	NCP	4.67E-03	9.98E-01	2.74E-03	BETTER
php-bug-309892-309910	修复时间	5.06E-03	9.98E-01	2.96E-03	BETTER
	NCP	4.92E-03	9.98E-01	2.88E-03	BETTER
wireshark-bug-37112-37111	修复时间	5.06E-03	9.98E-01	2.96E-03	BETTER
	NCP	4.84E-03	9.98E-01	2.83E-03	BETTER
python-bug-69783-69784	修复时间	5.06E-03	9.98E-01	2.96E-03	BETTER
	NCP	4.86E-03	9.98E-01	2.85E-03	BETTER

结束语 本文从自动修复的需求出发，提出面向自动修复并融合失效场景的缺陷定位方法，并将该方法应用到当前最具代表性的自动修复技术 GenProg 上。实验将本文方法与当前先进的缺陷定位技术 SFL 在 GenProg 中的性能进行了对比。初步实验结果表明，以构建失效场景的方式来指导自动修复后续补丁生成等活动，能更高效地提升自修复性能。未来将继续改善 FLFC 的精确度和效率，使用更为准确的动态切片模块，同时寻找效率更高的 SFL 技术以获得更好的效果。

参 考 文 献

[1] Naish L, Lee H J, Ramamohanarao K. A model for spectra-based software diagnosis [J]. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): 1-32

[2] Goues L C, Dewey-Vogt M, Forrest S, et al. A systematic study of automated program repair: fixing 55 out of 105 bugs for $8 each[C]// Proc. of the 34th International Conference on Software Engineering (ICSE). Zurich, Switzerland, 2012: 3-13

[3] Goues L C, Nguyen T, Forrest S, et al. GenProg: a generic method for automatic software repair[J]. IEEE Transactions on Software Engineering (TSE), 2012, 38(1): 54 -72

[4] Weimer W, Nguyen T, Goues L C, et al. Automatically Finding Patches Using Genetic Programming[C]// Proc. of the 31st International Conference on Software Engineering (ICSE). Vancouver, Canada, 2009: 16-24

[5] Seacord R C, Plakosh D, Lewis G A. Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices[M]. Addison-Wesley, 2003

[6] Anvik J, Hiew L, Murphy G C. Who should fix this bug? [C]// Proc. of the 28th International Conference on Software Engineering (ICSE). Shanghai, China, 2006: 361-370

[7] http://www.mozilla.org/security/bug-bounty.html MYM3,000/bug

[8] http://blog.chromium.org/2010/01/encouraging-more-chromium-security.html $500/bug

[9] http://www.tarsnap.com/bugbounty.html

（下转第 107 页）

这个测试项目是实验室一个科研人员曾经做过的项目，他重新查看这个项目的代码来考虑哪些地方应该进行函数抽取重构并进行记录。共进行了 20 个函数抽取重构。取相似度阈值从 0.1 到 0.9 进行了实验，得到的查准率、查全率如表 2 所列(与程序员操作记录比较)。

表 2　相似度阈值与查准率、查全率的关系

相似度阈值	检测结果总个数	为真个数	查准率	查全率
0.1	24	19	79%	95%
0.2	23	19	82%	95%
0.3	23	19	82%	95%
0.4	21	17	80%	85%
0.5	20	17	85%	85%
0.6	15	13	86%	65%
0.7	13	11	84%	55%
0.8	11	10	90%	50%
0.9	11	10	90%	50%

图 1 更直观地展现了相似度阈值的取值对实验结果的影响，其中横轴表示相似度阈值的变化，纵轴为查准率和查全率随着相似度的变化。从图中可以看出，当相似度阈值太大时，可以提高查准率但是会降低查全率，而相似度阈值太小时，查全率会增大但是查准率会下降。在相似度阈值为 0.5 时，查准率与查全率基本平衡。因此，后续实验中采用 0.5 作为相似度阈值。

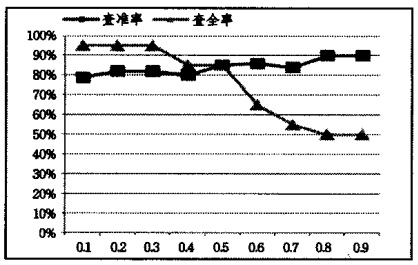

图 1　相似度阈值对查准率与查全率的影响

3.3　实验结果及分析

对以上 8 个开源项目进行了实验(相似度阈值取值为 0.5)，总共检测到 1393 个函数抽取重构。对检测的结果，由独立程序员进行核对确定检测结构是否正确。表 3 列出了每一个开源项目检测到的函数抽取重构的个数以及查准率。实验结果表明，本文提出的函数抽取重构检测方法的平均查准率约为 80%，在不同的项目上，其查准率在 65%到 90%之间略有波动。

表 3　检测结果 3

项目名称	检测到的函数抽取重构数量	查准率
Ant	229	80%
DoubleType	30	70%
Findbugs	117	85%
Hibernate	160	90%
JEdit	271	75%
Jfreechart	303	65%
PMD	45	90%
Weka	238	85%

结束语　软件重构及重构检测是软件工程领域的研究热点之一。本文提出了一种函数抽取重构的自动化检测方法，实现了相关算法并进行验证。在 8 个大型开源软件系统上的实验结果表明，该方法具有较高的准确率。

后续工作主要包括以下几个方面：首先，将该方法扩展至其他编程语言，比如 C、C++等；其次，计划在更多开源系统上进行进一步的实验验证。目前的实验采用了 8 个大型开源系统的 275 个版本，涉及的代码行数超过一千万。

参 考 文 献

[1] Mens T, Tourwe T. A survey of software refactoring [J]. IEEE Transactions on Software Engineering, 2004, 30(2): 126-139

[2] Opdyke W F. Refactoring object-oriented frameworks[D]. Llinois: University of Illinois at Urbana-Champaign, 1992

[3] Dig D, Comertoglu C, Marinov D, et al. Automatic detection of refactorings in evolving components[C]// European Conference on Object Oriented Programming. 2006: 404-428

[4] Xing Z, Stroulia E. UMLDiff: an algorithm for object-oriented design differencing[C]// Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. 2005: 54-65

[5] Weissgerber P, Diehl S. Identifying refactorings from source-code changes[C]//21st IEEE/ACM International Conference on Automated Software Engineering. 2006: 231-240

[6] Murphy G, Kersten M, Findlater L. How are Java software developers using the eclipse IDE[J]. Software IEEE, 2006, 23(4): 76-83

[7] http://code.google.com/p/java-diff-utils/

(上接第 104 页)

[10] Hailpern B, Santhanam P. Software debugging, testing, and verification[J]. IBM Systems Journal, 2002, 41(1): 4-12

[11] Arcuri A. On the automation of fixing software bugs[C]//Proc. of the 30th International Conference on Software Engineering (ICSE). Leipzig, Germany, 2008: 1003-1006

[12] Arcuri A. Evolutionary repair of faulty software[J]. Applied Soft Computing, 2011, 11(4): 3494-3514

[13] Wei Y, Pei Y, Furia C A, et al. Automated fixing of programs with contracts[C]// Proc. of the International Symposium on Software Testing and Analysis (ISSTA). Trento, Italy, 2010: 61-72

[14] Wei Y, Pei Y, Furia C A, et al. Inferring better contracts[C]// Proc. of the 33rd International Conference on Software Engineering (ICSE). Honolulu, Hawaii, USA, 2011: 191-200

[15] Kim D, Nam J, Song J, et al. Automatic patch generation learned from human written patches[C]// Proc. of 35th International Conference on Software Engineering (ICSE). San Francisco, California, 2013: 802-811

[16] Qi Yu-hua, Mao Xiao-guang, Lei Yan, et al. Using automated program repair for evaluating the effectiveness of fault localization techniques[C]// The 22th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). Lugano, Switzerland, 2013: 191-201

[17] Weiser M. Program slicing[J]. IEEE Transactions on Software Engineering, 1984, 10(4): 352-357

[18] Korel B, Laski J. Dynamic Program Slicing[J]. Information Processing Letters, 1988, 29(3): 155-163

[19] Agrawal H, Horgan J R. Dynamic program slicing[C]// Proceedings of the Conference on Programming Language Design and Implementation (PLDI). 1990: 246-256