类不变式,函数执行后的状态满足后置条件和类不变式,且执行过程中还需要满足过程内部断言(intermediate assertion).这些断言和不变式规范了程序的正确行为,一定程度上可以理解为基于模型的程序自动修复方法.AutoFix-E 在给定的实验数据集上进行实验,可以成功地修复 42 个错误中的 16 个.

2011 年,裴玉等人[74]提出了 AutoFix-E2,在 AutoFix-E 基于模型的修复方法基础上,进一步挖掘程序本身包含的信息来指导修复.他们将其称为基于证据的修复方法,将随机测试、错误定位、动静态分析收集的结果作为证据.具体通过静态分析(表达式依赖和控制依赖)和运行通过/未通过两种测试用例的动态分析,当检测到待修复表达式接受一个可疑值时将其转换为合法的值,从而保持契约的成立.

2014 年,裴玉等人[75]提出了 AutoFix 的最新实现和实验,其部分思想在文献[27,74]中已经介绍.AutoFix 使用 AutoTest 基于契约和随机测试框架自动产生测试用例,由失败的测试用例驱动动态分析进行错误定位,基于契约指导生成补丁并进行判定,最终利用相关性对判定通过的补丁进行排序.在给定的 204 个错误修复实验中,AutoFix 的正确率为 42%.由于 AutoFix 生成补丁的正确性判断依据是给定的契约(如果给定的契约表示了不完整的程序规约,则可能引入误报),在 AutoFix 给出的正确补丁中进行人工检查,其完全正确的补丁占有率为 59%.

2015 年,裴玉等人[76]提出了在 IDE 中进行程序自动修复的思想,将 AutoFix 集成到 EiffelStudio 集成开发环境中.AutoFix 作为一个推荐系统在 IDE 后台自动地检测源码错误并给出建议的修复补丁,这将为开发人员提供强大的自动化功能.

4.2 手工编写程序规约

如下修复问题中刻画的程序规约 S 使用线性时序逻辑(linear-temporal logic)、分离逻辑(separation logic)等手工编写,其针对不同程序和修复问题手工编写,该程序规约 S 是否完整不确定.

2005 年,Jobstmann 等人[77]将程序修复问题刻画为一种游戏,获得一组对程序修改的策略,使得修改后的程序符合程序规约,则认为获得一次胜利.程序规约 S 由线性时序逻辑(linear-temporal logic)给出.该方法假设程序出错范围仅在程序表达式语句或赋值语句的左值,也就是说,其修复能力仅包含程序表达式错误或赋值语句错误的修复.

2013 年,Essen 等人[78]使用线性时序逻辑(linear-temporal logic)给出形式化程序规约 S,提出一种新的修复方法,要求修复后的程序符合该给定的程序规约,同时和原 BUG 程序语义保持相似.该方法最大的特点是在修复过程中强制保持一个原 BUG 程序执行轨迹的子集,从而自动将程序正确部分本身的语义保持下来而不被修改.也就是说,原 BUG 程序符合程序规约的正确部分需要持续保持,同时只需要小范围地修改出错部分的语义,使得整个修复后的程序在保持原正确部分的同时,满足线性时序逻辑刻画的规约 S.

2015 年,Kneuss 等人[79]提出一种修复递归函数数据类型错误(树、链表、整形)的方法.该方法需要开发人员使用特定的语言编写程序和对应的程序规约 S,其假设待修复的错误程序是有限状态的(infinite-state)并且程序规约是完整正确的,最终的修复程序经过 Leon 系统进行形式化验证.

2018 年,Rijnard 等人[70]提出用静态方法检测和修复指针安全属性违反问题,其中,针对空指针解引用问题属于半完全规约的程序修复问题.形成的 FootPatch 方法是一个集成的工具,其中,解决空指针解引用问题的程序规约 S 使用分离逻辑手工编写且是否完整不确定;同时,修复空指针解引用问题后是否改变原程序语义也不确定.该方法对空指针解引用的处理具体包括指针为空添加前置条件检查、空指针引用报告异常处理等方式,暂未考虑实例化一个新的指针进行引用等多样化方式.以上多种修改方案都可能会改变原程序语义,但是因为 FootPatch 采用静态方法进行修复,不会过拟合类似动态测试用例提供的期望语义.在给定的实验中,成功地修复了 24 个空指针解引用错误.

4.3 其他程序规约

1) 测试用例的修复

2010 年,Daniel 等人[80]提出了 ReAssert 自动修复发生错误的测试用例.当一个测试用例执行失败时,可能的情况是被测程序出错或者测试用例出错.ReAssert 对执行出错的变量值和控制流进行动态分析,进而修改错误

赋值和断言,并尽可能地保持原测试用例的逻辑功能.ReAssert 利用以上策略对执行失败的测试用例进行修改,使得原执行失败的测试用例能够执行成功.其潜在假设是程序行为正确,测试用例执行成功则符合程序规约 *S*.因此,ReAssert 无法判定测试用例出错情况以及测试用例正确的执行逻辑.修复后的测试用例以建议的方式给出,其正确性还需要人工进行确认.

2016 年,Gao 等人[81]提出了 SITAR 自动修复 GUI 测试脚本.在回归测试中,由于原 GUI 程序图形组件的变化,使得原测试用例的事件或操作序列等输入不再适应当前的测试脚本,期望测试的 GUI 对象对应的断言和检查点等测试预言(test oracle)也需要适应性地更新.SITAR 通过逆向工程生成的事件流图和用户输入修复原有测试脚本,使其在新的 GUI 程序中可用.

2) 崩溃错误和资源竞争

2015 年,高庆等人[82]提出了基于问答系统的特定修复方法,利用 Q&A 问答系统中的知识修复崩溃错误.发生崩溃错误的原因很多,崩溃时程序行为如何不确定,目前不能确定地给出完整的程序规约 *S*.该方法具体通过抽取崩溃路径信息(crash trace)包含的行号作为候选出错位置,提炼崩溃报的错信息构造一组查询,检索 Stack Overflow 问答系统获得和崩溃相关的问题和回答页面列表.然后,通过模糊程序分析技术(fuzzy program analysis technique)形成修复的样例,并利用结构相似度和文本相似度进一步过滤噪声样例.利用编辑脚本将样例转化为最终的修复补丁.实验中,通过手工确认,表明可以修复实际的崩溃错误.具体在收集了 24 个可复发崩溃错误数据集上,该方法能够修复其中的 10 个崩溃错误,其中 8 个修复结果正确.

2016 年,Wang 等人[83]提出了 ARROW,针对现代浏览器软件的并发执行引起的 Web 应用程序资源竞争问题进行自动修复.例如,客户端 Web 页面包含各种类型脚本(HTML、JS 和样式表等),在并发渲染和异步加载时,这些异步事件和用户输入事件相互竞争资源并以非确定性的顺序执行,可能引起网络延迟、执行异常等问题.Web 页面异步加载和用户请求资源竞争问题存在很多执行交错,也不能完全串行,刻画的程序规约 *S* 是否完全符合预期行为不确定.ARROW 以浏览器渲染各页面元素的前后依赖关系,将 Web 页面静态建模为因果图,通过 Web 页面源码获取开发人员预期的各元素依赖关系.最后,利用求解器对构造的约束进行求解,使得两者不会出现不一致的情况.

3) 缓冲区溢出和整形错误

2016 年,Gao 等人[84]提出了 BovInspector,能够自动检测和修复缓冲区溢出漏洞.为了过滤误报,对静态分析的错误检测结果进行可达性分析,并在可达性指导下进行符号执行收集路径约束条件和指定的溢出约束条件,通过比对两者确定真正的缓冲区溢出漏洞.对确认的缓冲区溢出漏洞采用 3 种修复模板 *OP*:添加边界检查、替换为更安全的 API 和扩展缓冲区空间.以上修复操作都可能改变原程序语义,程序规约 *S* 完整性不确定,修复结果需要人工检查.在给定的实验中,BovInspector 能够以平均 74.9%的准确率检测该类漏洞,并全部产生正确的修复补丁.

2017 年,Cheng 等人[85]通过推测合适的变量和表达式数据类型,自动生成整形错误的补丁,提出交互式的修复方法 IntPTI.该方法通过静态值分析近似表达式的值,并收集其可能的正确类型约束,然后转化为约束求解问题,推测可能的正确数据类型来生成补丁,最后,通过 Web 界面展示供开发人员交互式选择和确认正确的补丁.具体来说,整形错误的程序规约 *S* 由静态分析收集约束并求解获得,其规约完整性不确定;同时,修复后是否引起原程序语义变化也不确定.该方法设计了 3 种修复模板 *OP*:完整性检查(sanity check)、显式类型转换(explicit type casting)和改变申明类型(declared type changing).补丁内容 *C* 即推断出的正确数据类型通过约束求解获得.IntPTI 在收集的 7 个实际项目数据集上实验,能够成功地使 25 个错误中的 23 个补丁被用户确认和采纳.其实,2 个误报是由于采用流不敏感的静态分析技术进行正确的数据类型推断所引起的.

4) 语法错误修复

2017 年,Gupta 等人[86]首次提出了 Deepfix,利用深度学习技术直接生成补丁.这种方法的修复对象是语法错误,将程序抽象为以语句为粒度的序列,利用多层次的神经网络模型来预测出错位置和正确的语句,其修复精度取决于从训练集中学习的神经网络.而神经网络模型的质量取决于特征向量的选择和训练集的质量.该模型表

示的程序规约 S 的完整性不确定.该方法相当于将学习获得的神经网络模型作为程序规约 S,对学生的句法错误修复实验显示,其完全修复的准确率在 27%,在其他更复杂的缺陷上的表现尚不清楚.

5) 搜索语句修复

2014 年,Gopinath 等人[87]提出一种利用面向数据的语言 ABAP 修复 SELECT 语句错误的方法.具体利用数据分布中所隐含的信息,使用半监督的学习方法识别正确行为,修复 SELECT 语句中 WHERE 子句附带的条件错误.该方法学习获得的程序规约 S 完整性不确定,需要人工确认修复结果.

6) 用户体验

2018 年,Sonal 等人[88]提出了 MFix 方法自动修复手机中网页的友好显示问题.由于很多网站不是专门为手机设备设计和开发的,这会导致在手机上显示文本不可读、导航混乱、内容溢出手机设备显示窗体等问题,手机上网页友好显示的问题规约 S 由用户手工确认.MFix 方法主要关注字体大小、点击目标间距、内容缩放调整等问题,通过自动生成层叠样式表(cascading style sheet,简称 CSS)补丁来优化上述问题的友好显示.MFix 首先建立基于图的网页布局模型;然后对这些图进行强制编码以增强显示友好性,同时最小化布局的损坏,从而生成 CSS 补丁.该方法使用访问频度靠前的 38 个网站主页进行评估实验,能够成功解决占比 95%的网站在手机上友好显示的问题.

4.4 小 结

半完全规约的程序修复问题对应的方法扩展了程序规约 S 的刻画方式和使用范畴,所使用的契约、形式规约和学习的行为模型等程序规约有效补充了测试用例不足的问题.该类方法存在的主要问题是:输出的补丁后期需要人工确认或者正确性证明,用于大规模程序错误的修复非常困难.同时,现实世界中自带契约、形式规约等程序规约的待修复问题非常少,很多问题的形式规约需要手工构造.

5 总结和展望

根据上述文献研究结果可以得出:程序自动修复技术虽然研究历史较短,但得到了学术界持续的高热度关注,并取得了大量的研究成果.虽然目前暂时还没有工业界的应用,但一系列的研究成果表明,程序自动修复技术已经在一定程度上具备自动修复实际应用中简单错误的能力.虽然国内对该问题的研究起步较晚,但近年来国内的研究情况令人欣慰,包括北京大学、国防科技大学、南京大学、武汉大学、上海交通大学和中国科学院软件研究所等单位的研究非常活跃,在顶级期刊发表的一系列研究成果得到了国外同行的认可.本文提出一种基于规约的程序自动修复描述,并从程序规约的角度将问题进行分类梳理,程序规约 S 的刻画方式直接影响着补丁生成函数 $P=patch(L,OP,C)$的求解过程和补丁判定条件的构造.从程序自动修复对象是否具有完整的程序规约 S 这个关键问题进行分类,对错误修复的不同场景和技术体系进行分类阐述.梳理了各类修复方法的研究进展,阐述了各类研究问题和方法的异同、研究重点和可能存在的问题.

程序自动修复的研究领域中机遇和挑战并存,有待更多的研究者们取得创新和突破.我们认为,该领域还存在如下值得进一步研究的问题.

(1) 程序自动修复技术给传统的错误定位提供了新的应用场景,传统的错误自动定位目的是辅助人工进行错误修复.辅助人工和辅助机器进行错误修复是不同的问题,他们要求的精度不同.例如,人工修复更倾向于定位到函数级,而机器自动修复更需要语句级甚至表达式级别的精确位置.传统的错误定位更多的研究关注于错误语句的位置排序和可能出错位置的最小集,而对出错语句可疑值本身的精确性和语句内部可疑错误位置研究不足.辅助人和软件进行错误自动修复所需的错误位置精度不同,到目前为止,还没有出现专门针对程序自动修复技术而设计的错误高精度自动定位方法.针对程序自动修复场景,设计更细粒度和更高精度的错误定位技术值得深入研究.

(2) 针对不完全规约的程序自动修复问题,即直接或间接地(基于测试集提取的条件约束)使用测试集作为待修复程序规约 S,高精度修复技术还有待进一步研究.一般的程序错误中,条件错误和函数调用错误占比更高,因此,对表达式条件或更复杂的条件错误、接口错误的修复值得深入研究.由于弱测试用

例问题依然存在,如何利用更细粒度的源代码静态信息、代码执行的动态信息以及其他测试用例之外的信息加强程序规约,从而加强对输出补丁的正确性判定条件和增强的规约指导补丁生成都是重要的研究问题.研究更多的程序规约刻画方法用于补丁质量判定,例如借鉴传统的程序验证和证明技术,结合具体程序修复场景进一步判定测试用例集无法区分的补丁正确性问题.多行错误、互有依赖的多个错误以及跨项目错误的程序自动修复是更复杂的问题,是同样值得研究和探讨的困难问题.

(3) 针对完全规约的程序自动修复问题,即待修复程序规约 S 能够完整刻画的修复问题,需要更多实例基础,更多类型特定错误的发生原因和人工修复方法有待充分的实证研究.基于充分的实证研究基础,更多具有完全规约的程序自动修复问题尚待进一步发掘,由于修复错误而引入的程序语义变化的合理性需要充分讨论.在提高修复精度的同时,修复的补丁质量(例如补丁可读性和人工修复的补丁质量近似等)也是重要的研究问题.另外,将多种特定类型错误修复技术结合,例如与缺陷自动分类技术结合来提升特定错误修复技术的可扩展性和修复能力.

(4) 对于半完全规约的程序自动修复问题,其待修复程序规约 S 是否能够完整刻画不确定.在程序自动修复场景中先假设有完全规约,最终生成的补丁程序正确性校验是核心问题.尤其面对程序规模较大的情况下,如何结合程序验证和证明技术自动进行正确性判定是困难问题.当然,对于完全规约和半完全规约程序修复问题本身也值得进一步研究,充分讨论其内涵和外延有助于进一步扩展程序自动修复技术所面向的问题.

统一的程序自动修复技术评价标准,测试用例和 banchmark 不足依然是面临的客观问题.测试用例自动生成和测试用例修复相关技术为程序自动修复提供有效支撑,更大和更符合错误自然分布的banchmark有待进一步建立.各修复技术的横向对比分析和统一的评价标准有待丰富,从而促进修复技术的持续发展和应用.

总体来讲,程序自动修复技术最终要解决的核心问题是修复真实 bug,针对实际问题的任何改进都值得深入研究.例如,待修复程序的规模,即如何修复规模尽可能大的程序中包含的真实 bug;修复能力,即如何修复尽可能多的真实 bug 和涵盖更广泛的错误类型;补丁质量,即在保证生成补丁正确性的前提下,如何使工具自动生成的补丁和人工修复补丁更接近,从而更容易被开发者接受.针对该热点研究,未来的机遇和挑战并存、荣耀和艰辛同在.

References:

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

[2] Anvik J, Hiew L, Murphy GC. Who should fix this bug? In: Proc. of the 28th Int'l Conf. on Software Engineering. 2006. 361−370.

[3] https://www.mozilla.org/en-US/security/client-bug-bounty

[4] https://blog.chromium.org/2010/01/encouraging-more-chromium-security.html

[5] https://technet.microsoft.com/en-us/security/dn425036

[6] https://en.wikipedia.org/wiki/Deep_Impact_(spacecraft)

[7] Xuan JF, Ren ZL, Wang ZY, Xie XY, Jiang H. Progress on approaches to automatic program repair. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):771−784 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4972.htm [doi: 10.13328/j.cnki.jos.004972]

[8] Wang Z, Gao J, Chen X, Fu HJ, Fan XY. Automatic program repair techniques: A survey. Chinese Journal of Computers, 2018, 41(3):588−610 (in Chinese with English abstract).

[9] Goues CL, Forrest S, Weimer W. Current challenges in automatic software repair. Software Quality Journal, 2013,21(3):421−443.

[10] Monperrus M. Automatic software repair: A bibliography. ACM Computing Surveys, 2018,51(1):Article No.17.

[11] Le XBD, Thung F, Lo D, Le Goues C. Overfitting in semantics-based automated program repair. In: Proc. of the Empirical Software Engineering. 2018. 1−27.

[12] Mechtaev S, Nguyen MD, Noller Y, Grunske L, Roychoudhury A. Semantic program repair using a reference implementation. In: Proc. of the 40th Int'l Conf. on Software Engineering (ICSE 2018). 2018. 11−22.

[13] Andersen J, Lawall JL. Generic patch inference. Automated Software Engineering, 2010,17(2):119−148.

[14] Staats M, Whalen MW, Heimdahl MPE. Programs, tests, and oracles: The foundations of testing revisited. In: Proc. of the Int'l Conf. on Software Engineering. 2011. 391−400.

[15] Gao Q, Xiong Y, Mi Y, Zhang L, Yang W, Zhou Z, Xie B, Mei H. Safe memory-leak fixing for c programs. In: Proc. of the 37th Int'l Conf. on Software Engineering (ICSE 2015), Vol.1. Piscataway: IEEE Press, 2015. 459−470.

[16] Wong WE, Gao R, Li Y, Abreu R, Wotawa F. A survey on software fault localization. IEEE Trans. on Software Engineering, 2016, 42(8):707−740.

[17] Yu K, Lin MX. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011,34(8):1411−1422 (in Chinese with English abstract).

[18] Chen X, Ju XL, Wen WZ, Gu Q. Review of dynamic fault localization approaches based on program spectrum. Ruan Jian Xue Bao/ Journal of Software, 2015,26(2):390−412 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

[19] Jones JA, Harrold MJ, Stasko J. Visualization of test information to assist fault localization. In: Proc. of the 24th Int'l Conf. on Software Engineering (ICSE 2002). 2002. 467−477.

[20] Tassey G. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, 2002,15(3):125−125.

[21] Dijkstra EW. Notes on structured programming. In: Dahl OJ, Dijkstra EW, Hoare CAR, eds. Proc. of the Structured Programming. Academic Press, 1972. 1−82.

[22] Qi Z, Long F, Achour S, Rinard M. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis. ACM Press, 2015. 24−36.

[23] Smith EK, Barr ET, Goues CL, Brun Y. Is the cure worse than the disease? Overfitting in automated program repair. In: Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM Press, 2015. 532−543.

[24] Le Goues C, Nguyen T, Forrest S, Weimer W. GenProg: A generic method for automatic software repair. IEEE Trans. on Software Engineering, 2012,38(1):54−72.

[25] D'Antoni L, Samanta R, Singh R. Qlose: Program repair with quantitative objectives. In: Proc. of the Int'l Conf. on Computer Aided Verification. Cham: Springer-Verlag, 2016. 383−401.

[26] Su XH, Yu Z, Wang TT, Ma PJ. A survey on exposing, detecting and avoiding concurrency bugs. Chinese Journal of Computers, 2015,38(11):2215−2233 (in Chinese with English abstract).

[27] Wei Y, Pei Y, Furia CA, Silva LS, Buchholz S, Meyer B, Zeller A. Automated fixing of programs with contracts. In: Proc. of the 19th Int'l Symp. on Software Testing and Analysis. ACM Press, 2010. 61−72.

[28] Long F, Rinard M. An analysis of the search spaces for generates and validates patch generation systems. In: Proc. of the IEEE/ ACM 38th Int'l Conf. on Software Engineering (ICSE). IEEE, 2016. 702−713.

[29] Le XBD, Le TDB, Lo D. Should fixing these failures be delegated to automated program repair? In: Proc. of the IEEE 26th Int'l Symp. on Software Reliability Engineering. IEEE Computer Society, 2015.427−437.

[30] Tan SH, Yi J, Mechtaev S, Roychoudhury A. Codeflaws: A programming competition benchmark for evaluating automated program repair tools. In: Proc. of the 39th Int'l Conf. on Software Engineering Companion. IEEE, 2017. 180−182.

[31] Mechtaev S, Yi J, Roychoudhury A. Angelix: Scalable multiline program patch synthesis via symbolic analysis. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM Press, 2016. 691−701.

[32] Le XBD, Lo D, Goues CL. Empirical study on synthesis engines for semantics-based program repair. In: Proc. of the IEEE Int'l Conf. on Software Maintenance and Evolution. IEEE, 2017. 423−427.

[33] Le XBD, Lo D, Goues CL. History driven program repair. In: Proc. of the Int'l Conf. on Software Analysis, Evolution, and Reengineering. IEEE, 2016. 213−224.

[34] Kim D, Nam J, Song J, Kim S. Automatic patch generation learned from human-written patches. In: Proc. of the 2013 Int'l Conf. on Software Engineering. IEEE Press, 2013. 802−811.

[35] Suzuki R, Suzuki R, Suzuki R, Polozov O, Gulwani S, Gheyi R, Hartmann B. Learning syntactic program transformations from examples. In: Proc. of the 39th Int'l Conf. on Software Engineering. IEEE Press, 2017. 404−415.

[36] Xiong Y, Wang J, Yan R, Zhang J, Han S, Huang G, Zhang L. Precise condition synthesis for program repair. In: Proc. of the 39th Int'l Conf. on Software Engineering. IEEE Press, 2017. 416−426.

[37] Zhang X, Gupta N, Gupta R. Locating faults through automated predicate switching. In: Proc. of the 28th Int'l Conf. on Software Engineering. ACM Press, 2006. 272−281.

[38] Saha RK, Lyu Y, Yoshida H, Prasad MR. Elixir: Effective object-oriented program repair. In: Proc. of the 2017 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2017. 648−659.

[39] Chen L, Pei Y, Furia CA. Contract-based program repair without the contracts. In: Proc. of the 2017 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2017. 637−647.

[40] Qi X, Reiss SP. Leveraging syntax-related code for automated program repair. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering. IEEE Press, 2017. 660−670.

[41] Sumi S, Higo Y, Hotta K, Kusumoto S. Toward improving graftability on automated program repair. In: Proc. of the 2015 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2015. 511−515.

[42] Sidiroglou-Douskos S, Lahtinen E, Long F, Rinard M. Automatic error elimination by horizontal code transfer across multiple applications. ACM SIGPLAN Notices, 2015,50(6):43−54.

[43] Ke Y, Stolee KT, Goues CL, Brun Y. Repairing programs with semantic code search. In: Proc. of the 2015 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2015. 295−306.

[44] Wen M, Chen JJ, Wu RX, Hao D, Cheung SC. Context-aware patch generation for better automated program repair. In: Proc. of the 40th Int'l Conf. on Software Engineering (ICSE 2018). 2018. 1−11.

[45] Hua JR, Zhang MS, Wang KY, Khurshid S. Towards practical program repair with on-demand candidate generation. In: Proc. of the 40th Int'l Conf. on Software Engineering. 2018. 12−23.

[46] Mechtaev S, Yi J, Roychoudhury A. Directfix: Looking for simple program repairs. In: Proc. of the 37th Int'l Conf. on Software Engineering. Vol.1. IEEE Press, 2015. 448−458.

[47] Do H, Elbaum S, Rothermel G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Software Engineering, 2005,10(4):405−435.

[48] Cadar C, Dunbar D, Engler D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. of the Usenix Conf. on Operating Systems Design and Implementation. USENIX Association, 2009. 209−224.

[49] Nguyen HDT, Qi D, Roychoudhury A, Chandra S. Semfix: Program repair via semantic analysis. In: Proc. of the 2013 35th Int'l Conf. on Software Engineering (ICSE). IEEE, 2013. 772−781.

[50] Le Goues C, Dewey-Vogt M, Forrest S, Weimer W. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In: Proc. of the 2012 34th Int'l Conf. on Software Engineering (ICSE). IEEE, 2012. 3−13.

[51] Xuan JF, Martinez M, DeMarco F, Clement M, Marcote SL, Durieux T, Le Berre D, Monperrus M. Nopol: Automatic repair of conditional statement bugs in Java programs. IEEE Trans. on Software Engineering, 2017,43(1):34−55.

[52] DeMarco F, Xuan JF, Berre DL, Monperrus M. Automatic repair of buggy if conditions and missing preconditions with smt. In: Proc. of the Int'l Workshop on Constraints in Software Testing, Verification, and Analysis. Hyderabad, 2014. 30−39.

[53] Qi X, Reiss SP. Identifying test-suite-overfitted patches through test case generation. In: Proc. of the 26th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM Press, 2017. 226−236.

[54] Kong X, Zhang L, Wong WE, Li B. Experience report: How do techniques, programs, and tests impact automated program repair? In: Proc. of the IEEE 26th Int'l Symp. on Software Reliability Engineering. IEEE Computer Society, 2015. 194−204.

[55] Qi Y, Mao X, Lei Y, Dai Z, Wang C. The strength of random search on automated program repair. In: Proc. of the 36th Int'l Conf. on Software Engineering. ACM Press, 2014. 254−265.

[56] Ackling T, Alexander B, Grunert I. Evolving patches for software repair. In: Proc. of the 13th Annual Conf. on Genetic and Evolutionary Computation. ACM Press, 2011. 1427−1434.

[57] Assiri FY, Bieman JM. An assessment of the quality of automated program operator repair. In: Proc. of the 2014 IEEE 7th Int'l Conf. on Software Testing, Verification and Validation (ICST). IEEE, 2014. 273−282.

[58] Tan SH, Yoshida H, Prasad MR, Roychoudhury A. Anti-patterns in search-based program repair. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2016. 727−738.

[59] Fan L, Rinard M. Automatic patch generation by learning correct code. In: Proc. of the ACM Sigplan-Sigact Symp. on Principles of Programming Languages. ACM Press, 2016. 298−312.

[60] Xiong YF, Liu XY, Zeng MH, Zhang L, Huang G. Identifying patch correctness in test-based program repair. In: Proc. of the 40th Int'l Conf. on Software Engineering. ACM Press, 2018. 789−799.

[61] Jin G, Song L, Zhang W, Lu S, Liblit B. Automated atomicity-violation fixing. In: Proc. of the ACM Sigplan Conf. on Programming Language Design and Implementation. 2011. 389−400.

[62] Jin G, Zhang W, Deng D, Liblit B, Lu S. Automated concurrency-bug fixing. In: Proc. of the Usenix Conf. on Operating Systems Design and Implementation. 2012. 221−236.

[63] Park S, Lu S, Zhou Y. CTrigger: Exposing atomicity violation bugs from their hiding places. ACM Sigarch Computer Architecture News, 2009,37(1):25−36.

[64] Liu P, Zhang C. Axis: Automatically fixing atomicity violations through solving control constraints. In: Proc. of the Int'l Conf. on Software Engineering. IEEE, 2012. 299−309.

[65] Liu P, Tripp O, Zhang C. Grail: Context-aware fixing of concurrency bugs. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2014. 318−329.

[66] Liu H, Chen Y, Lu S. Understanding and generating high quality patches for concurrency bugs. In: Proc. of the ACM Sigsoft Int'l Symp. on Foundations of Software Engineering. ACM Press, 2016. 715−726.

[67] Joshi P, Naik M, Sen K, Gay D. An effective dynamic analysis for detecting generalized deadlocks. In: Proc. of the 18th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2010. 327−336.

[68] Cai Y, Cao L. Fixing deadlocks via lock pre-acquisitions. In: Proc. of the Int'l Conf. on Software Engineering. IEEE, 2016. 1109−1120.

[69] Zhang W, Sun C, Lu S. ConMem: Detecting severe concurrency bugs through an effect-oriented approach. ACM SIGARCH Computer Architecture News, 2010,38(1):179−192.

[70] van Tonder R, Le Goues C. Static automated program repair for heap properties. In: Proc. of the 40th Int'l Conf. on Software Engineering (ICSE 2018). 2018. 151−162.

[71] Nistor A, Chang PC. CARAMEL: Detecting and fixing performance problems that have non-intrusive fixes. In: Proc. of the 2015 IEEE/ACM 37th IEEE Int'l Conf. on Software Engineering (ICSE). IEEE, 2015. 902−912.

[72] Weiss A, Guha A, Brun Y. Tortoise: Interactive system configuration repair. In: Proc. of the 2017 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2017. 625−636.

[73] Xiong Y, Zhang H, Hubaux A, *et al*. Range fixes: Interactive error resolution for software configuration. IEEE Trans. on Software Engineering, 2015,41(6):603−619.

[74] Pei Y, Wei Y, Furia CA, Nordio M, Meyer B. Code-based automated program fixing. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM Press, 2011. 392−395.

[75] Yu P, Furia CA, Nordio M, Meyer B. Automatic program repair by fixing contracts. In: Proc. of the Int'l Conf. on Fundamental Approaches to Software Engineering. Springer-Verlag, 2014. 246−260.

[76] Pei Y, Furia CA, Nordio M, Meyer B. Automated program repair in an integrated development environment. In: Proc. of the 37th Int'l Conf. on Software Engineering, Vol.2. IEEE Press, 2015. 681−684.

[77] Jobstmann B, Griesmayer A, Bloem R. Program repair as a game. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin, Heidelberg: Springer-Verlag, 2005. 226−238.

[78] Essen CV, Jobstmann B. Program repair without regret. In: Proc. of the Int'l Conf. on Computer Aided Verification. Springer-Verlag, 2013. 896−911.

[79] Kneuss E, Koukoutos M, Kuncak V. Deductive program repair. In: Proc. of the Int'l Conf. on Computer Aided Verification. Springer Int'l Publishing, 2015. 217−233.

[80] Daniel B, Jagannath V, Dig D, Marinov D. ReAssert: Suggesting repairs for broken unit tests. In: Proc. of the 24th IEEE/ACM Int'l Conf. on Automated Software Engineering. 2010. 433−444.

[81] Gao Z, Chen Z, Zou Y, Memon AM. SITAR: GUI test script repair. IEEE Trans. on Software Engineering, 2016,42(2):170−186.

[82] Gao Q, Zhang HS, Wang J, Xiong YF. Fixing recurring crash bugs via analyzing Q&A sites. In: Proc. of the 2015 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2015. 307−318.

[83] Wang W, Zheng Y, Liu P, Xu L, Zhang X, Eugster P. ARROW: Automated repair of races on client-side Web pages. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2016. 201−212.

[84] Gao F, Wang L, Li X. BovInspector: Automatic inspection and repair of buffer overflow vulnerabilities. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM Press, 2016. 786−791.

[85] Cheng X, Zhou M, Song X, *et al*. IntPTI: Automatic integer error repair with proper-type inference. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. IEEE, 2017. 996−1001.

[86] Gupta R, Pal S, Kanade A, Shevade S. Deepfix: Fixing common C language errors by deep learning. In: Proc. of the 31st AAAI Conf. on Artificial Intelligence (AAAI). 2017. 1345−1351.

[87] Gopinath D, Khurshid S, Saha D, Chandra S. Data-guided repair of selection statements. In: Proc. of the 36th Int'l Conf. on Software Engineering. 2014. 243−253.

[88] Mahajan S, Abolhassani N, McMinn P, Halfond GJ. Automated repair of mobile friendly problems in Web pages. In: Proc. of the 40th Int'l Conf. on Software Engineering. ACM Press, 2018. 140−150.

附中文参考文献:

[7] 玄跻峰,任志磊,王子元,谢晓园,江贺.自动程序修复方法研究进展.软件学报,2016,27(4):771−784. http://www.jos.org.cn/1000-9825/4972.htm [doi: 10.13328/j.cnki.jos.004972]

[8] 王赞,郜健,陈翔,傅浩杰,樊向宇.自动程序修复方法研究述评.计算机学报,2018,41(3):588−610.

[17] 虞凯,林梦香.自动化软件错误定位技术研究进展.计算机学报,2011,34(8):1411−1422.

[18] 陈翔,鞠小林,文万志,顾庆.基于程序频谱的动态缺陷定位方法研究.软件学报,2015,26(2):390−412. http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

[26] 苏小红,禹振,王甜甜,马培军.并发缺陷暴露、检测与规避研究综述.计算机学报,2015,38(11):2215−2233.

李斌(1985－),男,甘肃天水人,博士生,工程师,主要研究领域为软件分析和缺陷修复,安全操作系统.

马恒太(1970－),男,博士,副研究员,主要研究领域为软件安全分析,操作系统安全.

贺也平(1962－),男,博士,研究员,博士生导师,主要研究领域为系统安全,隐私保护.