软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(5):1256−1268 [doi: 10.13328/j.cnki.jos.005716]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

示例演化驱动的学生程序自动修复*

王甜甜[1], 许家欢[1], 王克朝[1,2], 苏小红[1]

[1](哈尔滨工业大学 计算机科学与技术学院,黑龙江 哈尔滨 150001)

[2](哈尔滨学院 信息工程学院,黑龙江 哈尔滨 150086)

通讯作者: 王甜甜, E-mail: sweetwtt@126.com

摘 要: 已有的程序自动化调试研究大多面向工业软件,而学生程序调试具有缺陷数多、类型复杂等特有难点问题,因此,针对学生程序设计应用背景,研究程序自动修复方法,利用模板示例程序指导补丁的演化.改进了遗传编程算法,包括适应度的计算、变异体的生成方式和变异位置及操作的选择方式,使其更加适合修复学生程序.提出了基于示例的静态错误定位方法,能够识别缺陷程序和参考程序差异和可能的变异操作,有效地缩小补丁的搜索空间以提高修复的准确性.提出了基于执行值序列的变量映射方法,以降低变异体的编译错误,提高修复的准确性.在此基础上,设计并实现了示例演化驱动的Java学生程序自动修复系统.实验结果表明,该方法可以修复含有多缺陷学生程序,对于所用的测试集,当学生程序只有 1 个~2 个错误时,修复率将近 100%;当含有 3 个缺陷时,修复率约为 70%;当含有 4 个及以上缺陷时,修复率约为 50%.

关键词: 自动修复;示例演化;遗传编程;错误定位;变量映射

中图法分类号: TP311

中文引用格式: 王甜甜,许家欢,王克朝,苏小红.示例演化驱动的学生程序自动修复.软件学报,2019,30(5):1256−1268. http://www.jos.org.cn/1000-9825/5716.htm

英文引用格式: Wang TT, Xu JH, Wang KC, Su XH. Example-evolution-driven automatic repair of student programs. Ruan Jian Xue Bao/Journal of Software, 2019,30(5):1256−1268 (in Chinese). http://www.jos.org.cn/1000-9825/5716.htm

Example-evolution-driven Automatic Repair of Student Programs

WANG Tian-Tian[1], XU Jia-Huan[1], WANG Ke-Chao[1,2], SU Xiao-Hong[1]

[1](School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

[2](School of Information Engineering, Harbin University, Harbin 150086, China)

Abstract: Most existing program repair researches are oriented to industrial software. Student program debugging has many unique problems, such as multiple bugs and complex bug types. Therefore, according to the application background of student programming, the automatic repair method is studied, and template programs are used to guide the evolution of patches. Genetic programming algorithm has been improved, such as fitness calculation, mutants generation, and mutation position and operator selection, to make it more suitable for repairing student programs. A static fault location method based on sample programs is proposed, which identifies the difference between the defect program and the sample program and recognizes the possible mutation operators. It can effectively reduce the search space of the patch and improves the accuracy of the program repair. A variable mapping method based on execution value sequence is proposed to reduce compilation errors of mutants and improve the accuracy of program repair. On this basis, an example-evolution-driven system for

* 基金项目: 国家自然科学基金(61672191); 国家重点研发计划(2018YFB1004800); 黑龙江省自然科学基金(JJ2019LH0048H); 哈尔滨科技创新人才研究专项资金(2016RAQXJ013)

Foundation item: National Natural Science Foundation of China (61672191); National Key Research and Development Program of China (2018YFB1004800); Natural Science Foundation of Heilongjiang Province (JJ2019LH0048H); Special Research Fund for Science and Technology Innovation Talents of Harbin Municipality (2016RAQXJ013)

本文由智能化软件新技术专刊特约编辑申富饶教授和李戈副教授推荐.

收稿时间: 2018-08-30; 修改时间: 2018-10-31, 2018-12-14; 采用时间: 2019-02-03

repairing students' Java programs was designed and implemented. The experimental results show that the method can repair student programs with multiple bugs. For the test set, the repair rate is nearly 100% when the student programs have only 1~2 bugs. When there are 3 bugs, the repair rate is about 70%. When there are 4 or more bugs in the student programs, the repair rate is about 50%.

Key words: automatic repair; example evolution; generic programming; fault localization; variable mapping

程序自动化调试有助于提高软件开发和维护的效率,提高软件的质量.因此,研究人员针对工业软件错误定位[1,2]和自动化修复[3,4]技术开展了大量的研究.

近年来,大规模网络公开课(massive open online courses,简称 MOOC)得到了广泛的关注,EdX、Cousera、Udacity 上的优秀课程吸引了几千名学生.然而,对于拥有大规模的学生数的 MOOC,教师很难与每个学生都一一进行点评互动.这给信息技术带来了新的挑战:如何自动化地评价学生的学习效果、提供充分的反馈,与学生互动?

对于实践性较强的程序设计课程而言,该问题的解决尤为重要.一方面,学生程序表示形式以及缺陷可能多种多样,教师理解这些程序是很困难的;另一方面,程序调试是一项很困难的任务:如果没有自动化技术的辅助,从源代码中定位到一个错误则是一项具有挑战性的工作.即使已经定位到错误,依旧需要大量的时间来分析问题的根源并做出适当的修正.教师人工判别学生程序中的错误进而指出修正方法,不仅工作量大,而且效率低,可能花费几天的时间才能给出反馈.当学生收到反馈后,需要重新回顾解题思路,这常常打消了他们进一步分析程序的积极性.因此,学生程序的自动化调试及评价反馈,也成为目前的一个研究热点.

程序自动修复方法通常由两个步骤构成:首先定位可疑语句位置,然后优选可疑度大的语句进行修复.学生程序中经常可能含有概念错误,错误密度也远远大于工业代码,缺陷的种类可能多种多样,甚至可能缺少关键语句,因此可能对所有的测试用例都执行失效.该特点导致学生在错误定位和修复这两个步骤上和工业软件调试都有所区别.

- 首先,工业界中的缺陷程序一般近似正确,并且补丁较小,错误定位方法研究基于的假设是"缺陷程序可以对某些测试用例执行成功,而对另一些测试用例执行失效"[5].通过对比成功执行和失效执行信息来定位可疑语句,例如,GenProg[6]等程序修复工具通常采用程序谱故障定位方法.然而学生程序中可能含有较大概念错误,导致程序对所有测试用例均产生失效,该情况下无法对比成功测试用例和失效测试用例的执行信息,因此需要研究适合于分析学生程序的错误定位方法.
- 另外,工业软件难于获得完整的规格说明,通常只能通过测试用例集评价修复程序,限制了程序修复的有效性.然而学生程序的规格说明是明确的,并且在编程练习或考试过程中可获得大量的学生程序,这些学生程序中有很多程序是正确的,可作为潜在的示例程序来指导其他含有缺陷的程序的故障定位和修复.然而实现相同编程任务的学生程序可能具有相似的实现形式,也可能实现方式不同,这给程序分析带来了困难.需要重点研究解决该问题.

本文针对学生程序设计的应用背景,研究程序的自动修复方法.

1 相关研究

1.1 修复学生程序的可行性

工业软件自动化调试的一个主要难点问题在于无法获知一个修正方案是否真正解决了错误根源,还是简单地掩盖了问题.通常,达到的最好解决程度是用测试用例或部分规格说明检验修正方案,这就导致了修正操作的种类、要修正的可疑语句位置、可用的修正元素的搜索空间巨大,进而导致修正方法的开销巨大,并且难于充分地遍历搜索空间,因而可能搜索不到正确补丁.

不同于工业软件,学生程序具有以下特点.

(1) 完整的规格说明是已知的,给定问题的答案是已知的.除了测试用例外,还可以由教师事先编写的完全满足规格说明的参考程序,隐含了期望解的规格说明.另外,MOOC 中学生提交了大量的正确程序,

© 中国科学院软件研究所 http://www.jos.org.cn

可以从这些程序中发现新的模板程序.

(2) 由于学生参加相同的课程,并且完成相同的题目,编写程序的思路甚至所犯的错误通常也相似.

以上两点,使得学生程序的自动化调试具有更强的可行性.

1.2 学生程序自动修复的研究现状

MIT的Singh等人开发了AutoProf[7],利用基于约束的合成算法为程序设计提供反馈.除了提供参考程序外,还需要提供人工定义的形式化错误模型来描述解决给定问题时学生易犯错误的修复规则.基于修复规则自动推导合成缺陷程序的修复版本,使该修复版本与参考程序功能等价.该研究为程序设计调试和反馈提供了一种新的研究思路,但是仍存在局限性:修复能力由人工定义的错误模型决定,而人工定义错误模型比较困难,修复规则通常比较简单,因此只能修改表达式类的简单缺陷,无法新增和移动语句,也不能修复较大的概念错误.另外,基于约束的程序合成方法复杂度比较高.

Purdue 大学的 Kim 等人开发了 APEX[8],输入参考程序和测试用例后,基于符号执行、最大满足求解和动态分析,通过匹配参考程序的成功执行和缺陷程序的失效执行的程序切片,来解释缺陷的产生原因.受符号执行的限制,该方法也只能对简单缺陷提供修改建议,不能对复杂的概念错误,例如修改控制结构、新增和移动语句等给出反馈.另外,该方法无法自动生成修复程序.

Yi 等人研究了当前比较流行的工业软件修复工具在修复学生程序中的性能[9],分析了 GenProg[6],AE[10],Prophet[11]和 Angelix[12]这 4 个工具.它们均采用测试驱动的方法,即:如果补丁程序通过了测试集中所有测试用例,则认为该补丁是一个修复程序.例如,GenProg 采用遗传编程算法,试图从待修复程序的其他地方转移正确的语句,经过不断的演化,来使补丁程序通过所有测试.Yi 等人基于这 4 个工具,利用部分修复策略指导学生编程.即:如果一个程序对之前成功的测试用例依然成功,之前失效的测试用例只要有一个变为成功,则认为该程序是一个部分修复,将部分修复反馈给学生,提示其修改程序.研究结果表明,这些工具的修复率较低.原因是这些工具通常每次只修改一个错误,而学生程序中错误通常较多,需要更复杂的修改;另外,测试只考虑了程序的执行结果,而没有考虑程序的行为和编程规范,指导搜索生成的补丁很可能牺牲了其他重要需求.因此,这些工具不适合直接应用于学生程序修复中.

以上研究给我们带来的启发是,学生程序需求明确且功能简单,教师可以提供充分的满足覆盖要求的测试用例集合作为选择补丁的判定准则.然而对于学生程序自动化调试而言,除了测试用例外,还可以提供教师事先编写的模板程序.模板程序中隐含了期望解的规格说明,使得程序设计自动化调试具有更强的可行性.因此,如果能将模板程序也有效应用于程序修复的演化过程中,指导错误定位和补丁的生成,则有望定位和修复学生程序中缺失语句、存在依赖关系的多处复杂缺陷.

2 总体研究框架

2.1 示例演化驱动的学生程序自动修复研究框架

将模板程序引入程序修复的演化框架中,提出示例演化驱动的学生程序自动修复方法,如图 1 所示,分为以下 4 个组成部分.

(1) 选择示例程序:对源程序和模板程序进行解析,生成抽象语法树,并计算其最大相似度,从而可以从模板程序集中选择与学生程序结构语义最为相似的模板程序作为参考示例程序.

(2) 静态错误定位和差异分析:程序修复的一个关键问题是“修复操作的可选位置以及可用的修复算子相组合,生成了巨大的搜索空间”,学生程序修复也存在该问题.为了有效选择修改位置、编辑操作以及从示例程序中转移正确的语句,本文提出了基于示例的静态错误定位和差异分析方法.一方面,通过匹配学生程序抽象语法树和示例程序抽象语法树,寻找学生程序中的差异语句节点,作为优先修改位置赋以较高的选择概率;将示例程序中差异节点作为候选的替换或插入节点,从而有效缩小修复转换的搜索空间,提高修复有效性.另一方面,对差异节点进一步分析修复该位置缺陷可能的变异操作,区分

© 中国科学院软件研究所 http://www.jos.org.cn

增加、删除、替换操作的权重,作为选择变异操作的概率.这样有助于根据实际的缺陷类型,选择合适的变异操作,提高变异转换的效率和有效性.由于无需执行程序,该方法可以有效分析无法获得成功执行结果的学生程序.

(3) 动态变量映射:学生程序和示例程序可能使用不同的变量名,该模块基于变量执行值序列识别出学生程序和示例程序中的等价变量,获得变量映射表,为后续变异过程中的变量重命名奠定基础.

(4) 遗传编程演化:在前 3 个模块的基础上,采用遗传编程算法生成修复后的学生程序和修复操作方案.利用示例程序向待修复程序中移植正确的代码,采用将示例程序语法树中的子树插入或替换到学生程序语法树中的方法进行变异,生成的新程序即为一个变异体.在适应度计算时,不但考虑测试用例的通过情况,还考虑补丁程序与示例程序的语法结构相似程度,指导补丁程序向期望的规格说明方向演化.通过在语法树上进行多次交叉、变异操作,可修复多个缺陷.

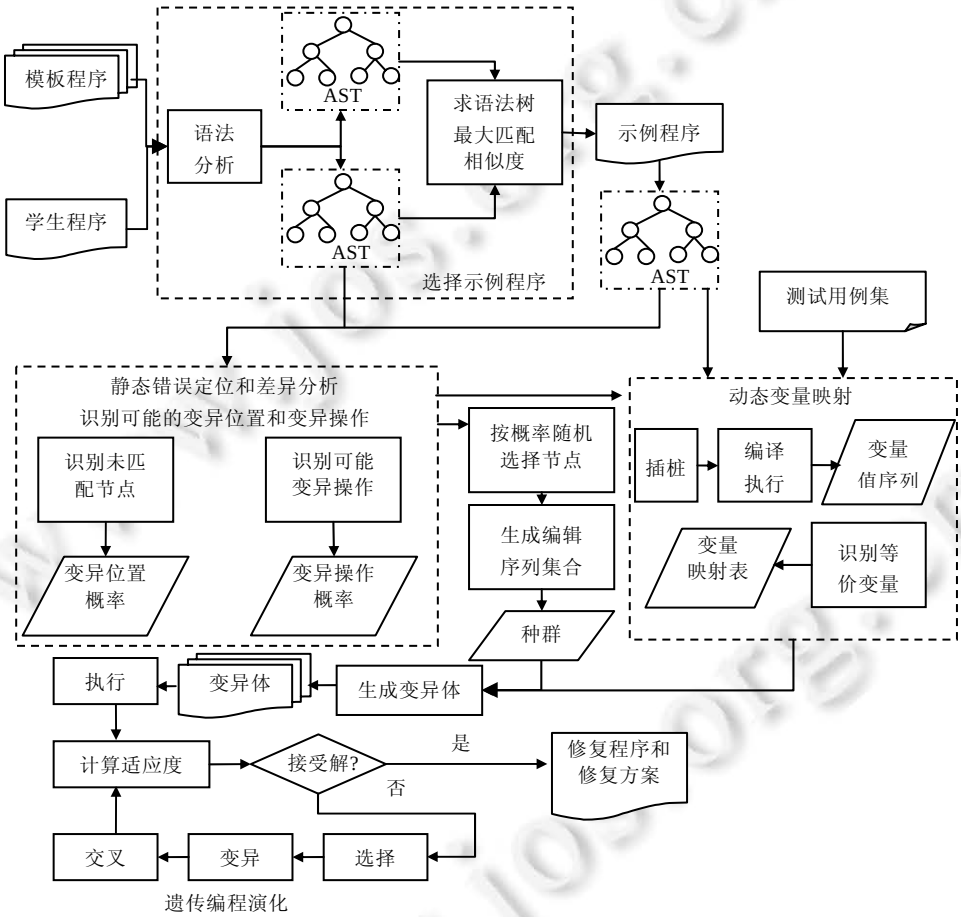

Fig.1 Research framework for example-evolution-driven automatic repair of student programs

图 1 示例演化驱动的学生程序自动修复研究框架

2.2 与GenProg的对比分析

本文方法和 GenProg[9]都是基于遗传编程框架研究程序的自动修复,不同之处见表 1.

(1) 变异来源.GenProg 通过重用、组合待修复程序中的已有语句来生成补丁,不能合成全新的代码,如果修复元素没有出现在程序的其他地方,则无法正确修复.本文方法从示例程序中挖掘语法树子树来进行变异,模板程序可为修复各种复杂缺陷提供充分的素材,也可以在修改时引入新的程序逻辑.

© 中国科学院软件研究所 http://www.jos.org.cn

(2) 错误定位方法.GenProg 采用基于程序谱的错误定位方法,该方法要求程序可以通过某些测试,而学生程序有的时候不能成功执行,因此不适合使用该方法定位可疑语句.本文提出基于示例的错误定位方法,通过对学生程序和模板程序执行上下文匹配,判断学生程序语法树和模板程序语法树的相似节点和不同节点,一方面,识别到可能错误的节点;另一方面,可将模板程序中的差异节点作为候选修复节点.该方法考虑程序的上下文,有助于生成有意义的补丁.

(3) 适应度计算.由于工业软件无法获知规格说明的局限性,GenProg 只使用通过测试用例数作为补丁程序的适应度衡量准则.而学生程序修复中,示例程序中蕴含了需求规格和编程规范,可以将其应用于度量补丁程序的适应度,因此,本文除了考虑测试结果外,还考虑补丁程序和示例程序的相似度,使得适应度的值更加精确,不会出现大量变异体适应度均为 0 的现象,有效指导补丁程序朝着期望演化.

(4) 变量重命名.GenProg 没有考虑不同上下文中变量名的差异,而实际上,实现类似功能的代码在不同的上下文中可能具有不同的变量名.为了提高修复的有效性,本文提出了基于执行值序列的变量映射方法,识别学生程序和示例程序中等价的不同变量名.

(5) 是否支持多缺陷修复.GenProg 虽然能修复多处缺陷,但生成的补丁较简单.本文方法在示例程序的指导下,通过对编辑序列的变异和交叉,可以成功修复含有多个复杂缺陷,即包含多个修复点的程序.

Table 1 Comparison between our method and GenProg

表 1 本文方法与 GenProg 的对比

对比内容	GenProg	本文方法
变异来源	组合待修复程序中的已有语句	示例程序
错误定位方法	程序谱定位方法	语法树匹配和上下文分析
适应度计算	保持成功测试用例数量和 杀死失败测试用例数量	保持成功测试用例数量和杀死失败测试 用例数量和与示例程序的语法结构相似度
变量的映射	不考虑不同上下文变量名差异	基于执行值序列的变量映射
是否支持多缺陷修复	是	是

3 关键技术

3.1 代码多样化问题

实现相同算法的多个程序语法表示形式可能不同.例如图 2 中的程序代码 *A* 与 *B*,它们采用相同的迭代算法实现求 $base^{exp}$ 的功能,但其语法表示形式不同,例如不同的变量名、语句顺序以及控制结构和表达式等,这种情况称为代码多样化.

Code *A* 模板程序	Code *B* 与模板等价的程序	Code *C* 缺陷程序

static public int *fact*(int *base*,int *exp*){
int *i*, *r*=1;
For (*i*=0; *i*<*exp*; *i*++)
*r**=*base*;
return *r*;
}
	static public int fact(int *base*,int *exp*){
	int *j*, *ret*;
	j=0;
	ret=1;
	While (*exp*>*j*){
	j=*j*+1;
	ret=*ret***base*;}
	return *ret*;
	}
		static public int fact(int *base*,int *exp*){
		int *j*, *r*;
		j=0;
		//缺陷,*r* 未初始化
		While (*exp*>*j*){
		j=*j*+1;
		r=*r***base*;}
		return *r*;
		}

调用 *fact*(3,4)时的执行值序列		
base 3	*base* 3	*base* 3
exp 4	*exp* 4	*exp* 4
i 0, 1, 2, 3	*j* 0, 1, 2, 3	*j* 0, 1, 2, 3
r 3, 9, 27, 81	*ret* 3, 9, 27, 81	*r* 随机数构成的序列

Fig.2 Sample programs with code variations

图 2 包含代码多样化的程序示例

© 中国科学院软件研究所 http://www.jos.org.cn

代码多样化给识别学生程序和模板程序的差异带来了困难.为了解决这个问题,我们将代码多样化划分为表达式、控制结构、函数调用、变量名、语句顺序多样化等,并提出了基于结构语义分析识别和消除这些代码多样化[13],进而识别等价的语法结构和表达式等.一方面可以用于减少所需要提供的模板数,对功能等价的含有代码多样化的程序只需提供一个模板;另一方面,提高学生程序和示例程序差异分析的准确性.

3.2 缺陷程序和示例程序的结构语义和执行特征值差异识别

为了有效定位学生程序中多种类型的缺陷,特别是缺失语句和错误变量以及存在依赖关系的复杂缺陷,并且避免代码多样化,特别是语句顺序多样化和变量名多样化影响学生程序和示例程序差异分析的准确性,提出结构语义和执行特征值交互分析方法,使得即使在缺陷程序不能通过任何测试用例的情况下,也可以定位其中的可疑语句,进而有效限定修复的搜索空间,并反馈错误产生原因.

输入示例程序、缺陷程序、测试用例集,目标是识别学生程序和示例程序中存在差异的结构、变量、表达式及语句作为可疑位置,并将差异分析结果以图和值序列的形式进行反馈,辅助理解错误的产生原因,并映射学生程序和模板程序中的变量,用以辅助程序修复.步骤如下.

(1) 对缺陷程序和示例程序进行执行特征值分析,识别匹配变量对.对示例程序中的每个变量值序列分别与学生程序中的每个变量序列使用最长公共子序列算法,得到每两个变量之间的相似度,与该变量值序列相似度最高的学生程序中的变量值序列即为对应的变量值序列.

(2) 采用最长公共子序列算法求缺陷程序和示例程序语法树的最大匹配.

(3) 等价的赋值语句在程序执行过程中有可能受到前面与其存在数据依赖关系的执行语句的影响,得不到等价的特征值序列.为了避免漏检,需要进一步分析未匹配的赋值语句,根据表达式的结构语义,进一步识别结构语义等价的赋值语句和可能匹配变量对.

(4) 输出缺陷程序和示例程序的语法树并标记未匹配的结构和语句,并输出匹配变量对、差异变量及其执行特征值序列,辅助学生理解错误的产生原因.

3.3 动态变量映射

实现相同算法的程序通常具有相似的程序结构特征和执行特征.各变量在执行过程中具有相同的值序列,这些值序列不受代码多样化的影响.通过匹配执行特征值序列,可以消除这种语句顺序和变量名多样化的影响.例如图 2,由于 C 中的变量 j 和 A 中的变量 i 对相同的输入都具有相同的值序列,因此(j,i)为匹配变量对.根据匹配变量对(j,i),则可以将 C 中的 $j=j+1$ 和 A 中的 i++语句准确匹配.

本文通过判定变量执行过程中值序列的相似性进行变量映射,确定学生程序和示例程序中的对应变量,步骤如下.

(1) 采用在语法树上实现程序插桩的方法,在不破坏程序语义的条件下,在程序语法树上的变量初始化和赋值表达式节点之后插入探针语句,用以在后续执行过程中捕获变量的值.

(2) 反向生成代码,并用测试用例执行插桩后的程序,收集输出的变量名和变量值序列.

(3) 通过对比两个程序的变量执行值序列,获得两个程序的对应变量,建立变量映射表.对示例程序 B 中的每个变量值序列分别与学生程序 A 中的每个变量序列使用最长公共子序列算法,得到每两个变量之间的相似度,与该变量值序列相似度最高的学生程序 A 中的变量值序列即为对应的变量值序列.

(4) 在后续进行程序变异时,先判断子树中是否有变量映射表中的变量:如果有,则用变量映射表中与示例程序变量名对应的学生程序变量名对待替换的示例程序子树中的变量进行替换,使变异体可以通过编译和运行.

3.4 基于示例的静态错误定位和差异分析

由于图 2 中缺陷程序 C 中缺少对变量 r 的初始化语句,导致 $r=r^*base$ 和示例程序 A 中的 $r^*=base$ 没有生成等价的执行特征值序列,因而通过动态变量映射无法将(r,r)识别为匹配的变量对.对于这样没有找到匹配变量对的语句,进一步进行表达式标准化和语法匹配,可识别它们结构语义等价,进而将(r,r)识别为可能匹配变量对.

© 中国科学院软件研究所 http://www.jos.org.cn

如图 3 所示,通过程序抽象语法树的子树匹配,判断学生程序语法树和示例程序语法树的相似节点和不同节点,从而定位到可能含有缺陷的节点,并识别可能的变异操作,缩小搜索范围.

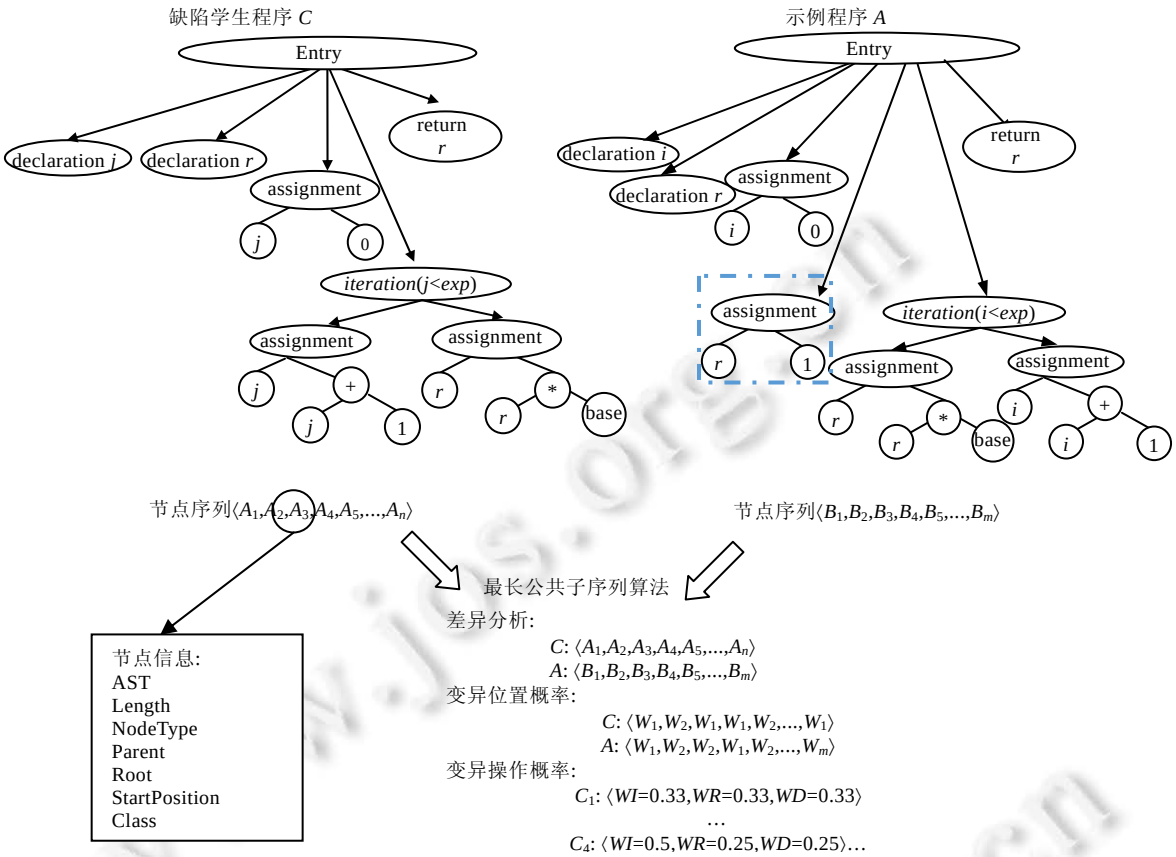

Fig.3 Static fault localization based on example and difference analysis

图 3 基于示例的静态错误定位和差异分析

(1) 使用最长公共子序列算法求学生程序和示例程序抽象语法树序列的最大匹配.

两个节点为相同节点的判断条件是:两个节点对应的字符串值完全匹配且父节点类型相同.

父节点类型相同是为保证学生程序语法树的节点 A 与模板程序语法树的节点 B 在同一结构中,如 A 的父节点是循环结构,那么 B 节点的父节点必须也是循环结构,考虑了程序上下文,保证程序结构的准确性和代码的完整性.由于执行了结构语义分析,可以将含有代码多样化的语句转换为相同内部表示形式.

(2) 记录两个节点序列中相同和不同的节点,对学生程序和示例程序的节点序列中每个节点赋予权值,匹配的节点赋予权值 W_1,不匹配的节点赋予权值 W_2,见公式(1)和公式(2):

$$W_1=\frac{1}{\text{匹配节点数}\times 2+\text{不匹配节点数}} \tag{1}$$

$$W_2=\frac{2}{\text{匹配节点数}\times 2+\text{不匹配节点数}} \tag{2}$$

在变异时随机选择变异节点,权值较高的节点被选中的概率更大,保证与示例程序不匹配的节点被选中参与修复的概率更大,避免盲目随机选择,提高修复的效率和准确性.

(3) 进一步分析学生程序和示例程序的节点序列中未匹配的节点,识别以下 3 类可能的变异操作节点位置,

© 中国科学院软件研究所 http://www.jos.org.cn

并给各个节点赋一个变异操作权值三元组$\langle WI,WR,WD\rangle$,分别表示该位置执行插入、替换和删除操作的概率.

- 插入语句位置:示例程序节点序列中对应位置有语句,而学生程序中该位置缺少与之相匹配的语句,此时,该位置的前一个语句节点是一个可能插入语句的位置,它的$\langle WI=0.5,WR=0.25,WD=0.25\rangle$.
- 替换语句位置:示例程序和学生程序此位置都有语句,但是内容不完全匹配,则该节点是一个可能替换的语句位置,它的$\langle WI=0.25,WR=0.5,WD=0.25\rangle$.
- 删除语句位置:学生程序中的语句节点在示例程序中找不到与之匹配的语句,则该节点是一个可能删除语句的位置,它的$\langle WI=0.25,WR=0.25,WD=0.5\rangle$.

对于不存在上述情况的节点,则赋以相同的变异操作权重,即$\langle WI=0.33,WR=0.33,WD=0.33\rangle$.

在执行变异操作时,根据概率选择变异位置和变异操作,既保留了遗传算法的随机性,也有助于指导补丁搜索朝着更有效的方向演化.

3.5 变异和交叉操作

为了节省变异体的存储空间,使用编辑序列代替变异体存储在种群中,编辑序列记录了从学生程序到生成变异体经历的操作步骤.由于本文方法的目标是从示例程序中转移正确逻辑和语句来修复学生程序中的缺陷语句,该转换操作是基于程序的抽象语法树执行的.根据学生程序和示例程序抽象语法树的匹配结果和需要的执行的转换操作,将变异操作分为 3 种,分别为插入(insert)、删除(delete)、替换(replace)操作,见表 2.

Table 2 Example of mutation operations

表 2 变异操作示例

变异操作	动作	学生程序 A	示例程序 B	含义
$I(A_1,B_2)$	Insert	A_1	B_2	在学生程序 A_1 节点后插入示例程序 B_2 节点
$D(A_2)$	Delete	A_2	–	删除学生程序中的 A_2 节点
$R(A_3,B_2)$	Replace	A_3	B_2	将学生程序 A_3 节点替换为示例程序 B_2 节点

在第 3.4 节的学生程序和示例程序的语法树差异分析过程中,识别了学生程序中可能插入、删除、或替换节点位置,并为这 3 种变异操作赋以不同的权重,权重高的变异操作具有较高的选择概率.这样有助于根据实际的缺陷位置和类型,选择合适的变异操作,提高变异转换的效率和有效性.

为了增加遗传的多样性,在变异之后对群体中的各个个体即编辑序列,按概率执行交叉操作.由于每个编辑操作都是一次针对特定的子树操作,因此各个编辑操作之间是相互独立的.本文方法的交叉操作沿用了 GenProg 所采用单点交叉操作.采用随机选择交叉点的策略,即随机产生一个交叉点位置,两个个体在交叉点位置互换部分编辑操作,形成两个子个体.如图 4 所示,分别生成两个小于或等于编辑序列操作个数的随机数,利用该随机数确定划分位置,将序列 1 和序列 2 分别划分为两部分,将序列 1 的前一部分与序列 2 的后一部分连接起来,序列 2 的前一部分与序列 1 的后一部分连接起来,形成两个新的编辑序列,替换原来的编辑序列,添加到种群中.

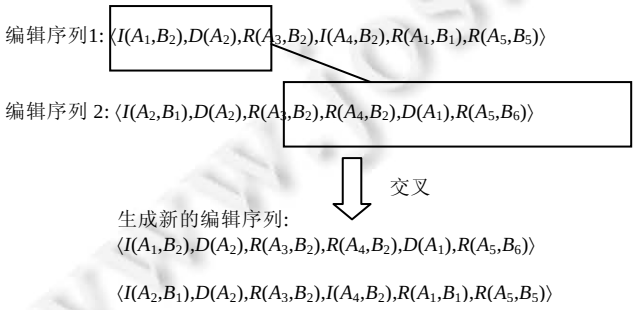

Fig.4 Example of crossover of editing sequences

图 4 编辑序列交叉示例

© 中国科学院软件研究所 http://www.jos.org.cn

3.6 适应度计算

变异体的适应度是遗传演化的依据,如公式(3)所示,根据变异体执行时通过测试用例数和未通过测试用例数以及变异体语法树与示例程序语法树的结构相似度 3 个元素衡量.

$$\left.\begin{aligned} fitness(P) = w_{posT} \times \frac{|t \in posT \text{ and } P \text{ passes } t|}{|posT|} + \\ w_{negT} \times \frac{|t \in negT \text{ and } P \text{ passes } t|}{|negT|} + \\ w_{similar} SimpleTreeMatching(P,S) \end{aligned}\right\} \tag{3}$$

其中,P 表示变异体,S 表示示例程序;$fitness(P)$表示变异体 P 的适应度;$posT$ 是成功测试用例集合,$|posT|$是其中测试用例的个数;$negT$ 表示失败测试用例集合,$|negT|$是其中测试用例的个数;t 表示一个测试用例,$|t \in posT \text{ and } P \text{ passes } t|$表示之前成功执行的测试用例执行变异体 P 时依然成功的个数,$|t \in negT \text{ and } P \text{ passes } t|$表示之前执行失败的测试用例执行变异体 P 时变为执行成功的个数;$SimpleTreeMatching(P,S)$表示变异体 P 和示例程序 S 的子树匹配相似度;w_{posT},w_{negT} 和 $w_{similar}$ 表示相应的权重值.

适应度计算步骤如下.

(1) 对于每一个编辑序列,在学生程序的语法树上执行编辑序列中的所有操作,重构语法树,再将语法树反向生成变异体;

(2) 使用测试用例执行种群中所有变异体,得到程序执行结果,统计成功通过测试用例的个数和失败测试用例的个数;

(3) 使用最长公共子序列算法计算变异体语法树与示例程序语法树的语法结构匹配相似度;

(4) 利用公式(3)计算该变异体的适应度;

(5) 通过变异体的适应度来筛选种群中的变异体,按照适应度决定变异体被随机选择的概率,以达到演化的目的.

3.7 示例演化驱动的学生程序自动修复算法

示例演化驱动的学生程序自动修复算法如算法 1 所示.输入含有缺陷的学生程序、示例程序、测试用例集以及遗传算法的相关参数,输出修复程序.

算法 1. 示例演化驱动的学生程序自动修复算法.

Input:缺陷学生程序 *P*;示例程序 *S*;测试用例集 *T*;种群规模 *popSize*;迭代最大次数 *pop*.

Output:适应度最高的修复程序 *Pr*.

1. 语法解析示例程序 *S*,创建模板程序节点列表 *nodeList*
2. 语法解析缺陷学生程序 *P*,创建源程序节点列表 *sourceNodeList*
3. *map*←*LCS*(*nodeList*,*sourceNodeList*); //最长公共子序列并判断结构差异
4. *group*←*Mutate*(*popSize*,*nodeList*,*sourceNodeList*); //生成一代变异序列
5. **repeat**
6. *fitnesses*←*SampleFit*(*group*,*T*); //计算每个变异体适应度
7. *parents*←*RouletteSelect*(*group*,*popSize*,*fitnesses*); //随机选择变异体
8. *group*←*Mutate*(*Parents*,*map*); //变异
9. *group*←*Crossover*(*group*); //交叉
10. **until** ∃*Pr*∈*group*,*FullFitness*(*Pr*,*T*)==*Passed* or 已循环 *pop* 次; //存在通过所有测试用例的变异体
11. **return** *Pr*;

- 第 1 行和第 2 行分别对示例程序和学生程序进行语法解析,用列表的形式存储语法树节点.
- 第 3 行将示例程序语法树的节点列表和源程序语法树的节点列表使用改进的最长公共子序列算法,判断两个程序的节点相似性,即两棵语法树的子树匹配.使用 *map* 存储程序节点与变异位置概率以及变

© 中国科学院软件研究所 http://www.jos.org.cn

异操作概率值的映射.

- 第 4 行生成第 1 代编辑序列,使用二维数组 *group* 存储 *popSize* 条编辑序列.
- 第 6 行将种群 *group* 中每一条编辑序列通过语法树重构和反向生成代码形成变异体并执行,按照将失败测试用例转变为成功测试用例的个数和成功通过成功测试用例的个数以及变异体与模板程序的语法结构相似程度计算适应度.
- 第 7 行使用轮盘赌算法按概率随机选择需要变异的变异体.
- 第 8 行对所选择的变异体按照位置概率和变异操作概率进行变异,生成新的编辑序列添加到种群 *group* 中.
- 第 9 行选择编辑序列和随机选择编辑序列的交叉点实现两个编辑序列的交叉,保证种群内变异体的多样性,并使得变异体的数量维持 *popSize* 个不变.
- 循环执行第 5 行~第 9 行,直到有变异体能够通过全部测试用例或迭代次数达到之前设置的迭代最大次数 *pop* 次.
- 第 11 行输出修复程序,如果没有能够通过全部测试用例的变异体,则输出适应度最高的变异体.

4 学生程序自动修复有效性实验分析

4.1 实验数据和环境

开发了示例演化驱动的 Java 学生程序自动修复系统,并使用赛码网上的真实习题进行测试.选择了 10 个真实在线编程题目,从真实学生提交的正确程序代码中选取使用不同语法结构的程序作为模板,并人工选择确认测试用例集合,使各个测试用例集合满足对所有模板程序的路径覆盖和条件判定覆盖.再从真实学生提交的含有缺陷的程序代码中选取不同错误版本,见表 3.

Table 3 Experimental data

表 3 实验数据

题目名称	模板数量	错误版本数	测试用例数	描述
回文串	10	10	15	判断能否添加字符成为回文串
研究生考试	10	10	11	4 科总分判断录取情况
上台阶	10	10	13	上台阶的走法
公交车乘客	10	10	10	上下车多次后公交车乘客数
分苹果	10	10	15	苹果分堆求原始个数
击鼓传花	10	10	20	花 *n* 次传回第 1 个人手中的方法数
字符判断	10	10	15	判断字符串 *b* 的所有字符是否都在字符串 *a* 中出现过
股神	10	10	10	股票涨跌 *n* 天后股值
刮刮卡兑换	10	10	15	刮刮卡最多兑换多少赠品
日期倒计时	10	10	15	计算出今天距离未来的某一天还剩多少天

回文串程序的分析示例见 http://homepage.hit.edu.cn/wangtiantian.

在选择测试程序的过程中,发现有部分错版本无法使用本文方法进行修复,包含以下几种情况.

(1) 函数调用参数个数不同.由于学生程序和模板程序中实现相同功能的函数可能含有不同个数的参数,而本文方法目前尚未考虑这种情况.

(2) 学生程序和模板程序的变量差异显著.学生程序的错误变量执行值序列和模板程序的变量执行值序列可能出现完全不相同的情况,此时,变量映射会出现偏差,而变量名映射不准确会导致学生程序无法正确执行.

(3) 多文件程序修复.当前只允许学生上传单文件程序代码,其中含有输入输出和 main 函数,可以直接编译运行,目前尚不支持多个文件之间的函数调用的分析.

(4) 学生程序因含有语法错误编译未通过.本文方法只针对可以编译运行的学生程序进行自动修复,可以分析无法通过任何测试用例的学生程序,但不支持分析含有语法错误的学生程序.

© 中国科学院软件研究所 http://www.jos.org.cn

最终,本文筛去上述类型的错误程序,选取了其中的 100 个错误版本作为测试数据进行实验,其中,所有学生程序的平均代码行数为 73 行.

本文从程序缺陷个数、缺陷类型和种群规模这 3 个方面分析程序自动修复的相关因素.实验环境硬件配置:2.3 GHz Intel Core i5 处理器,8GB 2133MHz LPDDR3 内存.

4.2 缺陷个数和修复结果关系

缺陷个数对修复结果的影响见表 4.本文方法可以修复含有多个缺陷的学生程序.当学生程序只有 1 到 2 个修复点时,修复率接近 100%;当含有 3 个修复点时,修复率为 70%;当含有 4 个及以上修复点时,修复率为 50%.种群规模与迭代次数增加,导致分析时间的增加.随着学生程序中的修复点数量增多,系统的修复率降低,且运行时间增加;学生程序中修复点数量越少,系统的修复率越高,运行时间越短.

Table 4 Relationship between the number of bugs and repair results

表 4 缺陷个数和修复结果间的关系

修复点个数	版本数	修复版本数	修复率(%)	平均种群规模	平均适应度(%)	运行时间(s)
1	30	30	100	50	93.37	17.863
2	40	39	97.5	70	79.42	159.722
3	20	14	70	100	62.58	409.216
4 及以上	10	5	50	100	30.29	738.927

多缺陷程序的修复率下降的主要原因是:尽管本文方法按照概率选择修复位置和操作,但是可疑语句位置、用于插入和替换的示例语句位置以及变异操作的选择都具有随机性,修复点越多,则组合生成的补丁搜索空间越大,导致在有限的种群规模和迭代次数内不能搜索到完全正确的补丁.

4.3 缺陷类型和修复结果关系

缺陷类型对修复结果的影响见表 5,其中,表达式、条件语句、循环语句缺陷在实际编程中较为常见,但是修复率与节点类型没有明显的关联.这是因为本文方法采用从示例程序中移植正确语句的方法,不限定所移植语句的类型,因此支持各种类型缺陷的修复.

Table 5 Relationship between bug types and repair results

表 5 缺陷类型和修复结果间的关系

节点类型	节点名称	缺陷出现次数	修复次数
VariableDeclarationStatement	初始化	18	15
ExpressionStatement	表达式	38	30
IfStatement.expression	If 条件表达式	21	18
WhileStatement.expression	While 循环条件表达式	14	10
ForStatement.expression	For 循环条件表达式	18	16
SwitchStatement	Switch 语句块	2	2
IfStatement.thenStatement	If 语句中的 Then 语句块	23	20
IfStatement.elseStatement	If 语句中的 Else 语句块	12	10
WhileStatement.body	While 循环语句块	16	13
ForStatement.body	For 循环语句块	18	14
ReturnStatement	返回值语句	15	12
Package	包的加载	15	13

4.4 种群规模的大小和迭代次数对修复的影响

本文选择 10 个含有 4 个及以上修复点的学生程序进行控制变量的实验,其他条件一样,测试种群规模的大小和迭代次数对程序自动修复的影响,结果见表 6 和表 7.

迭代次数保持不变时,种群的规模增加会使修复成功的数量增加,平均修复或未修复程序的最高适应度也随之增大,执行时间随之变长.当种群规模保持不变时,增加迭代次数可以使修复成功的数量增加,但是到了一定数量时,迭代次数的增加并不能带来明显的修复成功率变化,同时,运行时间也随之增大.

© 中国科学院软件研究所 http://www.jos.org.cn

Table 6 Relationship between population size and repair results

表 6 种群规模和修复结果间的关系

版本数量	迭代次数	种群规模	成功修复数量	平均适应度(%)	运行时间(s)
10	10	50	0	30.47	438.285
10	10	70	3	38.39	542.043
10	10	100	5	45.58	683.941
10	10	150	6	53.25	822.294
10	10	200	6	54.84	1 028.028

Table 7 Relationship between the number of iterations and repair results

表 7 迭代次数和修复结果间的关系

版本数量	迭代次数	种群规模	成功修复数量	平均适应度(%)	运行时间(s)
10	5	100	3	35.35	428.954
10	7	100	3	38.59	537.851
10	10	100	5	45.58	683.941
10	15	100	5	43.12	835.935
10	20	100	5	45.83	1 129.283

4.5 实验结果分析与讨论

本系统对于缺陷数量为 1 个~2 个的程序修复率较高,运行时间在 200s 之内.对于缺陷数量大于 2 的学生程序,建议多次执行或增大种群的规模.相比之下,增大种群规模更有助于提高修复率.因为种群规模变大,则每次迭代变异体的覆盖错误范围更大.

实验也证明了在本系统所针对分析的语句类型范围内,修复率与缺陷类型无关.但是对于本系统没有涉及分析的语句类型,则无法得到修复结果.在今后的工作中,可扩展分析的语句节点类型,使本系统可以修复更多的缺陷类型.

通过人工排查无法修复的程序,发现变量映射对程序修复至关重要.对于可以获得准确执行值序列匹配的变量,本文的值序列匹配方法可以准确对其进行映射;然而如果存在值序列差别较大,或存在学生程序中有而模板程序中没有的变量,则在执行程序变异时会导致变量名未正确匹配而使变异体无法通过编译.这也是本文需要进一步研究的关键内容.

本文面向学生程序自动化调试,重点研究了改进遗传编程的变异方法及适应度评价方法,并通过静态错误定位缩小遗传编程的搜索空间来提高修复有效性和效率.选择策略和交叉算子则沿用了 GenProg 中采用的方法.然而不同的选择、交叉策略对程序修复结果也可能产生影响.Kou 等人研究改进了 GenProg 的交叉策略[14],对所要交叉的个体对,根据差异度或测试用例通过率进行降序排序,选择排序靠前的个体对执行交叉操作;并且对比分析了单点交叉、均匀交叉和随机交叉策略与其选择策略配合使用时的程序修复效果.他们的实验结果表明,采用其所提出的测试用例通过率优先的选择操作,并且应用随机交叉策略时,程序修复效果优于 GenProg 采用的单点交叉策略[14].类似地,本文的后续工作将进一步实验分析不同选择算子和交叉算子等对学生程序修复结果的影响.

5 结 论

本文针对学生程序研究自动修复方法.提出了示例演化驱动的程序修复方法,利用模板程序指导补丁的搜索和生成,以修复学生程序中多个复杂缺陷.提出了基于示例的静态错误定位和差异分析方法,定位可疑语句并识别可能的变异操作,避免大量的盲目变异操作,缩小搜索空间.提出了基于值序列的变量映射方法,映射缺陷程序和示例程序中实现相同功能的变量.结合测试用例以及学生程序与示例程序的语法结构相似度来评价补丁的适应度,使得适应度的值更加准确,不会出现大量变异体适应度均为 0 的现象,有效指导补丁程序朝着期望演化.实现了一个针对 Java 语言学生程序自动修复的 Web 应用程序,为学生编程提供自动反馈,实验结果验证了该系统的有效性.后续将完善该方法研究,并进一步增强其实用性.

© 中国科学院软件研究所 http://www.jos.org.cn

References:

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

[2] Wang KC, Wang TT, Su XH, Ma PJ. Key scientific issues and state-art of automatic software fault localization. Chinese Journal of Computers, 2015,38(11):2262−2278 (in Chinese with English abstract).

[3] Zhong H, Su Z. An empirical study on real bug fixes. In: Proc. of the IEEE/ACM Int'l Conf. on Software Engineering. Firenze: IEEE, 2015. 913−923.

[4] 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]

[5] Gazzola L, Micucci D, Mariani L. Automatic software repair: A survey. IEEE Trans. on Software Engineering, 2019,45(1):34−67.

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

[7] Singh R, Gulwani S, Solar-Lezama A. Automated feedback generation for introductory programming assignments. In: Proc. of the ACM SIGPLAN Symp. on Principles of Programming Languages. Seattle: ACM Press, 2013. 15−26.

[8] Kim D, Kwon Y, Liu P, Kim IL, Perry DM, Zhang X. Apex: Automatic programming assignment error explanation. In: Proc. of the 2016 ACM SIGPLAN Int'l Conf. on Object-oriented Programming, Systems, Languages, and Applications. Amsterdam: ACM Press, 2016. 311−327

[9] Yi J, Ahmed UZ, Karkare A, Tan SH, Roychoudhury A. A feasibility study of using automated program repair for introductory programming assignments. In: Proc. of the 2017 11th Joint Meeting on Foundations of Software Engineering. Paderborn: IEEE, 2017. 740−751.

[10] Weimer W, Fry ZP, Forrest S. Leveraging program equivalence for adaptive program repair: Models and first results. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. Palo Alto: IEEE, 2013. 356−366.

[11] Long F, Rinard M. Automatic patch generation by learning correct code. In: Proc. of the ACM SIGPLAN Symp. on Principles of Programming Languages. Santa Barbara: ACM Press, 2016. 298−312.

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

[13] Wang T, Su X, Wang Y, *et al*. Semantic similarity-based grading of student programs. Information and Software Technology, 2007, 49(2):99−107.

[14] Kou, R, Higo Y, Kusumoto S. A capable crossover technique on automatic program repair. In: Proc. of the Int'l Workshop on Empirical Software Engineering in Practice. YamadaokaSuita: IEEE, 2016. 45−50.

附中文参考文献:

[2] 王克朝,王甜甜,苏小红,马培军.软件错误自动定位关键科学问题及研究进展.计算机学报,2015,38(11):2262−2278.

[4] 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]

王甜甜(1980－),女,辽宁丹东人,博士,副教授,CCF 专业会员,主要研究领域为软件自动化调试,计算机辅助教学.

王克朝(1980－),男,博士,副教授,CCF 专业会员,主要研究领域为软件工程,软件故障定位.

许家欢(1993－),女,硕士,主要研究领域为软件自动修复.

苏小红(1966－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为智能软件技术,无人机航迹规划,目标检测与跟踪.

© 中国科学院软件研究所 http://www.jos.org.cn