第44卷 第10期 计 算 机 工 程 2018年10月
Vol. 44 No. 10 Computer Engineering October 2018

·体系结构与软件技术· 文章编号: 1000-3428(2018)10-0095-06 文献标志码: A 中图分类号: TP391

一种基于StackOverflow分析的程序自动修复方法

刘旭亮,钟 浩

(上海交通大学 电子信息与电气工程学院,上海 200240)

摘 要: 现有程序缺陷修复方法缺乏充足的修复模板,多数只能修复小部分程序缺陷。为此,提出一种从StackOverflow中获取示例代码,并从示例代码中挖掘出修复样品的方法。从程序员日常开发讨论中获取与修复有关的代码对,并从中生成修复模板,该模板丰富了自动修复工具已有的操作,能够修复更多缺陷。对Defects4J数据集进行实验验证,结果表明,该方法能够修复23个程序缺陷,优于GenProg和Nopol等工具。

关键词: 程序自动修复;真实缺陷修复;修复模板;示例代码;修复样品

中文引用格式: 刘旭亮,钟 浩. 一种基于StackOverflow分析的程序自动修复方法[J]. 计算机工程,2018,44(10):95-100.

英文引用格式: LIU Xuliang,ZHONG Hao. A program automatic repair method based on StackOverflow analysis[J]. Computer Engineering,2018,44(10):95-100.

A Program Automatic Repair Method Based on StackOverflow Analysis

LIU Xuliang,ZHONG Hao

(School of Electronic Information and Electrical Engineering,Shanghai Jiaotong University,Shanghai 200240,China)

【Abstract】 Existing methods for program defect repair lack sufficient repair templates,and most of them can only repair a small number of program defects. To solve this problem,a method is proposed to obtain sample code from StackOverflow and to mine repair samples from sample code. From the programmer's day-to-day development discussions,get the code pairs that are relevant to the fix and generate the fix template,which enriches the existing operations of the automatic fix tool and enables it to fix more defects. Experimental results on Defects4J dataset show that the method can repair 23 program defects,and is superior to GenProg and Nopol.

【Key words】 program automatic repair; real defect repair; repair template; sample code; repair sample

DOI: 10.19678/j.issn.1000-3428.0048446

0 概述

近年来,随着对程序缺陷的深入探究[1-2],缺陷自动修复在软件工程领域已经成为研究热门。当为给定的缺陷程序生成修复补丁时,一个典型的方法就是应用导向算法优化修复动作,并不停地尝试生成新的补丁,直到生成的补丁能通过所有测试用例。尽管这项研究非常有前景,并且对学术界和工业界都具有吸引力,但是,最近的经验学习[3-4]表明,现有程序自动修复方法的修复能力存在很大的局限性。

文献[5]对真实的程序缺陷修复进行经验学习研究。在对比人工修复动作和现有自动修复工具的修复动作后,他们发现现有方法只能修复特定范围内的程序缺陷,原因是现有修复方法只支持限定的修复动作。例如,众所周知的GenProg工具[6]只支持3种类型的修复动作:插入语句,交换语句和删除语句。但是,修复一个程序缺陷经常需要更细的修改。为解决该问题,研究者们探索如何丰富修复动作。文献[7]手工分析数千人写的缺陷补丁,并从中总结出10个修复模板。然而,其修复模板在数量和粒度上还是存在局限性。

文献[8]研究表明重复使用StackOverflow讨论里的示例可以修复程序缺陷,其根据缺陷程序的崩溃信息检索StackOverflow里的相关贴子,并用这些贴子信息修复缺陷。但是,该方法非常依赖于实时检索和检索时所需的程序崩溃信息,事实上多数缺陷程序并不会崩溃,或者缺少崩溃信息。

针对上述方法存在的问题,本文从大量StackOverflow的历史贴子里挖掘有效的修复模板,用这些修复模板来修复新的程序缺陷。

基金项目: 国家重点基础研究计划(2015CB352203);国家自然科学基金面上项目(61572313)。

作者简介: 刘旭亮(1994—),男,硕士研究生,主研方向为软件工程;钟 浩,副研究员。

收稿日期: 2017-08-23 **修回日期:** 2017-11-12 **E-mail:** deongaree@sjtu.edu.cn

1 相关工作

1.1 程序自动修复

给定一个缺陷程序,程序自动修复工具会搜索一个源代码级别的补丁,该补丁能让程序在避免缺陷的同时保存关键功能[6]。近年来,很多用于搜索补丁的方法[7,9-13]被提出。文献[5]包含变异和交换算子,其随机地产生并演化补丁。文献[10]使用随机搜索。文献[14]较早产生最流行的补丁,然后用过去的修复训练出一个机器学习模型,并用得到的模型预测每个新产生的补丁的正确性。文献[15-16]在赋值语句和条件语句中修复缺陷。但仍然有研究者对上述方法的正确性持有争议。例如,文献[10]表示 RSRepair 比 GenProg 更有效,然而文献[17]观察后得出相反的结论。文献[18]证明 GenProg 能在105 个程序缺陷中修复 55 个,但文献[5]表示在修复的55 个中只有2 个是正确的。一些研究者批评指出现有方法对于修复真实程序缺陷而言缺少足够的修复模板。文献[7]从数千个缺陷修复中获取了 10 个修复模板。

本文从 StackOverflow 的千万数量级的贴子中挖掘修复模板并将其应用于缺陷修复。对一个运行崩溃的程序缺陷,文献[8]根据崩溃信息从 StackOverflow 中查询代码示例,并使用代码示例里的匹配信息来修复新的程序缺陷。本文发现该精确匹配示例信息对于修复许多程序缺陷而言不可行,并通过实验验证本文细粒度模板的修复性能。

1.2 在线论坛分析

在线论坛(如 StackOverflow)为数据分析提供了丰富的资源。文献[19]提出一个基于分级方法的序列模式,用来探查论坛主题的问答对。文献[20]用一个自适应的基于特征的矩阵因式分解框架来推荐相关贴子。文献[21]构建一个模型,基于推导网络来获取在线主题,该推导网络利用了论坛主题的结构特性。文献[22]采用投票技术,这种投票技术被应用于排名聚集任务,例如博客升华和专家探索。本文从另一个角度分析论坛帖子并从中挖掘修复模板。

2 方法设计与实现

如图 1 所示,本文方法共分 3 个阶段: 挖掘,筛选和修复。

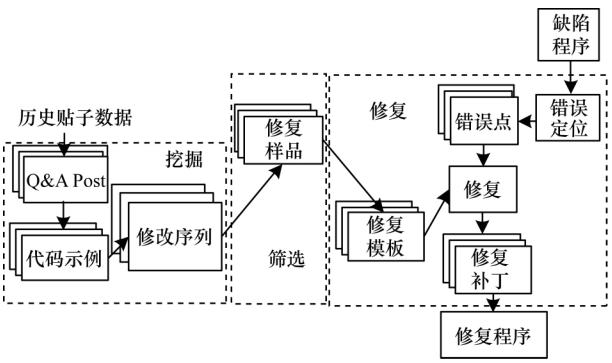

图 1 基于 StackOverflow 的程序修复方法框架

2.1 StackOverflow 挖掘

在图 1 中,首先从 StackOverflow 中挖掘相关联的修改序列(Linked Modification Sequence,LMS),具体步骤如下。

2.1.1 问答贴匹配

从 StackOverflow 的历史数据中重建问答贴(Q&A Post)的联系。在下载的 StackOverflow 数据中,贴子是以 XML 格式存储的。例如,图 2 所示的 XML 片段展示了 3 个贴子。每一个问题贴可能记录一个接受的答案贴编号,可以此为联系将问题贴和其接受的答案贴匹配成对。原始数据是以创建日期排序的,因此,在正常情况下,紧跟着一个问题贴的下一个贴子可能与其毫无关系。

```
// question Post
<row Id="4"
PostTypeId="1"
AcceptedAnswerId="7"
CreationDate="2008-07-31T21:42:52.667"
Body="<p>I want to…"
Tags="<c#><winforms<type-conversion><decimal><opacity>"
AnswerCount="13"
…
/>
// accepted answer Post
<row Id="7"
PostTypeId="2"
ParentId="4"
CreationDate="2008-07-31T22:17:57.883"
Body="<p>Anexplicit cast…"
…
/>
```

图 2 XML 片段示例

图 3 所示为重建问答贴之间联系的算法。

```
Input: QLIST
Output: QALIST
1. Init for QLIST
2. for each p ∈ PLIST do
3. if  p.type == 1 then
4. q←p
5. if  q.tags.contain(“Java”) then
6. QLIST.add(q)
7. end if
8. else
9. a←p
10. search for q ∈ QLIST,
11. such that
12. q.AcId == a.Id && p.PId == a.Id
13. do
14. QALIST.add(pair<q,a>)
15. QLIST.remove(q)
16. end if
17. end for
```

图 3 获取和筛选问答贴算法

由图 3 可以看出,第 1 行初始化一个空的列表(QLIST),用于存储在遍历所有贴子过程中还未被匹配到的问题贴;第 2 行~第 17 行遍历所有的贴子,将遇到的所有问题贴首先加入 QLIST 列表中,这里只挑选标签(Tags)里有 Java 关键字的问题贴;对于每一个 QLIST 列表中的贴子,第 9 行~第 14 行搜索其对应的接受的答案贴,并将问答帖一起加入到 QALIST 列表中;第 15 行移除已经匹配到的问题贴,在这里,一些问题贴可能找不到相应的被接受的答案贴,其原因可能是该问题贴根本没有被接受的答案或者答案贴不包含在本文数据集中。

2.1.2　代码对获取

从每一对问答贴中获取代码对,特别地,对于每一个贴子,需从 < code > 和 < pre > 标签共同包含的数据里提取代码示例。一个贴子偶尔也可能会包含多个代码示例。例如,一个回答者可能根据自己的理解重写提问者给出的缺陷代码。文献[8]倾向于通过匹配关键词来构建示例代码对,关键词有"instead of"和"change…to…"。但是,这个启发式的依据并不可靠。因为缺陷代码和修复代码经常是相似的,所以本文通过文本相似度来决定代码对的匹配。本文方法相似函数为:

$$Sim(Q,A)=\frac{SimTokenNum(Q,A)}{c\cdot Len(Q)+(1-c)\cdot Len(A)}$$

其中,Q 表示问题,A 表示答案,c 是一个相关系数,本文设置 c 的值为 0.1。

2.1.3　修改序列挖掘

对于每一对示例代码,本文应用 GumTree[23] 对比它们的不同点。GumTree 建立在 Spoon[24] 上,Spoon 是一个代码分析工具,其用自定义的元模型格式来构建抽象语法树。Spoon 的元模型包含 3 个部分:1)结构部分包含程序元素的声明(如接口、类、变量和方法);2)代码部分包含可执行的 Java 代码(如方法体);3)引用部分包含对程序元素的引用(如一个类型的引用)。基于该模型,Spoon 能够识别出3 种类型的代码元素。在给定的抽象语法树对(a_1和 a_2)上,GumTree 生成一个修改序列,将 a_1 转化为 a_2。本文对相关概念进行如下定义:

定义 1(元素)　一个元素是抽象语法树的一个节点或者一棵子树。对于一个元素,其拥有类型和值。如果该元素是一棵子树,定义其类型为该子树根节点的类型。

定义 2(动作)　一个动作是对一个元素的原子操作。原子操作包括插入、更新、移动和删除。

定义 3(修改)　一个修改表示为用一个动作将一个原始元素转变为新的元素。

StackOverflow 中的示例代码经常是碎片,它们可能缺乏 import 语句,也可能没有一个完整的编译单元,这使得 Spoon 可能无法构建代码的抽象语法树。为解决该问题,本文在示例代码头部添加 import 语句,并导入一些常用的包,例如"java. util. * "和"java. io. * "。另外,也为示例代码添加类型声明和方法声明语句。对于每一对代码抽象语法树,使用 GumTree 来获取其修改序列。

图 4 所示为一对示例代码对,通过对这 2 个示例代码的比较得到一个修改序列为:1)移动"z = 1";2)插入"++";3)更新">"为"<";4)插入"x";5)删除"y = 1"。为了后续分析,需要将得到的完整修改序列划分为多个有联系的修改序列段。

```
//缺陷代码
z = 1;
if(x>0)
x = abs(x);
y = 1;
-----------------------------------------------
//修复代码
if(x<0)
x = abs(x);
z = 1;
x++;
```

图 4　示例代码对

定义 4(有联系的修改)　如果 2 个修改的目标元素之间存在父子关系,则它们是相互联系的。特别地,如果 2 个修改的目标元素是单个节点,并且为兄弟关系,则它们也是相互联系的。以上述得到的修改序列为例,修改 1)和 5)的目标元素是一颗子树,其余都是作用于节点。根据定义 4,最终会得到 4 个有联系的修改序列:1),2),4),3),5)。

2.2　修复样品筛选

从有联系的修改序列中挖掘出修复样品,修复样品通常满足以下 2 个标准:

1)一个修复样品只存在一个待确定的值。

2)一个修复样品是被经常使用的。

目前,程序缺陷自动修复都是基于对测试用例的验证。如果一个缺陷的正确修复需要 2 个值,则修复方法就需要穷举所有合适的 2 个值的组合。这样的穷举在本文方法里不可行。因此,第 1 条标准要求修复只需搜索到一个正确的值;第 2 条标准保证了一个样品的质量。本文认为一个经常使用的修复样品更可能成功修复一个新的程序缺陷。

为满足上述 2 个标准,首先根据修改的元素的粒度(节点或者子树)将所有的有联系的修改序列分

为 2 个子序列。一个子序列只包含对子树的修改，另一个只包含对节点的修改。对于一个有联系的修改序列，如果它只包含对节点的修改，将运用以下方法对其进行筛选:

1) FilterCommonType 方法针对修改序列中的插入和更新。其只选出包含的插入和更新的元素类型为变量、方法调用、二元操作符或者变量类型的修改序列。为确认这些常用插入和更新元素的类型，将对所有数据集中所有插入和更新的节点元素类型进行排序，选出其中排在前面的节点元素。

2) FilterAction 方法舍弃一些不适用的修改序列。如果一个序列只包含删除和移动操作，会将之舍弃，因为这样的序列不太可能成为一个精确有效的修复样品。

3) FilterActionNumber 方法筛选出不超过 5 个插入或更新操作的修改序列。一个插入或者更新操作可能需要一个用作修复的值，该筛选方法的目的是限制所需修复的值的数量。

在筛选完成后，合并所有的同构修改序列。同构的定义如下:

定义 5(同构修改) 2 个同构的修改需要满足 2 个条件:

1) 修改的动作必须相同。

2) 如果动作是插入或者更新，则插入或者更新前的元素类型必须相同; 如果动作是移动或删除，则移动或者删除的元素的父节点元素类型必须相同。

定义 6(同构修改序列) 2 个同构的修改序列需要满足 2 个条件:

1) 修改序列的修改个数必须相等。

2) 序列中相同位置对应的每个修改必须是同构的。

最后，本文方法在整个数据集中对同构的修改序列进行计数。如果实例总数超过 5，则保存此同构的修改序列为一个修复样品。

2.3 改进的程序缺陷修复

在获取修复样品后，对样品进行进一步总结，推导出修复模板。检查每一个修复样品，一个修复样品拥有超过 5 个实例，对于这些实例，检查它们在抽象语法树上的共同结构以及它们修改元素的值，对其总结推导后得到修复模板。

文献 [25] 定义 3 种类型的缺陷程序:

1) L1: 程序在单条语句上有一个缺陷。

2) L2: 程序在多条语句上有一个缺陷。

3) L3: 程序有多个缺陷。

本文关注 L1 缺陷程序。对于每一个程序缺陷，用文献 [26] 定位缺陷的位置。对于缺陷程序中所有的语句，如果其怀疑值超过预定义的阈值，则尝试在这些语句上使用修复模板去修复。一个修复模板会被应用于含有特定类型元素的怀疑语句上。根据特定元素匹配，如果怀疑语句适用于一个或者多个修复模板，则尝试使用每一个修复模板，同时搜索获取修复所需的值并生成修复补丁。本文方法从2 个搜索阈中获取修复所需的值，一个是对应的修复样品中包含的修改元素的值，另一个是缺陷程序本身。如果一个新的修复补丁生成，本文将通过测试用例验证其正确性。

3 实验验证

3.1 数据集

如表 1 所示，选取在缺陷修复研究领域被广泛使用的文献 [27] 缺陷程序集合作为测试目标。但本次实验没有使用其中 Closure Compiler 项目中的程序缺陷，因为其缺少测试用例，即无法定位缺陷。

表 1 Defects4J 缺陷程序集合参数

项目	缺陷数	测试用例个数	简称
JFreeChart	26	2 205	Chart
Apache Commons Lang	65	2 245	Lang
Apache Commons Math	105	3 602	Math
Joda-Time	27	4 130	Time
总计	223	12 182	—

本文从 StackOverflow 网站下载 31 017 891 个贴子数据作为实验输入。这些贴子的时间跨度从 2008 年 7 月—2016 年 6 月。从这些贴子中获取到 277 020 对 Java 示例代码。

3.2 实验环境与参数设置

本文修复工具以 Astor [28] 为基础，用所有获取到的修复模板来进行程序缺陷修复，使用 GZoltar 定位缺陷。设置 GZoltar 中缺陷语句怀疑阈值为 0.01。对于每一条怀疑的缺陷语句，本文修复工具搜索其中的元素，并找到适用的修复模板。设定对于一个元素的修改而生成的补丁不超过 10 个。仅当生成的补丁跟人工补丁相同或者是语义上一致时，才认为缺陷被修复。本文实验运行于一个 Ubuntu 服务器上，服务器的使用配置为 2.00 GHz Intel Xeon E5-2620 CPU，16 GB 内存。设置每个程序缺陷的最大修复时间为 3 h。

3.3 实验结果与分析

表 2 所示为各修复工具对 Defects4J 程序缺陷集的修复结果。其中，修复成功的缺陷指生成的修复补丁跟人工补丁完全相同或者语义上相同。

表 2　不同修复工具修复结果

项目	本文方法	GenProg	PAR	Nopol	HistoricalFix	ACS
Chart	5	—	—	1	2	2
Lang	4	—	1	3	7	3
Math	13	5	2	1	6	12
Time	1	—	—	—	1	1
总计	23	5	3	5	16	18

本文方法共成功修复 23 个程序缺陷。文献[4]用 GenProg 和 Nopol 修复 Defects4J 里所有的缺陷,并且设置每个缺陷的最大修复时间为 3 h。结果显示,GenProg 和 Nopol 均只修复了 5 个缺陷。文献[9]用 PAR 和 HistoricalFix 分别修复从 Defects4J 里挑选出的 90 个程序缺陷,其设置每个程序缺陷的最大修复时间为 90 min。虽然该实验未尝试修复 Defects4J 里所有的缺陷,但是他们承认未选择的程序缺陷理论上是不能被他们的工具所成功修复的。实验结果显示,PAR 只修复了 3 个缺陷,尽管 HistoricalFix 修复了较多的缺陷,但是其假定缺陷位置已知,这会大幅降低修复缺陷的难度并缩短修复所需要的时间。文献[10]用 ACS 修复 Defects4J 里所有的缺陷,并设置每个缺陷的最大修复时间为30 min。实验结果表明,ACS 成功修复了较多的缺陷。但是,ACS 限定只修复条件语句出错的缺陷。ACS 从修复开始阶段就放弃修复非条件语句出错的缺陷,这能够大幅降低修复时间。理论上,给予 ACS 更多的修复时间,其也无法修复出 Defects4J 里更多的缺陷。

综上,本文方法使用缺陷定位工具搜寻缺陷可能的位置,不局限于修复特定位置的缺陷,且从实验结果来看,本文方法能够修复最多的缺陷程序。

4　结束语

本文通过分析 StackOverflow 的数据挖掘获取到细粒度的修复样品,基于修复样品中的共同结构进一步推导出修复模板。对获取到的修复模板在 Defects4J 程序缺陷集合上进行实验评估,结果表明,本文方法共修复 23 个缺陷,多于 GenProg 和 Nopol 等工具。如第 2.3 节所述,本文方法针对的缺陷主要为 L1 类别,而要成功修复 L2 和 L3 类别的缺陷将异常困难。因此,下一步考虑用更有效的算法将多个修复模板组合起来,以修复更复杂的缺陷程序。

参考文献

[1] 刘　杰,王嘉捷,欧阳永基,等. 基于污点指针的二进制代码缺陷检测[J]. 计算机工程,2012,38(24):46-49.

[2] 李志敏,殷蓓蓓,张　萍,等. 一种实时性缺陷定位方法及其可视化实现[J]. 计算机工程,2017,43(2):111-119.

[3] QI Z,LONG F,ACHOUR S,et al. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems[C]//Proceedings of 2015 International Symposium on Software Testing and Analysis. New York, USA: ACM Press,2015: 24-36.

[4] MARTINEZ M,DURIEUX T,SOMMERARD R,et al. Automatic repair of real bugs in Java: a large-scale experiment on the Defects4J dataset[J]. Empirical Software Engineering,2017,22(4):1-29.

[5] ZHONG H,SU Z. An empirical study on real bug fixes[C]//Proceedings of IEEE International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2015: 913-923.

[6] WEIMER W,NGUYEN T V,GOUES C L,et al. Automatically finding patches using genetic programming[C]//Proceedings of IEEE International Conference on Software Engineering. Washington D.C.,USA: IEEE Press,2009: 364-374.

[7] KIM D,NAM J,SONG J,et al. Automatic patch generation learned from human-written patches[C]//Proceedings of IEEE International Conference on Software Engineering. Washington D. C.,USA: IEEE Press,2013: 802-811.

[8] GAO Q,ZHANG H,WANG J,et al. Fixing recurring crash bugs via analyzing Q&A sites[C]//Proceedings of 2015 IEEE/ACM International Conference on Automated Software Engineering. Washington D. C., USA: IEEE Press, 2015: 25-31.

[9] WEIMER W,FRY Z P,FORREST S. Leveraging program equivalence for adaptive program repair: models and first results[C]//Proceedings of IEEE/ACM International Conference on Automated Software Engineering. Washington D. C., USA: IEEE Press,2014: 356-366.

[10] QI Y,MAO X,LEI Y,et al. The strength of random search on automated program repair[C]//Proceedings of the 36th International Conference on Software Engineering. New York,USA: ACM Press,2014: 254-265.

[11] MECHTAEV S,YI J,ROYCHOUDHURY A. DirectFix: looking for simple program repairs[C]//Proceedings of IEEE/ACM International Conference on Software Engineering. Washington D. C.,USA: IEEE Press,2015: 448-458.

[12] KE Y,STOLEE K T,GOUES C L,et al. Repairing programs with semantic code search[C]//Proceedings of IEEE/ACM International Conference on Software Engineering. Washington D.C.,USA: IEEE Press,2016: 295-306.

[13] FLONG F,RINARD M. Staged program repair with condition synthesis[C]//Proceedings of Joint Meeting on Foundations of Software Engineering. New York,USA: ACM Press,2015: 166-178.

[14] LONG F,RINARD M. Automatic patch generation by learning correct code[C]//Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York,USA: ACM Press, 2016: 298-312.

[15] MARTINEZ M,MONPERRUS M. Mining repair actions for automated program fixing[EB/OL]. [2017-07-20]. https://hal.inria.fr/hal-00696590/PDF/posterRennes12_v13.pdf.

[16] NGUYEN H D T,QI D,ROYCHOUDHURY A,et al. SemFix: program repair via semantic analysis[C]//Proceedings of International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2013: 772-781.

[17] SMITH E K, BARR E T, GOUES C L, et al. Is the cure worse than the disease? overfitting in automated program repair [C]//Proceedings of Joint Meeting on Foundations of Software Engineering. New York, USA: ACM Press, 2015: 532-543.

[18] GOUES C L, DEWEYVOGT M, FORREST S, et al. A systematic study of automated program repair: fixing 55 out of 105 bugs for $8 each [C]//Proceedings of International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2012: 3-13.

[19] CONG G, WANG L, LIN C, et al. Finding question-answer pairs from online forums [C]//Proceedings of SIGIR'08. Washington D. C., USA: IEEE Press, 2008: 467-474.

[20] YANG D, PIERGALLINI M, HOWLEY I, et al. Forum thread recommendation for massive open online courses [EB/OL]. [2017-07-20]. http://pdfs.semanticscholar.org/4667/69121ebb6e1aec6ac8f13ae038c99247739f.pdf.

[21] BHATIA S, MITRA P. Adopting inference networks for online thread retrieval [C]//Proceedings of the 24th AAAI Conference on Artificial Intelligence. [S. l.]: AAAI Press, 2010: 65-71.

[22] ALBAHAM A T, SALIM N. Adapting voting techniques for online forum thread retrieval [M]//HASSANIEN A E, SALEM A B M, RAMADAN R. Advanced machine learning technologies and applications. Berlin, Germany: Springer, 2012: 439-448.

[23] BLANC X, MARTINEZ M, MONPERRUS M. Fine-grained and accurate source code differencing [C]//Proceedings of ACM/IEEE International Conference on Automated Software Engineering. New York, USA: ACM Press, 2014: 313-324.

[24] PAWLAK R, MONPERRUS M, PETITPREZ N, et al. SPOON: a library for implementing analyses and transformations of Java source code [J]. Software Practice and Experience, 2016, 46(9): 1155-1179.

[25] DEBROY V, WONG W E. Using mutation to automatically suggest fixes for faulty programs [C]//Proceedings of the 3rd International Conference on Software Testing, Verification and Validation. Washington D. C., USA: IEEE Press, 2010: 65-74.

[26] JOSÉ C, PEREZ A, RUI A. GZoltar: an eclipse plug-in for testing and debugging [C]//Proceedings of IEEE/ACM International Conference on Automated Software Engineering. Washington D. C., USA: IEEE Press, 2013: 378-381.

[27] JALALI D, ERNST M D. Defects4J: a database of existing faults to enable controlled testing studies for Java programs [C]//Proceedings of International Symposium on Software Testing and Analysis. New York, USA: ACM Press, 2014: 437-440.

[28] MARTINEZ M, MONPERRUS M. ASTOR: a program repair library for Java (demo) [C]//Proceedings of International Symposium on Software Testing and Analysis. New York, USA: ACM Press, 2016: 441-444.

[29] LE X B D, LO D, GOUES C L. History driven program repair [C]//Proceedings of IEEE International Conference on Software Analysis, Evolution, and Reengineering. Washington D. C., USA: IEEE Press, 2016: 213-224.

[30] XIONG Y, WANG J, YAN R, et al. Precise condition synthesis for program repair [C]//Proceedings of IEEE/ACM International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2017: 416-426.

编辑 吴云芳

(上接第 94 页)

[11] 王恩东, 倪 璠, 陈继承, 等. 一种面向实时系统的程序基本块指令预取技术 [J]. 软件学报, 2016, 27(9): 2426-2442.

[12] GRAN R, SEGARRA J, RODRÍGUEZ C, et al. Optimizing a combined WCET-WCEC problem in instruction fetching for real-time systems [J]. Journal of Systems Architecture, 2013, 59(9): 667-678.

[13] GARSIDE J, AUDSLEYN C. WCET preserving hardware prefetch for many-core real-time systems [C]//Proceedings of the 22nd International Conference on Real-time Networks and Systems. New York, USA: ACM Press, 2014: 193-202.

[14] YUChenjie, PETROV P. Off-chip memory bandwidth minimization through cache partitioning for multi-core platforms [C]//Proceedings of the 47th Design Automation Conference. Washington D. C., USA: IEEE Press, 2000: 132-137.

[15] BURGER D, AUSTINT M. The simplescalar tool set, version 2.0 [J]. ACM SIGARCH Computer Architecture News, 1997, 25(3): 13-25.

[16] Mälardalen Real-time Research Center. WCET benchmarks [EB/OL]. [2017-08-31]. http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.

编辑 顾逸斐