*论文题目：Adaptively generating high quality fixes for atomicity violations

*作者：蔡彦、曹玲微、赵靖

*单位：中国科学院软件研究所、哈尔滨工程大学

联系方式：

*文章发表信息：Proceedings of the 11th Joint Meeting of the European Software Engineering Conference and the ACM SigSoft Symposium on The Foundations of Software Engineering (ESEC/FSE)，ACM，2017 年 9 月，303-314 页。

*原文链接地址：https://doi.org/10.1145/3106237.3106239

*正文：

1、 背景和动机

并发缺陷普遍存在于并发程序中，而由于并发执行的不确定性，对于并发缺陷的检测，重现以及正确修复都是非常困难的。手工修复程序缺陷不仅耗费人力，并且极易出错，因而并发缺陷的自动化修复研究越来越受到研究人员的关注。本文主要研究的是对于原子缺陷的高质量自动化修复。

某线程对某个共享对象的连续迁移序列具备原子性，意味着它不允许别的线程在该序列中间对该对象进行读或写操作。违反了原子性而导致程序出错，就属于原子性错误。

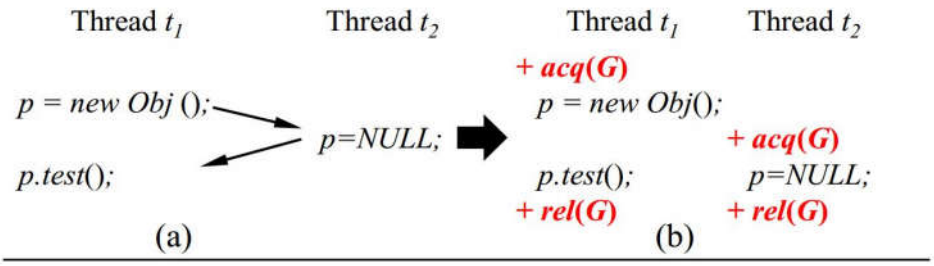

图 1.原子性缺陷示例

如图 1(a)是原子性错误的一个简单例子，如图 1（b）所示，对于原子性错误的主要修复思想是通过加锁来防止别的线程在相应序列中间对该对象进行读和写操作，例如 *AFix*, *Axis*, *Grail*, *Gadara*。我们称这些方法为 *Gate Lock Algorithm*，简称 *GLA*。*HFix* 也是尝试用调整锁的方式来修复原子性缺陷。通过添加新锁的方式进行原子性修复，极易引入新的死锁问题，同时也可能会引入严重的性能缺陷。

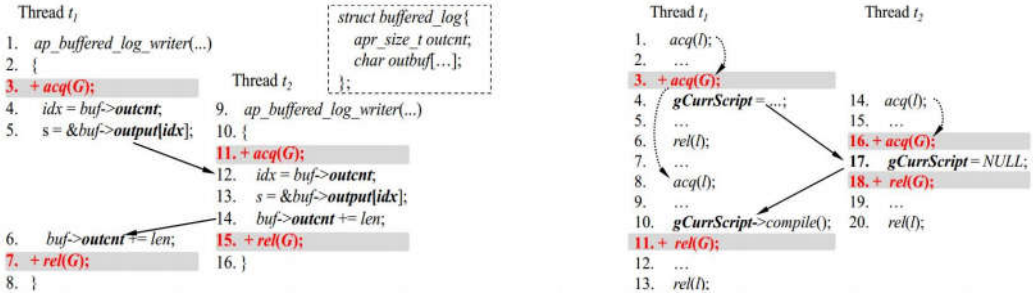

图 2. 程序中的原子性缺陷示例及 GLA 修复

如图 2（b）是 Mozilla 源码中的一个原子性缺陷实例。该原子性缺陷发生在全局变量 gCurrScript 上，GLA 修复方法在引入新锁的过程中，引入了新的锁序 l->G 和 G->l,从而引入了一个 l 和 G 上的死锁问题。

如图 2（a）是 Apache 源码中的一个原子性缺陷实例。该原子性缺陷发生在两个结构体变量上：buf->output 和 buf->outcnt。只有当两个线程的 buf 对象指向同一个地址时，原子性缺陷才会真正发生，而 GLA 通过添加全局保护锁的方式，会串行化所有线程的执行，造成过度序列化，引起较大的性能缺陷问题。

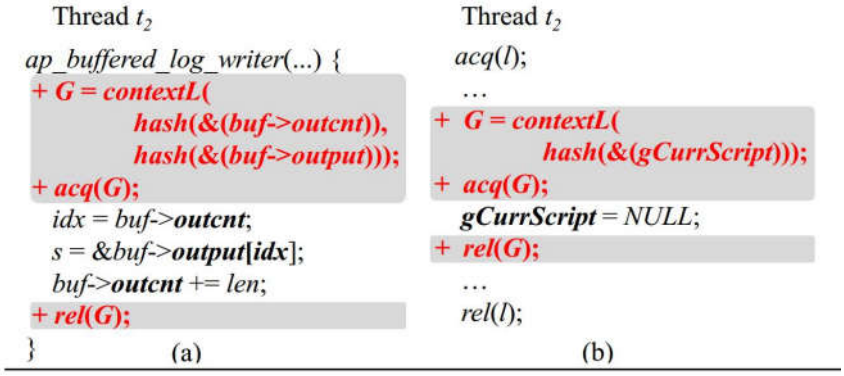

图 3. Grail 对原子性缺陷实例修复

如图 3 所示，针对过度序列化问题，Grail 根据原子性错误中的相关变量进行哈希计算得到一个与上下文变量相关的锁，从而避免了线程的完全序列化执行，但是 Grail 计算锁的过程复杂，修复代码难以理解和维护，并且该修复方式并不能保证正确修复，和图 2（b）中的分析一样，也会引入新的死锁问题。

根据以上分析，我们提出了一种基于代码结构，可适配性提供修复方案的修复技术 *AlphaFixer*.

2、 我们的工作

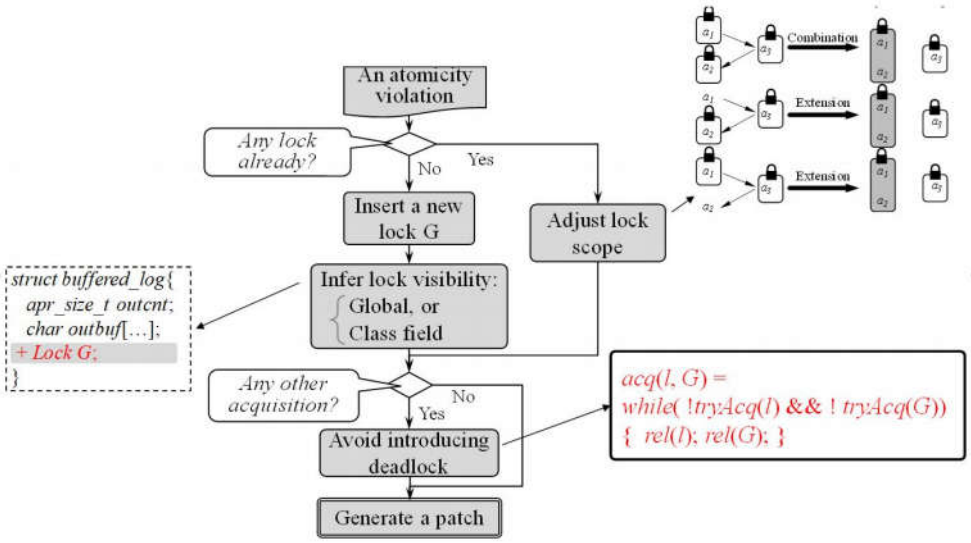

图 4. *AlphaFixer 修复策略框架*

如图 4 所示，*AlphaFixer* 对于给定的原子性缺陷，按照以下步骤进行修复：

第一步：确定导致原子性缺陷发生的变量周围是否已存在锁对变量进行保护，如果有，我们根据具体情况进行锁调整从而实现缺陷修复，进行第三步，如果没有，进行第二步；

第二步：确定导致原子性缺陷发生的变量的可见性，即全局变量或者类/结构体变量，根据变量的可见性，添加相应级别的保护锁；进行第三步；

第三步：确定调整锁或者添加保护锁的过程中是否引入了新的内部锁序，如果有，那么调用 DFixer 中的锁提前同步获取的方法，进行锁序消除。

我们以图 2 中的两个原子性缺陷实例来具体说明 *AlphaFixer 对他们的修复：*
如下图 5（a）所示,变量 buf->output 和 buf->outcnt 周围不存在现有保护锁，那么我们需要添加一个新的锁，由于是结构体变量，所以，我们在对应的结构体内添加一个 G 锁，然后在相应的位置添加该锁的获取和释放操作，即图中的红色标注代码部分。
如下图 5（b）所示，变量 gCurrScript 周围已经存在现有的保护锁 l，相应的，我们只需要调整 l 锁的保护范围进行缺陷修复，即途中的红色标注代码部分。

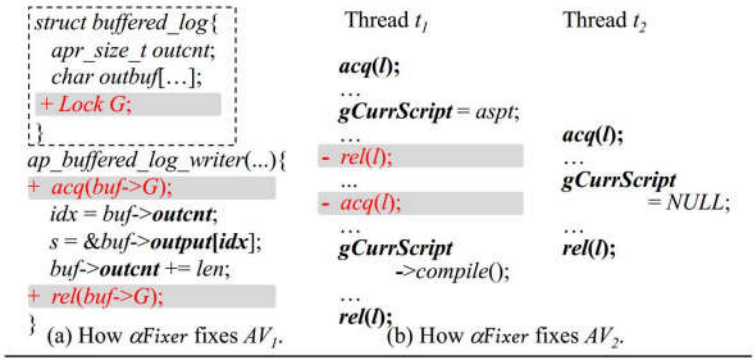

图 5. *AlphaFixer 对原子性缺陷实例的修复*

3、 实验结果

实验中，我们选取了 15 个程序中的 15 个原子性缺陷，从性能和修复正确性上进行了对比实验。

表 1. *AlphaFixer.*，GLA, Grail 的实验结果

#	#of fixed atom. violations				Avg. overhead			
Total	*GLA*	*Grail*	*HFix*	*αFixer*	*GLA*	*Grail*	*HFix*	*αFixer*
15	10 (67%)	10 (67%)	2 (13%)	15 (100%)	120.2%	82.9%	32.5%	21.1%

表 1 实验数据表明，和 GLA, Grail 以及 HFixer 方法相比，*AlphaFixer* 成功修复了所有的原子性缺陷，而 GLA 和 Grail 只能正确修复其中的 10 个原子性缺陷。HFixer 由于其可修复缺陷类型的限制，只能正确修复其中的 2 个原子性缺陷。另外，在性能上，*AlphaFixer.*仅仅引入了约 21.1%的 Overhead，而 GLA 和 Grail 分别引入了超过 120.2%和 82.9%的 Overhead。

为了对比 AlphaFixer 和 Grail 修复代码的可读性，我们特地设计了两个工具可读性比较的调研问卷，邀请了 40 位不同领域的参与者（包括 3 位数学专业，4 位金融，33 位计算机专业）填写我们的问卷。

表 2. *AlphaFixer*, Grail 可读性对比问卷调研结果

		Tool1 (*Grail*)	**Tool2** (*αFixer*)	**Any**
AV_1	Understandability	2.5%	87.5%	10.0%
	Readability	0.0%	90.0%	10.0%
	Larger Overhead	67.5%	12.5%	20.0%
	Preferred Fix	5.0%	95.0%	0.0%
AV_2	Understandability	7.5%	92.5%	0.0%
	Readability	2.5%	85.0%	12.5%
	Larger Overhead	75.0%	10.0%	15.0%
	Preferred Fix	10.0%	82.5%	7.5%

如表 2 数据显示，超过 85%的参与者认为，*AlphaFixer* 的修复代码比 *Grail* 的修复代码更具有可读性，更加简单易懂；并且超过 67%的参与者认为，*Grail* 修复代码比 *AlphaFixer* 可能带来更大的性能问题。

综上所述，*AlphaFixer* 相比于现有的原子性修复技术，具有以下优势：

（1） 能够正确修复原子性缺陷，保证不引入死锁问题
（2） 不会造成较大的性能缺陷
（3） 修复代码简洁易懂，可读性和可维护性强

作者简介：

蔡彦，2014 年于香港城市大学获得博士学位，之后在中国科学院软件研究所（计算机科学国家重点实验室）任副研究员。主要研究并发程序的测试问题，并注重在大规模真实程序中的应用。近五年来发表文章 20 多篇（CCF A 类 15 篇），其中在 CCF A 类顶级期刊和会议 IEEE TSE、ICSE、FSE 上以第一作者身份发表 9 篇。是 COMPSAC SETA 2018 Co-Program Chair，JSS (Journal of Systems and Software)的 Editorial Board 成员，FCS (Frontiers of Computer Science) 期刊 Young Associate Editor（青年 AE），多次为 IEEE TSE/ TR/TC/ TSC、JSS、The Computer Journal 等著名期刊审稿、担任多个国际会议 PC 等。获 FCS 期刊 2016 年优秀青年 AE 奖。更多信息见 http://lcs.ios.ac.cn/~yancai 。

曹玲微，中国科学院大学 2015 级硕士研究生。主要研究并发缺陷的检测与修复。研究生阶段，在 ICSE 和 FSE 会议上发表文章 4 篇。

赵靖，教授，博士导师。哈尔滨工业大学计算机系统结构专业博士，2010.8-2011.9 杜克大学电子与计算机工程系博士后，2015-2016 杜克大学电子工程系访问学者，哈尔滨工程大学支持的第二批青年骨干教师。在 TDSC ACM JETC，JSS，PE，《计算机学报》，《计算机研究与发展》等国内外著名期刊和国际会议上发表学术论文 20 余篇，其中被 SCI/EI 检索十几篇。主要研究方向为可信软件、移动计算和机器学习。可信软件方面，主要的研究内容为软件性能测试、软件安全测试及组合测试，软件系统的仿真及重要性采样；移动计算方面，主要的研究内容是网络信息分发优化、自组网的抗毁性分析；机器学习方面，主要的研究内容是计算机视觉，Hadoop 分布式文件系统，最近邻算法等。