软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(7):3314−3331 [doi: 10.13328/j.cnki.jos.006922]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

SMTLOC: 基于多源频谱的 SMT 求解器缺陷定位*

王笑爽[1], 周志德[1], 李晓晨[1], 江 贺[1], 任志磊[1,2]

[1](大连理工大学 软件学院, 辽宁 大连 116024)

[2](高安全系统的软件开发与验证技术工业和信息化部重点实验室 (南京航空航天大学), 江苏 南京 210016)

通信作者: 任志磊, E-mail: zren@dlut.edu.cn

摘 要: SMT 求解器作为重要的基础软件, 其存在的缺陷可能会导致依赖于它的软件功能失效, 甚至带来安全事故. 然而, 修复 SMT 求解器缺陷是一个十分耗时的任务, 因为开发者需要花费大量的时间和精力来理解并找到缺陷的根本原因. 虽然已有许多软件缺陷定位方面的研究, 但尚未有系统的工作研究如何自动定位 SMT 求解器缺陷. 因此, 提出一种基于多源频谱的 SMT 求解器缺陷定位方法 SMTLOC. 首先, 对于给定的 SMT 求解器缺陷, SMTLOC 提出一种枚举算法, 用以对触发该缺陷的公式进行变异, 从而生成一组不触发缺陷, 但与触发缺陷的公式具有相似执行路径的证人公式. 然后, SMTLOC 根据证人公式的执行路径以及 SMT 求解器的源码信息, 提出一种融合覆盖频谱和历史频谱的文件可疑度计算方法, 从而定位可能存在缺陷的文件. 为了验证 SMTLOC 的有效性, 收集 60 个 SMT 求解器缺陷. 实验结果表明, SMTLOC 的缺陷定位效果明显优于传统的频谱缺陷定位方法, SMTLOC 可以将 46.67% 的缺陷定位在 TOP-5 的文件内, 定位效果提升了 133.33%.

关键词: SMT 求解器; 缺陷定位; 覆盖频谱; 历史频谱

中图法分类号: TP311

中文引用格式: 王笑爽, 周志德, 李晓晨, 江贺, 任志磊. SMTLOC: 基于多源频谱的SMT求解器缺陷定位. 软件学报, 2024, 35(7): 3314–3331. http://www.jos.org.cn/1000-9825/6922.htm

英文引用格式: Wang XS, Zhou ZD, Li XC, Jiang H, Ren ZL. SMTLOC: Bug Localization for SMT Solver Based on Multi-source Spectrum. Ruan Jian Xue Bao/Journal of Software, 2024, 35(7): 3314–3331 (in Chinese). http://www.jos.org.cn/1000-9825/6922.htm

SMTLOC: Bug Localization for SMT Solver Based on Multi-source Spectrum

WANG Xiao-Shuang[1], ZHOU Zhi-De[1], LI Xiao-Chen[1], JIANG He[1], REN Zhi-Lei[1,2]

[1](School of Software Technology, Dalian University of Technology, Dalian 116024, China)

[2](Key Laboratory of Safety-critical Software of Ministry of Industry and Information Technology (Nanjing University of Aeronautics and Astronautics), Nanjing 210016, China)

Abstract: SMT solver is an important system software. Therefore, bugs in the SMT solver may lead to the function failure of software relying on it and even bring security incidents. However, fixing bugs in the SMT solver is time-consuming because developers need to spend a lot of effort in understanding and finding the root cause of the bugs. Although many studies on software bug localization have been proposed, there is no systematic work to automatically locate bugs in the SMT solver. Therefore, this study proposes a bug localization method for the SMT solver based on multi-source spectrums, namely SMTLOC. First, for a given bug in the SMT solver, SMTLOC proposes an enumeration-based algorithm to mutate the formula that triggers the bug by generating a set of witness formulas that will not trigger the bug but has a similar execution trace with the formula that triggers the corresponding bug. Then, according to the execution trace of the witness formulas and the source code information of the SMT solver, SMTLOC develops a technique based on the coverage spectrum and historical spectrum to calculate the suspiciousness of files, thus locating the files that contain the bug. In order to

* 基金项目: 南京航空航天大学科研基地创新 (理工类) 项目 (NJ2020022); 国家自然科学基金 (62072068, 62032004); 国家重点研发计划 (2018YFB1003900)
收稿时间: 2022-08-30; 修改时间: 2022-12-06; 采用时间: 2023-02-23; jos 在线出版时间: 2023-08-09
CNKI 网络首发时间: 2023-08-10

evaluate the effectiveness of SMTLOC, 60 bugs in the SMT solver are collected. Experimental results show that SMTLOC is superior to the traditional spectrum bug localization method and can locate 46.67% of the bugs in TOP-5 files, and the localization effect is improved by 133.33%.

Key words: SMT solver; bug localization; coverage spectrum; historical spectrum

SMT (satisfiability modulo theory, 可满足性模理论) 求解器[1]是用来判定一阶逻辑公式是否可满足的重要基础软件. 结合高效的可满足性判定算法, SMT 求解器已被广泛应用于测试程序生成[2,3]、程序分析与验证[1,4]、程序修复[5]、线性逻辑约束公式优化问题求解等领域. 然而, SMT 求解器同样可能出现缺陷, 从而导致依赖于 SMT 求解器的软件功能失效, 甚至带来安全事故. 例如, 在程序验证中, 错误的可满足性结果会使程序验证结果失效, 对安全领域产生不良影响[6]. 但是, 修复 SMT 求解器缺陷是一个十分耗时的任务, 因为开发者需要花费大量的时间来理解并找到缺陷的根本原因. 本文统计了著名开源 SMT 求解器 Z3[7]自 2015 年以来缺陷仓库中的 4 515 个已修复缺陷的提交与关闭时间. 结果表明, 修复一个 Z3 缺陷所需的平均时间为 39.15 天. 因此, 为提升 SMT 求解器缺陷的修复效率, 自动定位 SMT 求解器缺陷已成为一个亟待解决的问题.

软件缺陷定位技术是在执行测试用例集后发现有部分测试用例执行失败时, 确定缺陷所在具体位置的一种分析方法[8]. 目前, 针对通用的软件, 许多缺陷定位技术[9−17]已经被提出, 如基于频谱的缺陷定位技术 SBFL (spectrum-based fault localization)[9−11], 基于切片的缺陷定位技术[12,17], 基于变异的缺陷定位技术[13−16]. SBFL 技术利用失败测试用例与通过测试用例执行路径的差异来定位缺陷所在文件. 但是 SMT 求解器是比较庞大和复杂的软件, 失败测试用例和通过测试用例执行路径差异较大, SBFL 技术很难有效地定位 SMT 求解器缺陷. 基于切片的技术和基于变异的技术都要求对源程序进行修改, 但是由于 SMT 求解器的复杂性, 对源程序进行切片和变异是比较耗时的. 例如, 基于变异的缺陷定位技术 MUSE[13]针对每个特定的缺陷, 对平均代码行约为 10 KLOC 的 C 语言程序进行变异, 所花费时间平均为 3.57 h. 而目前主流的 SMT 求解器如 Z3[7]、CVC5[18]等都具有几十万甚至上百万行的代码量, 使得在 SMT 求解器源程序上执行变异比较耗时. Chen 等人提出了基于变异的技术 DiWi[19]和 RecBi[20], 对编译器进行缺陷定位. DiWi 和 RecBi 使用变异的方法对触发缺陷的测试程序 (如 C 语言程序) 进行变异, 从而生成一组不会触发缺陷, 但执行路径与触发缺陷的测试程序相似的证人测试程序, 进而将编译器缺陷定位问题转换成证人测试程序生成的问题. 然后, 通过比较证人测试程序和失败测试程序的执行路径差异来对编译器缺陷进行定位. 然而, DiWi 和 RecBi 也同样不能直接用于 SMT 求解器缺陷定位. 一方面, DiWi 和 RecBi 所提出的变异方法仅适用 C 语言等编程语言, 而 SMT 求解器的输入则是 SMT-LIB 格式的逻辑公式; 另一方面, DiWi 和 RecBi 仅以执行路径的差异来定位缺陷, 而忽略了代码自身包含的信息, 定位效果不理想. 因此, 基于 DiWi 和 RecBi 的缺陷定位思想, 本文提出了一种基于多源频谱的 SMT 求解器缺陷定位方法 SMTLOC, 主要解决两个方面的挑战.

1) 如何生成多样化的证人公式. 给定 SMT 求解器缺陷, 要获得与触发缺陷时相似的执行路径, 那么一种便捷的方法是对触发缺陷的公式进行变异. Chen 等人[19]提出, 对触发缺陷的输入进行变异的粒度越小, 越容易产生与触发缺陷的输入相似的执行路径. 同时, 变异后的输入应具有多样性, 从而更有利于排除与缺陷无关的文件. 因此, 针对 SMT 求解器缺陷定位, 如何设计细粒度的公式变异方法, 并生成一组多样化的证人公式就成为必须解决的挑战.

2) 如何利用多源代码信息, 提升缺陷定位效果. 根据执行路径的差异, 在一定程度上可以定位缺陷. 但代码自身在演化过程中包含了丰富的信息, 这些信息也有助于缺陷定位. 例如, 在软件演化过程中, 代码被修改次数越多, 表明这部分代码越有可能产生新的缺陷. 这意味着代码的修改信息将有助于缺陷的定位, 因此, SMT 求解器缺陷的另一个挑战是如何利用多源代码信息, 提高缺陷定位效果.

首先, 针对第 1 个挑战, SMTLOC 根据 SMT 求解器输入公式的特点, 设计了两类共 307 种公式变异规则, 并提出一种枚举策略来生成多样化的证人公式. 对于给定的触发缺陷的公式, SMTLOC 将其转化为抽象语法树 (abstract syntax tree, AST), 然后遍历 AST 中的节点, 并对节点可能的变异操作进行枚举, 进而生成一组多样化的

证人公式. 其次, 针对第 2 个挑战, SMTLOC 提出一种融合覆盖频谱和历史频谱的文件可疑度计算方法. 根据证人公式产生的执行路径信息, SMTLOC 首先利用 Ochiai 公式[21]计算被执行代码的覆盖频谱得到文件可疑度; 然后, 通过分析 SMT 求解器的演化历史, 得到由源代码语句的历史频谱计算的文件可疑度; 最后, 融合覆盖频谱和历史频谱计算最终文件可疑度. 根据文件可疑度对文件进行排名, 可疑度越高, 则说明缺陷越有可能包含在该文件中.

为了评估 SMTLOC 的有效性, 本文首先以 Z3[7]和 CVC5[18]为研究对象, 构建了一个包括 60 个 SMT 求解器缺陷的数据集, 其中包含 30 个 Z3 求解器缺陷和 30 个 CVC5 求解器缺陷. 然后, 本文将 SMTLOC 与 SBFL 进行比较, 实验结果表明 SMTLOC 的缺陷定位效果显著优于 SBFL, SMTLOC 在 TOP-1/5 文件级别可以比 SBFL 多定位 900%/133.33% 的缺陷. 同时, 本文进一步研究了证人公式生成方法和文件可疑度计算方法的有效性. 为了研究证人公式的有效性, 本文将使用 SMTLOC 方法生成的证人公式替换为开发人员提供的测试公式, 该变体称为 $\text{SMTLOC}_{\text{dev}}$. 为了研究枚举算法在证人公式生成中的贡献, 本文将其替换为基于随机搜索的方法 $\text{SMTLOC}_{\text{rand}}$ 和基于马尔可夫链的方法 $\text{SMTLOC}_{\text{mc}}$. 实验结果表明 SMTLOC 的效果优于 $\text{SMTLOC}_{\text{dev}}$、$\text{SMTLOC}_{\text{rand}}$ 和 $\text{SMTLOC}_{\text{mc}}$, SMTLOC 可以在 TOP-1/5 上比 $\text{SMTLOC}_{\text{dev}}$、$\text{SMTLOC}_{\text{rand}}$ 和 $\text{SMTLOC}_{\text{mc}}$ 分别多定位到 150%/55.56%、100%/27.27% 和 100%/27.27% 的缺陷. 文件可疑度计算方法包含基于覆盖频谱的文件可疑度计算和基于历史频谱的文件可疑度计算. 为了探究覆盖频谱和历史频谱的有效性, 本文对 SMTLOC 设计了两个变体 SMTLOC_h 和 SMTLOC_m. 实验结果表明, 对于收集到的所有缺陷, SMTLOC 的缺陷定位效果显著优于 SMTLOC_h 和 SMTLOC_m, SMTLOC 可以在 TOP-1/5 上比 SMTLOC_h 和 SMTLOC_m 分别多定位到 66.67%/133.33% 和 100%/33.33% 的缺陷. 综上所述, 本文的贡献包含以下 3 个方面.

1) 本文是第 1 次研究 SMT 求解器缺陷定位问题.

2) 提出了一种基于多源频谱的缺陷定位方法.

3) 构建了一个包括 60 个 SMT 求解器缺陷的数据集, 可用于进一步研究 SMT 求解器缺陷定位和修复.

本文第 1 节介绍研究背景, 并通过一个简单实例引出研究动机. 第 2 节介绍本文提出的方法框架 SMTLOC. 第 3 节通过对比实验验证所提方法的有效性. 第 4 节讨论和分析相关工作. 第 5 节总结全文.

1 研究背景和动机

可满足性模理论主要研究在特定理论下的一阶逻辑公式的可满足性问题, 理论包含整型、实数型、布尔型、字符串等. 目前, 针对可满足性模理论, 已有高效的算法被提出, 并且开发了被广泛使用的 SMT 求解器, 如 Z3 和 CVC5. 在程序验证等领域, SMT 求解器作为重要的基础软件, 是保证相关算法或技术有效性的关键所在.

对于给定的公式输入, SMT 求解器可以自动判定该公式是否可满足, 并在公式可满足时, 给出对应的解. 然而, 若是 SMT 错误地将可满足的公式判定为不可满足, 或者将不可满足的公式判定为满足, 那么对于依赖 SMT 求解器的应用, 其有效性将会受到严重威胁, 甚至带来安全性事故. 这两类缺陷通常由依赖于 SMT 求解器的客户端程序直接发现, 是 SMT 求解器最严重的缺陷[22], 称之为健壮性缺陷. 此外, SMT 求解器还存在其他类型的缺陷. 由于 SMT 求解器支持不可判定的理论, 它们不能总是确定一个公式是否可满足, 有时会返回未知的结果. 如果 SMT 求解器将一个可判定的公式报告为未知, 说明求解器存在着缺陷. 除了上述缺陷, SMT 求解器还可能会存在崩溃、超时问题. 崩溃是指 SMT 求解器在求解过程中意外终止, 超时意味着 SMT 求解器没有在给定时间内求解出公式的结果[23].

图 1 是 Z3 缺陷仓库中的一个触发缺陷的公式, 该公式结果应该是可满足的, 即 Z3 应该返回 sat, 但当 Z3 使用编译选项“smt.string_solver=seq”求解该公式时, 却将该公式判定为不可满足, 返回 unsat. 这个缺陷出现在文件 theory_seq.cpp 中, 根本原因是开发人员写错了三元分支条件的索引. 然而, 当开发者对该缺陷进行分析定位时, 可能需要检查 271 个源码文件才能定位该缺陷所在的文件, 因为包含该缺陷的 Z3 求解器拥有 271 个源代码文件. 为了自动定位软件中的缺陷, Chen 等人[19]提出, 若是对软件的输入进行细粒度的变异, 则可以产生与触发缺陷的输入相似的执行路径, 从而根据路径差异排除与缺陷无关的源代码文件, 进而定位到缺陷所在位置. 基于

该思想, 本文对图 1 的公式进行变异, 生成了 8 个证人公式. 然后, 根据证人公式与触发缺陷的公式的执行路径差异建立覆盖频谱, 最后使用 Ochiai 公式[21]根据覆盖频谱计算文件可疑度. 通过这种方式, 存在缺陷的文件被排在了第 4 位.

```
(set logic QF_S)
(declare-fun E ( ) String)
(assert (= (str.++ E “aa” E “ab” E) (str.++ “a” E E “aabaa” ))
(check-sat)
```

图 1　触发 Z3 缺陷的公式示例 (https://github.com/Z3Prover/z3/issues/3100)

上述示例基于公式执行路径的差异来定位 SMT 求解器缺陷的可能位置, 但该方法只从覆盖频谱的角度进行缺陷定位, 而忽略了代码本身包含的丰富信息, 缺陷定位准确度不够理想. 郭肇强等人[24]提出, 软件项目在演化过程中会通过版本更迭来实现对项目的更新和完善, 每一次更新都会向代码库中添加丰富的历史信息. 与此同时, 多数缺陷也会被引入到项目中. Hassan 等人[25]利用代码修改的复杂性来估计文件出错的概率, 其原理是随着文件修改次数的增加, 文件存在缺陷的概率也在增加. 这些观点表明了分析软件的历史信息可以为缺陷定位带来一些帮助. 因此, 本文从历史信息角度来改善缺陷定位效果. 本文提出使用历史切片[26]的方法跟踪缺陷版本中每个代码语句的演化历史, 根据代码语句在不同版本中是否被修改建立历史频谱, 然后根据历史频谱计算文件可疑度. 最后将基于覆盖频谱和历史频谱计算的文件可疑度结合对 SMT 求解器文件进行排名. 通过这种方式, 图 1 中存在缺陷的文件被排在了第 1 位. 存在缺陷的文件排名越靠前, 开发者就可以花费更少的精力找到缺陷位置.

2　SMTLOC

本节介绍基于多源频谱的 SMT 求解器缺陷定位方法 SMTLOC 的具体步骤. 首先, 第 2.1 节介绍 SMTLOC 的方法框架, 阐述方法的主要流程. 其次, 第 2.2 节介绍本文提出的证人公式生成方法. 最后, 在第 2.3 节详细介绍文件可疑度计算方法.

2.1　方法框架

SMTLOC 的方法框架如图 2 所示. SMTLOC 的核心思想是构造一组不触发缺陷, 但与触发缺陷的公式具有相似执行路径的证人公式, 然后根据证人公式和触发缺陷的公式的执行路径差异, 以及 SMT 求解器缺陷版本的代码修改信息进行缺陷定位. 因此, SMTLOC 包含两部分内容: 证人公式生成和文件可疑度计算. SMTLOC 的输入为 SMT-LIB 理论、触发缺陷的公式和 SMT 求解器缺陷版本. SMT-LIB 理论是 SMT 求解器的输入语法, 提供了 SMT 求解器使用的理论的标准严格描述. 根据 SMT-LIB 理论, 可以设计 SMT 求解器输入公式的变异规则, 然后根据触发缺陷的公式和设计的变异规则可以构造证人公式. 另外, 根据 SMT 求解器缺陷版本可以获得缺陷版本的代码修改信息.

SMTLOC 方法的前半部分为证人公式生成. 生成证人公式的目的是构造一组不触发缺陷的公式. 为了生成证人公式, 首先需要根据 SMT-LIB 理论设计变异规则得到变异规则集合, 然后将触发缺陷的公式转化为 AST, 之后使用枚举算法将 AST 中可能的变异操作进行枚举变异, 生成一组变异后的 AST 集合, 进而得到一组不触发缺陷的证人公式. 之后分别收集触发缺陷的公式和证人公式在执行过程中的覆盖信息. 本文使用枚举算法来选择变异规则, 原因是枚举算法可以对公式中可变异的节点进行最详尽的变异操作, 可以生成多样性的证人公式.

SMTLOC 方法的后半部分为文件可疑度计算. 对于给定的 SMT 求解器缺陷版本, 首先根据方法前半部分生成的证人公式与触发缺陷的公式的执行路径差异建立语句覆盖频谱, 使用 Ochiai 公式[21]计算被执行语句的覆盖频谱得到覆盖信息文件可疑度. 然后获取 SMT 求解器缺陷版本中文件的语句演化历史, 根据语句是否被修改为每个语句建立历史频谱, 基于所有语句的修改次数计算历史信息文件可疑度. 之后结合从覆盖频谱和历史频谱得到的文件可疑度得到最终文件可疑度. 最后, 本文将文件可疑度由高到低排序获得一个可疑文件排名列表. 开发人员可以通过检查这个文件列表快速定位缺陷所在文件.

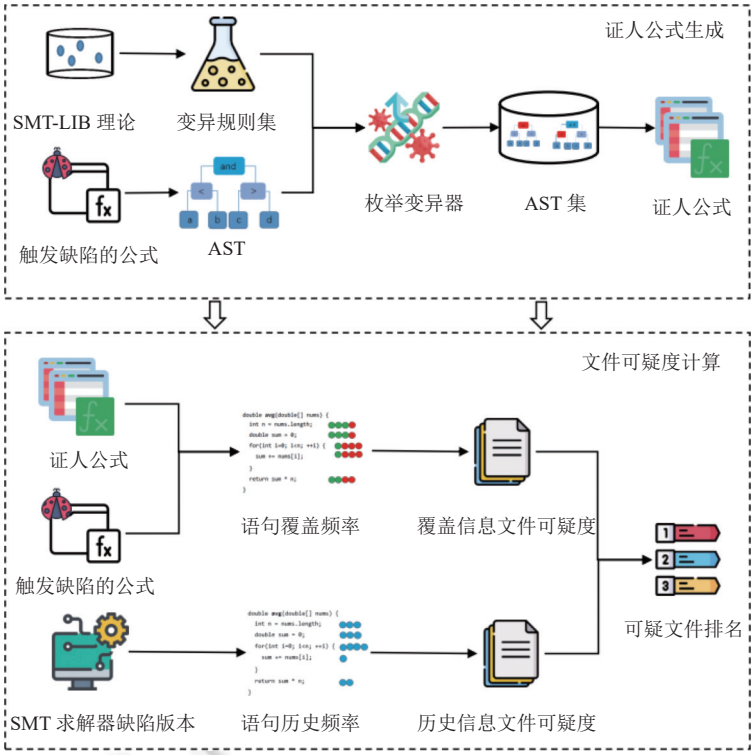

图 2 SMTLOC 方法框架

2.2 证人公式生成

给定 SMT 求解器缺陷, 要获得一组不触发缺陷, 但与触发缺陷的公式具有相似执行路径的证人公式, 需要对触发缺陷的公式进行变异操作. 因此, 本文根据 SMT 求解器的输入语法 SMT-LIB 理论设计变异规则, 然后采取一个合适的策略选择变异规则, 对触发缺陷的公式形成的 AST 进行变异生成一组证人公式. 第 2.2.1 节和第 2.2.2 节分别介绍公式变异规则和变异规则选择.

2.2.1 公式变异规则

对触发缺陷的公式进行变异的目标是引入不同的执行路径来改变 SMT 求解器的执行结果, 生成一组不触发缺陷的证人公式. 此外, 还需要保持证人公式和触发缺陷的公式执行路径的相似性, 从而根据路径差异排除与缺陷无关的源代码文件. 针对这些变异目标, 本文提出了 4 种替换操作和 3 种剪枝操作, 对触发缺陷的公式进行变异. 替换操作指将公式中的常量、变量、操作符和函数进行替换, 剪枝操作指将公式中包含的某个子公式进行删除.

参照先前工作[27], 本文首先将触发缺陷的公式解析为 Instance 类. Instance 类包括 4 个部分: Set logic、Declare、Assert 和 Check sat. Set logic 部分指定了公式使用的逻辑; Declare 部分声明了公式可能使用到的变量及变量类型; Assert 部分使用一个列表存储抽象语法树 AST, 公式中每一个断言语句都会形成一棵 AST; Check sat 部分是公式的固定语法, 让 SMT 求解器查询解算程序以确定断言的可满足性. 如果所有的断言都是可满足的, 则公式是可满足的, 否则公式是不可满足的. 本文的测试公式是基于 SMT-LIB 2.6 实现的. 由于 SMT-LIB 包含众多理论, 如 Ints, Reals, Strings, Bool 等. 每种理论中的操作符、常量、变量、函数都不完全相同. 例如, Strings 理论中包含字符串函数, 例 (str.contains String String Bool), 而其他理论都没有. 因此, 针对每种理论, 本文都设计相应的替换变异规则. SMT-LIB 的不同理论形成的断言语句 Assert 包含的子断言类型主要有布尔型, 数值型和字符串型, 因此我们针对这 3 种类型设计剪枝操作.

本文对触发缺陷的公式中 Assert 部分形成的 AST 节点进行替换操作, AST 节点有常量、变量、操作符和函数. 因此, 本文提出了 4 种替换操作: 常量替换、变量替换、操作符替换和函数替换. 以 Strings 理论为例, 对于常

量替换, 分为数值型常量替换和字符串型常量替换, 数值型常量替换是对数值进行加 1 或减 1 操作, 字符串型常量替换是将字符串替换为空字符串. 对于变量替换, 每个变量都可以替换为公式中的另一个类型相同的变量. 对于操作符替换, 每个操作符都可以替换为另一个类型相同的操作符. 对于函数替换, 指的是将公式中的函数替换为另一个输入输出相同的函数, 例如 (str.contains String String Bool) 可以替换为 (str.prefixof String String Bool). 每个变异类型的替换操作变异规则如表 1 所示. 针对 SMT 求解器的每种理论, 我们都设计相应的替换变异规则. Bool 理论的替换变异规则有 14 种, Strings 理论的替换变异规则有 118 种, Ints 理论的替换变异规则有 72 种, Reals 理论的替换变异规则有 72 种.

表 1 替换操作变异规则

变异类型	变异规则
常量	数值型常量: value+1, value–1; 字符串型常量: value替换为“ ”
变量	使用公式中类型相同的一个变量替换另一个变量
操作符	使用同一个类别的二元操作符替换另一个二元操作符, 包括算数操作符、关系操作符和和逻辑操作符
函数	使用输入输出类型相同的函数替换另一个函数

本文对触发缺陷的公式中 Assert 部分形成的 AST 非叶节点进行剪枝操作. 然而, 并不是所有的非叶节点都适合修剪, 因为在修剪 AST 结构时可能会引入语法错误. 例如, 有些断言节点会对变量进行定义, 如果定义变量的节点被删除, 公式会产生语法错误. 所以, 本文规定禁止修剪在断言中定义变量的节点. 根据 SMT 公式的节点类型, 本文提出了 3 种剪枝操作: 布尔型节点剪枝, 数值型节点剪枝和字符串型节点剪枝. 对于布尔型节点剪枝, 每个布尔型节点都可以使用 True/False 进行替换, 替换后将它的孩子节点删除. 对于数值型节点剪枝, 每个数值型节点都可以使用数值 0 进行替换, 替换后将孩子节点删除. 对于字符串型节点, 使用空字符串替换节点, 替换后将孩子节点删除. 每个变异类型的剪枝变异规则如表 2 所示. 布尔型节点的剪枝规则有 15 种, 数值型节点的剪枝规则有 10 种, 字符串型节点的剪枝规则有 6 种.

表 2 剪枝操作变异规则

变异类型	变异规则
布尔型节点	将布尔型节点的值替换为True/False, 孩子节点置空
数值型节点	将数值型节点的值替换为0, 孩子节点置空
字符串型节点	将字符串型节点的值替换为“ ”, 孩子节点置空

2.2.2 变异规则选择

对于一个给定的触发缺陷的公式, 不同的变异规则在产生多样化的证人公式上的有效性上是不同的. 因此, 需要选择一个策略来指导证人公式的生成. 本文研究了所有收集的触发缺陷的公式的特点. 经观察得知, 收集到的公式都是已经被约简过的, 具有简短的特点. 为了生成尽可能多样的证人公式, 本文使用枚举算法选择变异规则对公式进行变异, 因为枚举算法可以将可变异节点的所有变异规则进行穷举, 进而生成不同的证人公式.

如图 3(a) 是一个触发 Z3 求解器缺陷的公式. 为了生成证人公式, 首先将该公式中所有断言语句转换为 AST, 如图 3(b) 所示. 然后, 使用广度优先遍历的方式对图 3(b) 的 AST 进行遍历. 对于每个遍历到的节点, 判断该节点是否可变异. 比如遍历到 assert 节点, 该节点是断言的特定组成部分, 则该节点不能进行变异. 若节点不可变异, 继续遍历下一个节点. 若遍历到可变异的节点, 枚举该节点对应的替换操作和删除操作. 以遍历到的第 1 个“/”节点为例, 我们可以枚举出其替换操作有: 将“/”替换为“+”; 将“/”替换为“–”; 将/替换为“*”, 其删除操作有: 将“/”替换为“0”, 并将孩子节点删除. 图 3(d) 是将“/”替换为“+”后对应的 AST, 该 AST 对应的公式如图 3(c) 所示.

对图 3(a) 公式中的断言语句对应的 AST 中每个可变异的节点进行变异, 可以生成一组新测试公式, 但我们无法确定新测试公式是否会触发 SMT 求解器的缺陷. 因此, 在每次变异生成新测试公式后, 本文使用触发缺陷的求解器和另外一个主流的求解器对新测试公式进行差分测试. 若求解结果一致, 则认为变异生成的新测试公式不再触发求解器的缺陷, 为证人公式, 否则丢弃本次变异生成的新测试公式.

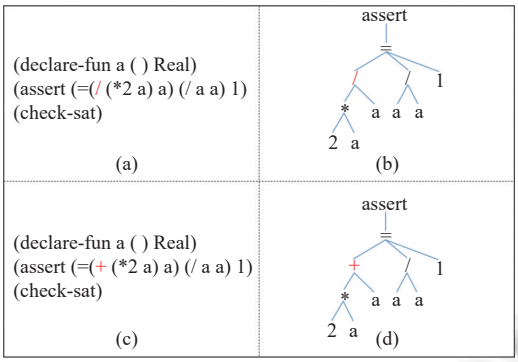

图 3 变异测试公式示例

算法 1 呈现了变异规则选择的主要步骤. 算法的输入是给定的触发缺陷的公式集合以及针对不同理论设计的变异规则, 输出是证人公式集合. 算法的第 2 行选择一个触发缺陷的公式, 第 3 行将触发缺陷的公式中的断言语句转换为抽象语法树 AST, 并对 AST 进行广度优先遍历. 5、6 行判断遍历到的节点是否可变异, 并枚举可变异节点的所有变异规则. 7–13 行枚举节点对应的所有变异规则, 对节点进行变异后形成的新公式进行差分测试. 若生成的新公式不再触发缺陷, 将其添加到证人公式集, 并收集 SMT 求解器执行新公式过程中的覆盖信息. 第 15 行遍历下一个 AST 节点. 循环终止条件为所有 AST 的节点全部被遍历完. 该算法的时间复杂度约为 O(mn), m 为 AST 的节点数量, n 为节点对应的变异规则数量, n 不固定 (取值范围在 1–5).

算法 1. 枚举算法伪代码.

输入: 触发缺陷的公式 S_{fail}, 每种理论的变异规则列表 S_{MR};
输出: 证人公式集合 S_{pass}.

```
(1) WHILE not termination do
(2)   F ← Select( S_fail )
(3)   Parse F into AST and traverse AST nodes
(4)   WHILE node != null do
(5)     IF node is mutating THEN
(6)       MR_n ← Enumrate(node, S_MR )
(7)       FOR MR ∈ MR_n do
(8)         F_m ← Mutate(F, node, MR)
(9)         Different-Test( F_m )
(10)        IF F_m is Passing THEN
(11)          S_pass ← S_pass ∪ F_m
(12)          collect-Cov( F_m )
(13)        END IF
(14)      END FOR
(15)    END IF
(16)    node ← node + 1
(17)  END WHILE
(18) END WHILE
```

2.3 文件可疑度计算

在生成一组证人公式后, 本文根据证人公式与触发缺陷的公式的执行路径差异, 以及 SMT 求解器缺陷版本的历史修改信息提出文件可疑度计算方法, 然后根据计算得出的文件可疑分数对文件进行排名. 因此, 本节将文件可疑度计算方法分为 3 个部分: 基于覆盖频谱的文件可疑度计算 (第 2.3.1 节), 基于历史频谱的文件可疑度计算 (第 2.3.2 节), 可疑文件排名 (第 2.3.3 节).

2.3.1 基于覆盖频谱的文件可疑度计算

由 2.2 节对触发缺陷的公式变异生成一组证人公式后, 本文基于证人公式和触发缺陷的公式的执行路径差异建立覆盖频谱, 然后根据覆盖频谱计算文件的可疑度. 与 DiWi[19]和 RecBi[20]相似, 本文在实验中也采用 SBFL 技术的思想来计算文件可疑度. 具体来说, 在实验中使用 Ochiai 公式[21]计算公式执行路径中的每个语句的得分, 因为据报告[18,28,29], 它是最先进的 SBFL 公式. Ochiai 公式的定义如下:

$$Mscore(s)=\frac{ef_s}{\sqrt{(ef_s+nf_s)(ef_s+ep_s)}} \tag{1}$$

其中, ef_s 表示执行该语句 s 且触发缺陷的公式的数量; nf_s 表示未运行语句 s 且触发缺陷的公式的数量; ep_s 表示执行了语句 s 且并未触发缺陷的证人公式的数量. 由于只有一个给定的触发缺陷的公式, 并且只考虑该公式在求解过程中涉及的 SMT 求解器文件中的语句, 因此 ef_s=1, nf_s=0. 我们可以将公式简化为:

$$Mscore(s)=\frac{1}{\sqrt{1+ep_s}} \tag{2}$$

因为本文的目标是定位包含缺陷的文件. 因此, 和先前的工作[19,20]相同, 聚合文件中每个语句的可疑分数作为文件的可疑分数. 每一个参与编译的 SMT 求解器文件的覆盖信息可疑度分数定义如下:

$$Mscore(f)=\frac{\sum_{i=1}^{n_f}Mscore(s_i)}{n_f} \tag{3}$$

其中, n_f 表示在编译过程中, 求解器文件 f 所有被执行到的语句的数量.

2.3.2 基于历史频谱的文件可疑度计算

借鉴覆盖频谱的思想, 本文从另外一个频谱维度计算文件可疑度, 即历史频谱. 在软件开发过程中, 研究人员会进行多次提交来修改代码以增加新的功能或改正错误, 源代码文件中的语句也被修改了多次. 但是, 并不是所有提交都包含对所有代码行的修改, 开发人员的一次提交可能只修改了某个源代码文件中的某些代码行. 而代码被修改次数越多, 表明这部分代码越有可能产生新的缺陷. 因此, 我们跟踪 SMT 求解器缺陷版本每个代码语句的演化过程并为其建立历史频谱, 根据文件中语句被修改次数计算文件可疑度.

为了获得源代码语句的演化历史, 本文首先使用 Diff 工具获得缺陷版本中每个文件从初始版本到缺陷版本之间每两个连续版本的内容差异. 以图 4 为例, 该图展示了 Z3 求解器中的 qe_arith.cpp 文件在提交 b62d73f20 前后对应的两个版本的部分内容差异. 从图中可以看出, 使用 Diff 工具会得到代码行 3 种不同类型的修改: 插入、删除和更新. 在图 4 中, 355,359c365 表示前一个版本的 355–359 行更新后变成了 365 行; 385d392 表示前一个版本的 385 行在后一个版本已被删除, 其后面一行对应的是 392 行; 413a421 表示前一个版本的 413 行所对应的后一个版本的 420 行后新插入了一行.

使用 Diff 工具获得两个连续版本之间的文件差异后, 对两个连续版本中的语句建立映射关系. 对于文件中没有被修改的代码行, 可以直接建立映射. 对于文件中被修改的代码行, 分 3 种类型进行映射. 如图 5 所示是 Z3 求解器中的 qe_arith.cpp 文件在两个连续版本中的部分内容差异, "…"代表上下文未修改内容, 可以直接建立映射; 而对于插入、删除和更新的语句, 需要为图 5(a) 中的语句建立前向映射 forward, 为图 5(b) 中的语句建立后向映射 backward. 对于插入语句, 也就是经提交 b62d73f20 修改后, qe_arith.cpp 文件中插入了语句 421 行, 则后一个版本的 backward 为空; 对于删除语句, 也就是经提交 b62d73f20 修改后, qe_arith.cpp 文件中的语句 385 行被删除, 则前

一个版本的 forward 为空; 对于更新语句, 也就是经提交 b62d73f20 修改后, qe_arith.cpp 文件中的语句 355–359 行被更新为 365 行, 我们需要找到前一个版本的语句 355–359 行和后一个版本的语句 365 行的映射关系. 为了找到最佳映射关系, 本文借鉴历史切片的工作[26], 将其作为寻找加权二分图的最小匹配问题来处理. 首先将每个语句视为一个顶点, 得到两个不相交的顶点集 A: {355, 356, 357, 358, 359}和 B: {365}, 然后将 A 中的每个语句与 B 中的每个语句连接起来, 使用 Levenshtein 编辑距离[30]计算连接的两行语句之间的权值. 为了获得 Levenshtein 编辑距离, 本文将每一行语句标记为单词向量, 然后计算将一个单词向量变为另一个单词向量所需的单个单词编辑的最小次数. 数字越小, 说明这两个语句之间的相似性越高. 最后, 使用 Kuhn-Munkres 算法[31]找到 A 和 B 之间的最佳匹配关系. 经匹配, 图 5(a) 图中的 359 行对应图 5(b) 图中的 365 行, 图 5(a) 图中的{355, 356, 357, 358}行的映射为空.

```
355,359c365
<            expr* e = kv.m_key;
<            if (!var_mark.is_marked(e)) {
<               mark_rec(fmls_mark, e);
<            }
<            index2expr.setx(kv.m_value, e, 0);
---
>            index2expr.setx(kv.m_value, kv.m_key, 0);


385d392
<            row const& r = rows[i];
413a421
>            t = mk_add(ts);
```

图 4 使用 Diff 工具获得的文件部分差异

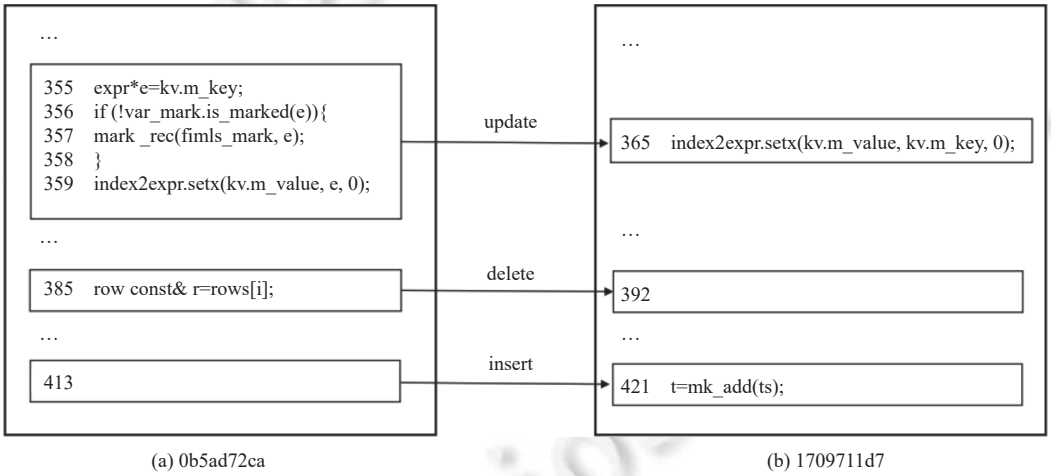

图 5 前后两个版本部分语句映射

由于我们的目标是找到缺陷版本中每个语句被修改的次数, 而很多语句在中间版本被添加或删除, 若从初始版本开始找每两个连续版本语句的映射关系会导致很多语句的映射关系被忽略. 因此, 我们从后向前进行分析, 首先找到缺陷版本的后向映射 backward, 即找到了缺陷版本与其上一个版本的映射关系, 进而知道版本所有文件中每个语句是否被修改. 通过这种方式, 可以得到缺陷版本中的每个语句向后映射对应的前一个版本的语句, 然后根据每两个连续版本之间语句的映射, 得到缺陷版本所有语句的演化历史.

在得到缺陷版本所有语句的演化历史后, 计算缺陷版本中每个语句在历史版本演化过程中被修改的次数. 语句修改次数越多, 则该语句应该有更高的可疑度. 因此, 语句可疑度的定义如下所示:

$$Hscore(f)=\frac{times(s)}{times(v)} \tag{4}$$

其中, $times(s)$ 指语句 s 在历史版本中被修改的次数, $times(v)$ 指缺陷版本 v 中所有语句一共被修改的次数. 和基于覆盖频谱计算文件可疑度相同, 聚合文件中每个语句的可疑分数作为该文件的历史信息可疑分数. 每一个 SMT 求解器文件的历史信息可疑分数定义如下:

$$Hscore(f)=\frac{\sum_{i=1}^{n_f} Hscore(s_i)}{n_f} \tag{5}$$

其中, n_f 表示求解器文件 f 中所有语句的数量.

2.3.3 可疑文件排名

根据覆盖频谱和历史频谱分别计算文件可疑度后, 我们结合这两种可疑度得到文件最终可疑度. 文件最终可疑分数计算公式如下:

$$Score(f)=\begin{cases}\alpha\times Mscore(f)+(1-\alpha)\times Hscore(f), & f\in F_{cov}\\ 0, & f\notin F_{cov}\end{cases} \tag{6}$$

其中, α 是基于覆盖频谱的文件可疑度和基于历史频谱的文件可疑度的组合权重. 在默认情况下, α 的值设置为 0.5. 本文在第 3.4 节中的 RQ4 研究了 α 对实验结果的影响. 在公式 (2) 中, 本文将没有被触发缺陷的公式覆盖到的文件可疑度分数设置为 0, 原因是如果缺陷版本的某个文件没有被触发缺陷的公式执行到, 它不太可能是缺陷的根本原因. 使用文件最终可疑分数 $Score(f)$ 对缺陷版本的可疑文件进行排序可以得到文件最终排名.

3 实验分析

3.1 实验数据

本文使用两个主流的 SMT 求解器 Z3 和 CVC5 作为实验对象. 为了探索本文提出的 SMTLOC 方法的效果, 我们从 Z3 和 CVC5 缺陷仓库中手工收集了 60 个缺陷, 其中每个求解器 30 个缺陷. 本次实验收集的缺陷目前包含两类健壮性缺陷: SMT 求解器将可满足的公式报告为不可满足, SMT 求解器将不可满足的公式报告为可满足. 收集这两类缺陷是因为健壮性缺陷是 SMT 求解器最严重的缺陷[22], 其结果的不正确性会直接使依赖于 SMT 求解器的应用程序失效. 目前, 已有很多论文[22,32−34]提出针对该类缺陷对 SMT 求解器进行测试. 在收集过程中, 每一个收集的缺陷需要满足以下条件.

1) 缺陷报告中包含触发该缺陷的公式以及相应的编译选项和缺陷版本.

2) 缺陷已经被修复并且缺陷报告中开发者已经指出修复版本和修复文件.

3) 缺陷可以被复现.

3.2 实验环境设置

本文的实验在一台装载有 Ubuntu 20.0.04 Linux 操作系统的计算机上运行, 计算机的处理器为 Intel Core i7-7700 CPU @ 3.20 GHz, 内存为 32 GB.

仿照之前 SMT 求解器测试工作[22,23], 本文也利用 Gcov (https://gcc.gnu.org/onlinedocs/gcc/Gcov.html) 来收集 SMT 求解器执行过程中的覆盖信息. Gcov 是 GNU Compiler Collection (https://gcc.gnu.org/) 的一个被广泛使用的测试覆盖工具.

3.3 评价指标

本文采用常用的评价指标 TOP-n、MFR 和 MAR 来评估缺陷定位方法的效果, 该评价指标也广泛用于之前的研究工作[19,20,35]. 对于有相同可疑分数的文件, 本文使用最差排名来作为最终结果.

TOP-n: 指在排序列表中前 n 位成功定位到缺陷文件. 在本文中, n 有 4 个候选值, 即 1, 5, 10 和 20. 理想情况下, 我们希望所有的缺陷都能在排名列表的第 1 个文件中找到. 因此, TOP-n 值越大表示定位效果越好.

MFR (mean first ranking): 该指标用于衡量每个缺陷的第 1 个缺陷文件所在位置的平均值, 强调了定位第 1 个

缺陷文件的效果. MFR 的值越小, 则说明定位效果越好, 开发人员可以越快定位到缺陷的位置.

MAR (mean average ranking): 该指标用于衡量每个缺陷的所有缺陷文件所在位置的平均值, 强调了定位所有缺陷文件的效果. 与 MFR 相似, MAR 的值越小越好.

3.4 实验结果与分析

为了评估基于多源频谱的 SMT 求解器缺陷定位方法 SMTLOC 的有效性, 本文研究了以下 4 个问题.

• RQ1: SMTLOC 在 SMT 求解器缺陷定位上的效果如何?

• RQ2: 本文所提出的证人公式生成方法是否有利于定位 SMT 求解器缺陷?

• RQ3: 本文所提出的文件可疑度计算方法是否有利于定位 SMT 求解器缺陷?

• RQ4: 组合权重 α 对 SMT 求解器缺陷定位的影响?

RQ1 研究 SMTLOC 在 SMT 求解器缺陷定位上的效果. 本文是第一个提出对 SMT 求解器的缺陷进行定位的工作. 而 SBFL 技术是软件工程领域的一个经典方法, 且在其他软件系统上被证明是有效的[36], 是目前最流行的基于覆盖的缺陷定位技术之一[37]. 因此, 本文也选择 SBFL 技术作为对比技术. SBFL 技术根据通过测试用例与失败测试用例的执行路径差异计算文件可疑度值, 进而定位缺陷所在文件. 研究人员已经提出各种公式计算可疑度值, 与已有工作[38]相似, 本文实验评估了 5 种流行公式在 SMT 求解器缺陷定位上的效果, 包括 Ochiai、Tarantula、Dstar、Op2 和 Barinel. 实验结果发现这些公式在 SMT 求解器缺陷定位上具有十分相似的结果, 本文使用 Ochiai 公式作为计算公式. 在本文中, 对于每一个缺陷, SBFL 使用给定的触发缺陷的公式, 同时使用开发人员提供的测试公式作为通过的测试公式. RQ2 研究证人公式生成方法的有效性. 本文提出的证人公式生成方法使用枚举算法生成证人公式. 为了探究证人公式的有效性, 本文将证人公式替换为开发人员提供的测试公式, 该变体称为 $\text{SMTLOC}_{\text{dev}}$. 为了探究枚举算法的有效性, 本文将枚举算法替换为基于随机搜索的方法和基于马尔可夫链的方法, 这两种变体分别称为 $\text{SMTLOC}_{\text{rand}}$ 和 $\text{SMTLOC}_{\text{mc}}$. 基于马尔可夫链的方法是编译器缺陷定位 DiWi[19]使用的方法. 与 DiWi[19]相同, $\text{SMTLOC}_{\text{rand}}$、$\text{SMTLOC}_{\text{mc}}$ 的终止时间都设置为 1 h, 每个实验重复运行 5 次, 以减少随机性的影响. 我们取 5 次实验数据的中位数作为 $\text{SMTLOC}_{\text{rand}}$、$\text{SMTLOC}_{\text{mc}}$ 的最终结果. RQ3 研究文件可疑度计算方法的有效性, 本文的文件可疑度计算包含基于覆盖频谱的文件可疑度计算和基于历史频谱的文件可疑度计算. 为了探究覆盖频谱和历史频谱的有效性, 本文对 SMTLOC 设计了两个变体 SMTLOC_h 和 SMTLOC_m. SMTLOC_h 指只使用历史频谱计算文件可疑度, 用来探究覆盖频谱的有效性; SMTLOC_m 指只使用覆盖频谱计算文件可疑度, 用来探究历史频谱的有效性. RQ4 研究组合权重 α 对 SMT 求解器缺陷定位的影响. SMTLOC 默认将权重 α 设置为 0.5, 以让覆盖频谱与历史频谱具有相同的权重, 但目前还不清楚 α 为 0.5 时定位效果是否达到最优. 因此, 本文将权重 α 从 0.1 改到 1.0, 步长为 0.1, 然后探究 SMTLOC 的定位效果.

RQ1: SMTLOC 在 SMT 求解器缺陷定位上的效果如何?

RQ1 通过比较 SMTLOC 和 SBFL 的效果来探究本文提出的 SMTLOC 方法的有效性. 从表 3 可以得知, SMTLOC 的效果显著优于 SBFL 的效果. 对于实验中的所有缺陷, SMTLOC 可以在 TOP-1/5/10/20 上定位到 10/28/33/40 个缺陷, 也就是大约 16.67%、46.67%、55% 和 66.67% 的缺陷的所在文件可以在前 1、5、10 和 20 位被定位到, 而 SBFL 方法只能将 1/12/18/29 个缺陷定位到 TOP-1/5/10/20 文件级别. 从 MFR 和 MAR 来看, 对于 Z3 缺陷和 CVC5 缺陷, 使用 SMTLOC 方法进行缺陷定位得到的 MFR 和 MAR 的值显著小于使用 SBFL 方法得到的 MFR 和 MAR 值. 从 MAR 进行分析, SBFL 将 Z3 和 CVC5 的缺陷文件平均排在 25.24 位和 28.77 位, 而 SMTLOC 方法可以将 Z3 和 CVC5 的缺陷文件平均排在 18.82 位和 20.11 位, 分别提升了 25.44% 和 30.10%.

回答 RQ1: 实验结果表明, 对于收集的所有缺陷, SMTLOC 方法显著优于 SBFL 的定位效果, SMTLOC 可以在 TOP-1/5/10/20 上比 SBFL 多定位到 900%/133.33%/83.33%/37.93% 的缺陷, 说明了本文提出的 SMTLOC 方法的有效性.

RQ2: 本文所提出的证人公式生成方法是否有利于定位 SMT 求解器缺陷?

RQ2 通过比较 SMTLOC 和 $\text{SMTLOC}_{\text{dev}}$ 的效果来探究 SMTLOC 所生成的证人公式的效果. 从表 3 可以得知, 对于实验中的所有缺陷, SMTLOC 可以将 10/28/33/40 个缺陷分别定位到 TOP-1/5/10/20 文件级别, 在 TOP-

1/5/10/20 上比 $SMTLOC_{dev}$ 多定位到 150%/55.56%/37.50%/17.65% 的缺陷. 这意味着 SMTLOC 可以比 $SMTLOC_{dev}$ 取得更加有效的缺陷定位结果. 从 MFR 和 MAR 指标来看, 对于实验中的所有缺陷, SMTLOC 的 MFR 和 MAR 的值小于 $SMTLOC_{dev}$ 的 MFR 和 MAR 的值. SMTLOC 的 MFR 值和 MAR 值分别是 20.25 和 20.14, 而 $SMTLOC_{dev}$ 的 MFR 值和 MAR 值分别是 24.49 和 24.16, 分别实现了 17.31% 和 16.64% 的改进.

为了探究为什么 SMTLOC 的缺陷定位效果优于 $SMTLOC_{dev}$, 本文计算了给定的触发缺陷的公式和变异生成的证人公式、开发者提供的测试公式之间的覆盖相似性. 为了在 SMTLOC 和 $SMTLOC_{dev}$ 之间进行公平的比较, 本文使用公式 (7) 来计算证人公式与触发缺陷的公式之间的覆盖相似性, 因为 DiWi[19]和 RecBi[20]也使用相同的公式来计算覆盖相似性. 相似性的定义如下:

$$Sim(a,b)=\frac{|Cov_a \cap Cov_b|}{|Cov_a \cup Cov_b|} \tag{7}$$

其中, $Sim(a, b)$ 表示公式 a 和公式 b 执行路径的相似性, Cov_a 表示公式 a 的执行路径覆盖到的语句, Cov_b 表示 b 的执行路径覆盖到的语句.

表 3 SMT 求解器缺陷定位效果比较

SMT求解器	缺陷定位技术	TOP-1	TOP-5	TOP-10	TOP-20	MFR	MAR
Z3	SBFL	0	7	11	17	25.17	25.24
	$SMTLOC_{dev}$	3	11	14	19	20.10	22.45
	$SMTLOC_{rand}$	5	15	18	20	19.50	20.25
	$SMTLOC_{mc}$	5	15	18	20	19.50	20.29
	$SMTLOC_{h}$	4	10	11	11	58.33	66.14
	$SMTLOC_{m}$	3	8	13	19	21.53	22.21
	SMTLOC	**6**	**14**	**18**	**20**	**17.70**	**18.82**
CVC5	SBFL	1	5	7	12	31.70	28.77
	$SMTLOC_{dev}$	1	7	10	15	28.87	26.03
	$SMTLOC_{rand}$	0	7	14	18	25.93	23.31
	$SMTLOC_{mc}$	0	7	14	17	29.83	26.97
	$SMTLOC_{h}$	2	2	4	12	81.00	88.06
	$SMTLOC_{m}$	2	13	14	19	20.43	18.43
	SMTLOC	**4**	**14**	**15**	**20**	**22.80**	**20.11**
ALL	SBFL	1	12	18	29	28.44	26.93
	$SMTLOC_{dev}$	4	18	24	34	24.49	24.16
	$SMTLOC_{rand}$	5	22	32	38	22.72	21.74
	$SMTLOC_{mc}$	5	22	32	37	24.67	23.49
	$SMTLOC_{h}$	6	12	15	23	69.67	76.65
	$SMTLOC_{m}$	5	21	27	38	20.98	20.67
	SMTLOC	**10**	**28**	**33**	**40**	**20.25**	**20.14**

图 6 展示了这两种技术的证人公式与给定的触发缺陷的公式的覆盖相似性的比较结果. 在图 6 中, 纵轴表示覆盖相似性, 横轴中的 Dev 表示使用开发者提供的测试公式. 从图 6 中可以看出 SMTLOC 生成的证人公式与触发缺陷的公式之间的覆盖相似性明显大于开发人员提供的测试公式与触发缺陷的公式之间的覆盖相似性. 对于 Z3 求解器的缺陷, SMTLOC 生成的证人公式与触发缺陷的公式的覆盖相似性中位数是 0.88, 而 Dev 和触发缺陷的公式的覆盖相似性中位数只有 0.54. 对于 CVC5 求解器的缺陷, SMTLOC 生成的证人公式与触发缺陷的公式的覆盖相似性中位数是 0.91, 而 Dev 和触发缺陷的公式的覆盖相似性中位数只有 0.60. 这些数据说明了 SMTLOC 基于变异生成的证人公式比开发人员提供的测试公式具有与给定的触发缺陷的公式更高的相似性, 这与缺陷定位效果是一致的.

RQ2 进一步比较 SMTLOC、$SMTLOC_{rand}$ 和 $SMTLOC_{mc}$ 的效果来探究是否 SMTLOC 的枚举算法的效果优

于其他搜索方法. 表 3 中的行“$SMTLOC_{rand}$”和“$SMTLOC_{mc}$”的第 3–6 列是基于每种方法 5 次迭代结果的中位数的 TOP-*n* 的度量结果, 第 7, 8 列是 MFR 和 MAR 的度量结果. 从表 3 可以看出, 对于实验中的所有缺陷, SMTLOC 可以在 TOP-1/5 上比 $SMTLOC_{rand}$ 多定位到 100%/27.27% 的缺陷, 比 $SMTLOC_{mc}$ 多定位到 100%/27.27% 的缺陷. 对于实验中收集的 CVC5 缺陷, SMTLOC 可以将 4/14 个缺陷定位到 TOP-1/5 文件级别, 而 $SMTLOC_{rand}$ 和 $SMTLOC_{mc}$ 分别只能将 0/7 个和 0/7 个缺陷定位到 TOP-1/5 文件级别. 对于实验中收集的 Z3 缺陷, SMTLOC、$SMTLOC_{rand}$、$SMTLOC_{mc}$ 的 TOP-*n* 取得了相似的效果. 分析原因得知, 由于收集的 Z3 和 CVC5 的缺陷对应的触发缺陷的公式都是已被约简后的公式, 具有简短的特点, 对于收集到的部分缺陷, $SMTLOC_{rand}$ 和 $SMTLOC_{mc}$ 可以在 1 个小时之内搜索到所有可变异的节点及其对应的所有变异规则, 因此它们可能具有相似的效果. 从时间效率上分析, 本文所提出的证人公式生成方法耗时的地方是收集每个证人公式在 SMT 求解器中的覆盖信息, 约需 35 s. 因此, 公式所对应的 AST 节点数量越少, 变异得到的证人公式数量就越少, 耗时越少. 经统计, 本次实验收集到的触发 Z3 和 CVC5 缺陷的公式所对应的 AST 节点数量在 6–66 之间, 平均节点数为 16.67 和 18.37. 本文计算了拥有最多节点数的公式经枚举算法生成证人公式所需时间约为 28 min. 并经观察得知, Z3 和 CVC5 的 GitHub 仓库是非常活跃的, 其中被提交的大多数触发缺陷的公式都是被约简后的, 节点数目较少. 因此, 使用枚举算法可以在较短时间内对大多数公式进行变异生成新公式.

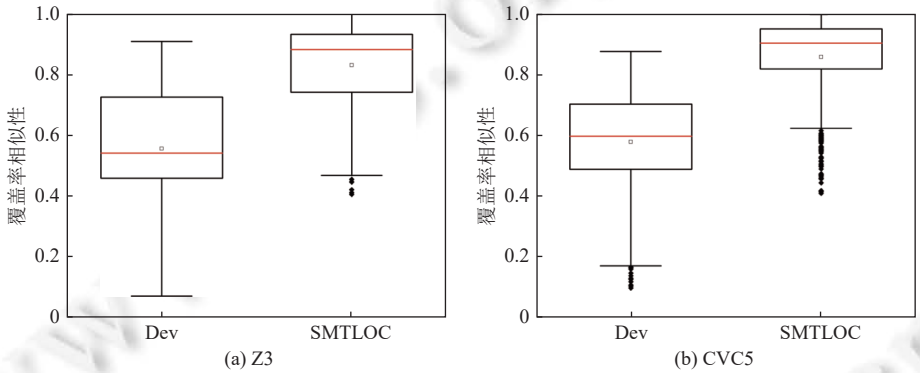

图 6 证人公式与触发缺陷的公式覆盖相似性对比图

回答 RQ2: 实验结果表明, 对于收集的所有缺陷, SMTLOC 可以在 TOP-1/5/10/20 上比 $SMTLOC_{dev}$ 多定位到 150%/55.56%/37.50%/17.65% 的缺陷, 说明了 SMTLOC 生成的证人公式比开发人员提供的测试公式在 SMT 求解器缺陷定位上具有更好的效果. 另外, 对于收集的所有 Z3 和 CVC5 缺陷, 与 $SMTLOC_{rand}$ 和 $SMTLOC_{mc}$ 相比, SMTLOC 可以在 TOP-1/5 文件级别定位更多缺陷, 说明了枚举算法的有效性. 这些实验结果说明了本文所提出的证人公式生成方法有利于定位 SMT 求解器缺陷.

RQ3: 本文所提出的文件可疑度计算方法是否有利于定位 SMT 求解器缺陷?

RQ3 通过比较 SMTLOC 与 $SMTLOC_h$ 的效果来探究覆盖频谱的有效性. 从表 3 可以得知, SMTLOC 的效果显著优于 $SMTLOC_h$ 的效果. 比如, 对于实验中的所有缺陷, SMTLOC 可以在 TOP-1/5/10/20 上比 $SMTLOC_h$ 多定位到 66.67%/133.33%/120%/73.91% 的缺陷, 这意味着 SMTLOC 比 $SMTLOC_h$ 取得更加有效的缺陷定位结果. 从 MFR 和 MAR 来看, 对于实验中的所有缺陷, SMTLOC 可以将缺陷文件定位在 20.25 MFR 和 20.14 MAR 内, 而 $SMTLOC_h$ 只可以将缺陷文件定位在 69.67 MFR 和 76.65 MAR 内, 说明了覆盖频谱的有效性.

RQ3 通过比较 SMTLOC 和 $SMTLOC_m$ 的效果来探究历史频谱的有效性. 从表 3 可以得知, SMTLOC 的效果优于 $SMTLOC_m$ 的效果. 比如, 对于实验中的所有缺陷, SMTLOC 可以在 TOP-1/5/10/20 上比 $SMTLOC_m$ 多定位到 100%/33.33%/22.22%/5.26% 的缺陷. 这意味着 SMTLOC 比 $SMTLOC_m$ 取得更加有效的缺陷定位结果. 从 Z3 缺陷的定位效果来看, SMTLOC 可以将 6/14/18/20 个缺陷定位到 TOP-1/5/10/20 文件级别, 而 $SMTLOC_m$ 只能将 3/8/13/19 个缺陷定位到 TOP-1/5/10/20 文件级别, 分别实现了 100%/75%/38.46%/5.26% 的改进.

回答 RQ3: 实验结果表明, 对于收集到的所有缺陷, SMTLOC 优于 $SMTLOC_h$ 和 $SMTLOC_m$ 的缺陷定位效果, SMTLOC 在 TOP-1/5 上比 $SMTLOC_h$ 和 $SMTLOC_m$ 分别多定位到 66.67%/133.33% 和 100%/33.33% 的缺陷, 说明了本文所提出的文件可疑度计算方法优于只使用覆盖频谱或只使用历史频谱的文件可疑度计算方法, 本文所提出的文件可疑度计算方法有利于定位 SMT 求解器缺陷.

RQ4: 组合权重 α 对定位 SMT 求解器缺陷的影响?

RQ4 使用一系列不同的权值 α (α 从 0.0 到 1.0, 步长为 0.1) 来探究权值 α 对 SMTLOC 整体效果的影响. 图 7 展示了不同权值 α 下 TOP-n 的缺陷定位效果. 横轴表示不同的权值 α, 纵轴表示将缺陷定位到 TOP-n (n 为 1, 5, 10) 文件级别的数量. 当 α 为 0 时, SMTLOC 就等同于基于历史频谱的结果; 当 α 等于 1 时, SMTLOC 等同于覆盖频谱的结果. 图 7(a) 展示了不同权值 α 下 Z3 缺陷的定位效果, 从图中可以看出, 当 α 比较小时, SMTLOC 的 TOP-1/5/10 的值在增大. 当 α 在区间 [0.3, 0.6] 时, TOP-1/5/10 的效果比较好. 而当 α 继续增大时, TOP-1/5/10 的效果开始变差. 图 7(b) 展示了不同权值 α 下 CVC5 缺陷的定位效果, 从图中可以看出, CVC5 的缺陷定位效果与 Z3 保持一致. 当 α 为 0.5 时, TOP-1/5/10 的效果达到了最好. 而当 α 小于 0.5 或大小 0.5 时, 缺陷定位效果都在下降. 综合 Z3 和 CVC5 的效果, 当 α 为 0.5 时, 缺陷定位效果达到了最好, 此时在收集的 Z3 和 CVC5 的 60 个缺陷中, 分别有 10/28/33 个缺陷被定位到了 TOP-1/5/10 文件级别.

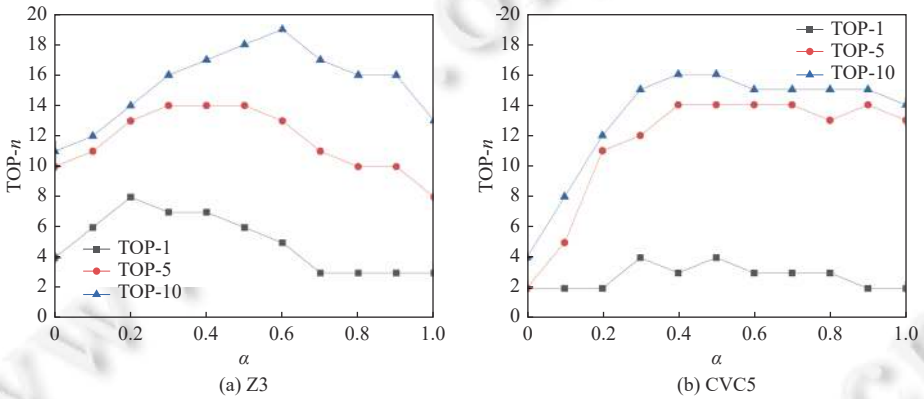

图 7 TOP-n 在不同权值 α 下的实验结果

回答 RQ4: 实验结果表明, SMTLOC 的有效性受到参数 α 的影响. 当 α 在区间 [0.0, 0.3] 和 [0.6, 1.0] 时, SMTLOC 的缺陷定位效果变化幅度较大. 当 α 在区间 [0.3, 0.6] 时, 缺陷定位效果比较稳定, 且该区间缺陷定位效果比较好. 当 α 取 0.5 时, 缺陷定位效果最优. 该实验说明了在 SMTLOC 中仅使用两个频谱中的覆盖频谱或历史频谱均不能获得最优的定位效果, 而将两个具有相似权重的频谱结合定位效果最优.

3.5 讨 论

RQ2 对文中提出的证人公式生成方法, 即枚举算法的时间效率进行了简要分析. 对于大多数触发 SMT 求解器缺陷的公式, 枚举算法可以在 30 min 内对公式进行变异生成新公式, 并得到公式的覆盖信息. 但是如果触发缺陷的公式比较复杂, 使用枚举算法对公式变异的时间开销会比较大. 未来我们考虑使用遗传算法的方式对变异算法进行优化. 具体的操作方法为: 将设计的变异规则编码为二进制 0 和 1, 0 代表使用该变异规则, 1 代表不使用该变异规则. 然后使用二进制列表构造个体, 在构造个体时, 为了保证变异后公式与原公式的覆盖相似性, 规定每个个体只随机选择一个变异规则对应的二进制位赋值为 1, 其他二进制位赋值为 0. 然后采用随机生成个体的方式对种群进行初始化. 之后搜索种群, 对于被搜索到的个体, 找到个体上二进值为 1 的编码对应的变异规则. 然后遍历触发缺陷的公式对应的抽象语法树 AST, 若 AST 上存在节点与变异规则对应, 则对节点进行变异并使用差分测试保证变异后公式的正确性, 之后计算变异后公式与原公式覆盖相似性作为适应度, 否则将该个体适应度设置为 0. 最后使用竞标赛选择个体构成子种群, 对子种群的个体进行单点交叉和二进制变异, 循环操作直到算法运行过程

中实际适应度值计算次数到达指定次数. 通过遗传算法, 最终可以得到一组变异后的新公式. 因为算法结束条件是适应度值计算次数, 采用该方法可以在公式比较复杂的情况下有效节约时间.

4 相关工作

4.1 SMT 求解器测试工作

迄今为止, 已经有很多对 SMT 求解器进行测试的工作. 其中一些工作[22,32−34]专注于找到 SMT 求解器的健壮性缺陷, 即结果错误缺陷. 为了找到 SMT 求解器的健壮性缺陷, 一些方法[22,33,34]通过构造结果已知的测试用例对 SMT 求解器进行测试. 这些方法将结果已知的测试用例输入 SMT 求解器, 如果 SMT 求解器求解的结果与预期不一致, 说明测试用例触发了 SMT 求解器的缺陷. 例如, SAE[34]使用过近似和欠近似的方法来保持变异后测试用例与原测试用例的等满足性, 然后使用变异后的测试用例测试 SMT 求解器. 另外一些工作[32,39]是通过差分测试的方法保证生成的测试用例的结果正确性. 例如, OpFuzz[32]通过使用同类型的操作符替换 SMT 求解器测试用例中的操作符, 生成变异公式, 然后用来测试 SMT 求解器, 本文的方法也借鉴了其中的思想. 还有一些工作[23,27,40]专注于找到 SMT 求解器的性能缺陷. 例如, BanditFuzz[27]使用强化学习的方法找到容易导致 SMT 求解器的性能问题的语法结构; MulStringFuzz[23]使用多目标优化的方法, 采用运行时间、代码覆盖率和测试用例复杂度作为适应度函数指导测试用例的生成, 然后查找字符串求解器的性能缺陷.

4.2 基于变异的缺陷定位技术

基于变异的缺陷定位技术[13−16]根据通过测试程序和失败测试程序在编译过程中执行路径的差异来定位缺陷所在位置. Metallaxis-FL[15]是第一个基于变异的缺陷定位技术, 其思想是对被测程序的语句进行变异可能会使测试用例由通过变为不通过, 也可能会使测试用例由不通过变为通过. 该工作通过对被测程序的语句执行不同的变异, 然后使用结果改变的测试用例数量来计算语句的可疑分数. MUSE[13]的关键思想是对被测程序中正确的语句变异会容易使得通过测试用例变为失败, 而变异错误的语句会容易使得失败测试用例变为通过. 利用该思想, 该工作对被测程序的语句进行变异, 然后分别计算原来失败的测试用例经变异后变为通过测试用例的比例、原来通过的测试用例经变异后变为失败测试用例的比例, 根据得到的比例计算被测程序语句的可疑分数. Chen 等人[19,20]提出了两种基于变异的技术对编译器 GCC 和 LLVM 进行缺陷定位, 即 DiWi 和 RecBi. 他们将编译器缺陷定位问题转换为生成不再触发相应编译器缺陷的证人测试程序的问题, 然后比较证人测试程序和给定的失败测试程序的执行路径差异来定位编译器缺陷. 他们的方法可以运用到代码量大、文件数多的大型软件系统上, 本文的方法也借鉴了该思想.

4.3 基于版本历史的缺陷定位技术

在现有的基于信息检索的技术中, 很多方法是将版本历史作为一个重要的特征组件与其他特征相结合来进行缺陷定位. Hassan 等人[25]利用代码修改的复杂性来估计文件出错的概率, 其原理是随着文件修改次数的增加, 文件存在缺陷的概率也在增加. Nagappan 等人[41]表示在软件开发的某些阶段引入修改的提交的快速增长可以用来预测未来的缺陷. 他们的直觉是在短时间内引入很多变化会使开发过程复杂化, 从而使代码出现缺陷. Sisman 等人[42]在之前工作的基础上提出两点: 在短期内被多次修改的文件包含缺陷的可能性很大, 历史上包含缺陷的文件在之后版本中依然可能包含缺陷. Wang 等人[43]考虑到新发现的缺陷通常是由最新的提交而不是相距时间很长的提交引入的, 因此在他们的实验中只考虑与缺陷报告相距不超过 K 天的版本历史信息. Wen 等人[38]从版本历史中识别引入缺陷的提交, 根据语句分别被引入缺陷的提交和非引入缺陷的提交的修改次数构造历史频谱, 然后和基于频谱的缺陷定位技术结合进行缺陷定位. 而本文基于版本历史计算文件可疑度时, 是根据版本演化历史建立每个语句的历史频谱, 然后根据文件中所有语句的被修改次数计算文件可疑度, 再结合覆盖频谱进行缺陷定位.

5 总 结

本文提出了一种基于多源频谱的 SMT 求解器缺陷定位方法 SMTLOC. 首先, 对于给定的 SMT 求解器缺陷,

SMTLOC 提出枚举算法对触发缺陷的公式进行变异, 从而生成一组与触发缺陷的公式执行路径相似的证人公式. 然后, SMTLOC 提出一种文件可疑度计算方法, 融合了基于覆盖频谱的文件可疑度和基于历史频谱的文件可疑度, 对 SMT 求解器可疑文件进行排名. 其中由覆盖频谱计算文件可疑度指的是根据证人公式与触发缺陷的公式的执行路径差异计算文件可疑度, 由历史频谱计算文件可疑度指的是根据 SMT 求解器源码信息计算文件可疑度. 基于 30 个 Z3 求解器缺陷和 30 个 CVC5 求解器缺陷的评估结果表明, SMTLOC 可以成功将 16.67% 和 46.67% 的缺陷分别定位到 TOP-1 和 TOP-5 文件级别, 明显优于传统的 SBFL 技术.

References:

[1] Detlefs D, Nelson G, Saxe JB. Simplify: A theorem prover for program checking. Journal of the ACM, 2005, 52(3): 365–473. [doi: 10.1145/1066100.1066102]

[2] Cadar C, Dunbar D, Engler DR. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. of the 8th USENIX Conf. on Operating Systems Design and Implementation. San Diego: USENIX Association, 2008. 209–224.

[3] Godefroid P, Klarlund N, Sen K. DART: Directed automated random testing. In: Proc. of the 2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation. Chicago: ACM, 2005. 213–223. [doi: 10.1145/1065010.1065036]

[4] Bhargavan K, Bond B, Delignat-Lavaud A, Fournet C, Hawblitzel C, Hritcu C, Ishtiaq S, Kohlweiss M, Leino R, Lorch J, Maillard K, Pang JY, Parno B, Protzenko J, Ramananandro T, Rane A, Rastogi A, Swamy N, Thompson L, Wang P, Zanella-Béguelin S, Zinzindohoué JK. Everest: Towards a verified, drop-in replacement of HTTPS. In: Proc. of the 2nd Summit on Advances in Programming Languages. Asilomar: Schloss Dagstuhl, Leibniz-Zentrum für Informatik, 2017. 1–12.

[5] 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. Austin: IEEE, 2016. 691–701. [doi: 10.1145/2884781.2884807]

[6] Wiels V, Delmas R, Doose D, Garoche PL, Cazin J, Durrieu G. Formal verification of critical aerospace software. Aerospace Lab, 2012(4): 1–8.

[7] de Moura L, Bjørner N. Z3: An efficient SMT solver. In: Proc. of the 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Budapest: Springer, 2008. 337–340. [doi: 10.1007/978-3-540-78800-3_24]

[8] Wang Z, Fan XY, Zou YG, Chen X. Genetic algorithm based multiple faults localization technique. Ruan Jian Xue Bao/Journal of Software, 2016, 27(4): 879−900 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4970.htm [doi: 10.13328/j.cnki.jos.004970]

[9] DiGiuseppe N, Jones JA. On the influence of multiple faults on coverage-based fault localization. In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. Toronto: ACM, 2011. 210–220. [doi: 10.1145/2001420.2001446]

[10] Liblit B, Naik M, Zheng AX, Aiken A, Jordan MI. Scalable statistical bug isolation. ACM SIGPLAN Notices, 2005, 40(6): 15–26. [doi: 10.1145/1064978.1065014]

[11] Santelices R, Jones JA, Yu YB, Harrold MJ. Lightweight fault-localization using multiple coverage types. In: Proc. of the 31st IEEE Int'l Conf. on Software Engineering. Vancouver: IEEE, 2009. 56–66. [doi: 10.1109/ICSE.2009.5070508]

[12] Xuan JF, Monperrus M. Test case purification for improving fault localization. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Hong Kong: ACM, 2014. 52–63. [doi: 10.1145/2635868.2635906]

[13] Moon S, Kim Y, Kim M, Yoo S. Ask the mutants: Mutating faulty programs for fault localization. In: Proc. of the 7th IEEE Int'l Conf. on Software Testing, Verification and Validation. Cleveland: IEEE, 2014. 153–162. [doi: 10.1109/ICST.2014.28]

[14] Papadakis M, Le Traon Y. Using mutants to locate "unknown" faults. In: Proc. of the 5th IEEE Int'l Conf. on Software Testing, Verification and Validation. Montreal: IEEE, 2012. 691–700. [doi: 10.1109/ICST.2012.159]

[15] Papadakis M, Le Traon Y. Metallaxis-FL: Mutation-based fault localization. Software Testing, Verification and Reliability, 2015, 25(5–7): 605–628. [doi: 10.1002/stvr.1509]

[16] He T, Wang XM, Zhou XC, Li WJ, Zhang ZY, Cheung SC. A software fault localization technique based on program mutations. Chinese Journal of Computers, 2013, 36(11): 2236–2244 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.02236]

[17] Wen WZ, Li BX, Sun XB, Liu CC. Technique of software fault localization based on hierarchical slicing spectrum. Ruan Jian Xue Bao/Journal of Software, 2013, 24(5): 977−992 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4342.htm [doi: 10.3724/SP.J.1001.2013.04342]

[18] Barbosa H, Barrett C, Brain M, Kremer G, Lachnitt H, Mann M, Mohamed A, Mohamed M, Niemetz A, Nötzli A, Ozdemir A, Preiner M, Reynolds A, Sheng Y, Tinelli C, Zohar Y. cvc5: A versatile and industrial-strength SMT solver. In: Proc. of the 28th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Munich: Springer, 2022. 415–442. [doi: 10.1007/978-3-030-99524-9_24]

[19] Chen JJ, Han JQ, Sun PY, Zhang LM, Hao D, Zhang L. Compiler bug isolation via effective witness test program generation. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 223–234. [doi: 10.1145/3338906.3338957]

[20] Chen JJ, Ma HY, Zhang LM. Enhanced compiler bug isolation via memoized search. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2020. 78–89.

[21] Abreu R, Zoeteweij P, Golsteijn R, van Gemund AJC. A practical evaluation of spectrum-based fault localization. Journal of Systems and Software, 2009, 82(11): 1780–1792. [doi: 10.1016/j.jss.2009.06.035]

[22] Winterer D, Zhang CY, Su ZD. Validating SMT solvers via semantic fusion. In: Proc. of the 41st ACM SIGPLAN Conf. on Programming Language Design and Implementation. London: ACM, 2020. 718–730. [doi: 10.1145/3385412.3385985]

[23] Ren ZL, Fan XF, Li XC, Zhou ZD, Jiang H. Multi-objective evolutionary algorithm for string SMT solver testing. In: Proc. of the 8th Int'l Conf. on Dependable Systems and Their Applications (DSA). Yinchuan: IEEE, 2021. 102–113. [doi: 10.1109/DSA52907.2021.00019]

[24] Guo ZQ, Zhou HC, Liu SR, Li YH, Chen L, Zhou YM, Xu BW. Information retrieval based bug localization: Research problem, progress, and challenges. Ruan Jian Xue Bao/Journal of Software, 2020, 31(9): 2826−2854 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6087.htm [doi: 10.13328/j.cnki.jos.006087]

[25] Hassan AE. Predicting faults using the complexity of code changes. In: Proc. of the 31st IEEE Int'l Conf. on Software Engineering. Vancouver: IEEE, 2009. 78–88. [doi: 10.1109/ICSE.2009.5070510]

[26] Servant F, Jones JA. History slicing: Assisting code-evolution tasks. In: Proc. of the 20th ACM SIGSOFT Int. Symp. on the Foundations of Software Engineering. North Carolina: ACM, 2012. 43. [doi: 10.1145/2393596.2393646]

[27] Scott J, Mora F, Ganesh V. BanditFuzz: A reinforcement-learning based performance fuzzer for SMT solvers. In: Proc. of the 12th Int'l Conf. on Software Verification. Los Angeles: Springer, 2020. 68–86. [doi: 10.1007/978-3-030-63618-0_5]

[28] Abreu R, Zoeteweij P, van Gemund AJC. On the accuracy of spectrum-based fault localization. In: Testing: Academic and Industrial Conf. Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007). Windsor: IEEE, 2007. 89–98. [doi: 10.1109/TAIC.PART.2007.13]

[29] de Souza HA, Chaim ML, Kon F. Spectrum-based software fault localization: A survey of techniques, advances, and challenges. arXiv:1607.04347, 2016.

[30] Levenshtein VI. Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady, 1966, 10(8): 707–710.

[31] Kuhn HW. The Hungarian method for the assignment problem. Naval research logistics quarterly, 1955, 2(1–2): 83–97. [doi: 10.1002/nav.3800020109]

[32] Winterer D, Zhang CY, Su ZD. On the unusual effectiveness of type-aware operator mutations for testing SMT solvers. Proc. of the ACM on Programming Languages, 2020, 4(OOPSLA): 193. [doi: 10.1145/3428261]

[33] Mansur MN, Christakis M, Wüstholz V, Zhang FY. Detecting critical bugs in SMT solvers using blackbox mutational fuzzing. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 701–712. [doi: 10.1145/3368089.3409763]

[34] Yao PS, Huang HQ, Tang WS, Shi QK, Wu RX, Zhang C. Skeletal approximation enumeration for SMT solver testing. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 1141–1153. [doi: 10.1145/3468264.3468540]

[35] Le TDB, Lo D, Le Goues C, Grunske L. A learning-to-rank based fault localization approach using likely invariants. In: Proc. of the 25th Int'l Symp. on Software Testing and Analysis. Saarbrücken: ACM, 2016. 177–188. [doi: 10.1145/2931037.2931049]

[36] Pearson S, Campos J, Just R, Fraser G, Abreu R, Ernst MD, Pang D, Keller B. Evaluating and improving fault localization. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Buenos Aires: IEEE, 2017. 609–620. [doi: 10.1109/ICSE.2017.62]

[37] Lou YL, Zhu QH, Dong JH, Li X, Sun ZY, Hao D, Zhang L, Zhang LM. Boosting coverage-based fault localization via graph-based representation learning. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 664–676. [doi: 10.1145/3468264.3468580]

[38] Wen M, Chen JJ, Tian YQ, Wu RX, Hao D, Han S, Cheung SC. Historical spectrum based fault localization. IEEE Trans. on Software Engineering, 2021, 47(11): 2348–2368. [doi: 10.1109/TSE.2019.2948158]

[39] Blotsky D, Mora F, Berzish M, Zheng YH, Kabir I, Ganesh V. StringFuzz: A fuzzer for string solvers. In: Proc. of the 30th Int'l Conf. on Computer Aided Verification. Oxford: Springer, 2018. 45–51. [doi: 10.1007/978-3-319-96142-2_6]

[40] Brummayer R, Biere A. Fuzzing and delta-debugging SMT solvers. In: Proc. of the 7th Int'l Workshop on Satisfiability Modulo Theories. Montreal: ACM, 2009. 1–5. [doi: 10.1145/1670412.1670413]

[41] Nagappan N, Zeller A, Zimmermann T, Herzig K, Murphy B. Change bursts as defect predictors. In: Proc. of the 21st IEEE Int'l Symp.

on Software Reliability Engineering. San Jose: IEEE, 2010. 309–318. [doi: 10.1109/ISSRE.2010.25]

[42] Sisman B, Kak AC. Incorporating version histories in information retrieval based bug localization. In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories (MSR). Zurich: IEEE, 2012. 50–59. [doi: 10.1109/MSR.2012.6224299]

[43] Wang SW, Lo D. Version history, similar report, and structure: Putting them together for improved bug localization. In: Proc. of the 22nd Int'l Conf. on Program Comprehension. Hyderabad: ACM, 2014. 53–63. [doi: 10.1145/2597008.2597148]

附中文参考文献:

[8] 王赞, 樊向宇, 邹雨果, 陈翔. 一种基于遗传算法的多缺陷定位方法. 软件学报, 2016, 27(4): 879−900. http://www.jos.org.cn/1000-9825/4970.htm [doi: 10.13328/j.cnki.jos.004970]

[16] 贺韬, 王欣明, 周晓聪, 李文军, 张震宇, 张成志. 一种基于程序变异的软件错误定位技术. 计算机学报, 2013, 36(11): 2236–2244. [doi: 10.3724/SP.J.1016.2013.02236]

[17] 文万志, 李必信, 孙小兵, 刘翠翠. 一种基于层次切片谱的软件错误定位技术. 软件学报, 2013, 24(5): 977−992. http://www.jos.org.cn/1000-9825/4342.htm [doi: 10.3724/SP.J.1001.2013.04342]

[24] 郭肇强, 周慧聪, 刘释然, 李言辉, 陈林, 周毓明, 徐宝文. 基于信息检索的缺陷定位: 问题、进展与挑战. 软件学报, 2020, 31(9): 2826−2854. http://www.jos.org.cn/1000-9825/6087.htm [doi: 10.13328/j.cnki.jos.006087]

王笑爽(1998－), 女, 硕士生, 主要研究领域为软件测试, 缺陷定位.

江贺(1980－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为智能软件工程, 工业软件测试.

周志德(1990－), 男, 博士, CCF 专业会员, 主要研究领域为智能软件工程, 可信编译, 深度学习优化.

任志磊(1984－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为演化计算, 算法自动构造, 数据挖掘, 软件大数据分析.

李晓晨(1989－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为演化计算, 智能软件工程, 开源软件工程.