贝叶斯程序分析

张 昕[1,2],王冠成[1,2],吴宜谦[1,2],陈逸凡[1,2],李天驰[1,2],张羿凡[1,2],熊英飞[1,2*]
(1. 高可信软件技术教育部重点实验室(北京大学),北京 100871;2. 北京大学计算机学院,北京 100871)

摘 要: 程序分析在软件开发和维护中发挥着关键作用. 然而,传统基于逻辑的程序分析方法在处理现代复杂、大规模和动态特性丰富的软件系统时往往效果有限,其根源在于软件系统中的不确定性. 研究人员针对具体的程序分析问题提出了一系列新的技术,其特征是在传统逻辑分析的基础上结合概率信息来捕获软件系统中的不确定性. 通过总结和抽象这些已有工作,本文提出了贝叶斯程序分析框架,其核心思想是结合程序分析和贝叶斯统计推断,通过建模和更新关于程序的概率分布来推断有关程序行为的信息. 贝叶斯程序分析采用概率逻辑编程来同时处理概率信息和逻辑信息,用统一的方式捕获了现有的多项不同工作,也能泛化到程序缺陷定位和差异调试等非传统程序静态分析任务上. 本文给出了贝叶斯程序分析框架的定义,展示了该框架在程序分析和相关领域的应用,并展望了未来发展方向.

关键词: 程序分析;逻辑编程;概率逻辑编程;贝叶斯网络;贝叶斯推断
基金项目: 国家重点研发计划(No.2022YFB4501902);国家自然科学基金(No.62172017)
中图分类号: TP312 **文献标识码:** A **文章编号:** 0372-2112(2024)04-1155-18
电子学报 URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20230973

Bayesian Program Analysis

ZHANG Xin[1,2], WANG Guan-cheng[1,2], WU Yi-qian[1,2], CHEN Yi-fan[1,2], LI Tian-chi[1,2],
ZHANG Yi-fan[1,2], XIONG Ying-fei[1,2*]
(1. *Key Laboratory of High Confidence Software Technologies* (*Peking University*), *Ministry of Education*, *Beijing* 100871, *China*;
2. *School of Computer Science*, *Peking University*, *Beijing* 100871, *China*)

Abstract: Program analysis plays a critical role in software development and maintenance. However, traditional logic-based program analysis methods exhibit significant limitations when dealing with modern, complex, large-scale, and dynamically rich software systems. The root cause of these limitations lies in the uncertainty present in software systems. To address this issue, researchers have proposed a series of new techniques for specific program analysis problems. These techniques combine probability information with traditional logic analysis to capture the uncertainty inherent in software systems. By summarizing and abstracting existing work in this area, this paper introduces the Bayesian program analysis framework. The core idea of this framework is to integrate program analysis with Bayesian statistical inference. It does so by modeling and updating probability distributions about the program to infer information about program behavior. Bayesian program analysis employs probabilistic logic programming to simultaneously handle both probability and logic information, providing a unified approach that encompasses various existing works. It can also be generalized to non-traditional static program analysis tasks, such as program fault localization and delta debugging. This paper provides a definition of the Bayesian program analysis framework, demonstrates its applications in program analysis and related fields, and outlines future directions for development.

Key words: program analysis; logic programming; probabilistic logic programming; bayesian network; bayesian inference
Foundation Item(s): National Key Research and Development Program of China (No.2022YFB4501902); National Natural Science Foundation of China (No.62172017)

收稿日期:2023-10-19;修回日期:2024-02-09;责任编辑:宋甲英
*通讯作者:熊英飞

1 引言

程序分析是指基于程序语法,回答关于程序语义的问题的技术[1]. 程序分析是软件工程、程序设计语言等领域的基础技术,和软件系统有关的大量研究和软件系统开发都离不开程序分析技术[2~5]. 比如,编译器大量依赖程序分析:编译器需要通过程序分析检查程序是否类型正确,在执行程序优化的时候需要通过程序分析保证程序优化的前条件是否满足[6]. 类似的,测试生成工具需要利用程序分析推断出能满足覆盖的程序输入,静态缺陷查找工具需要通过程序分析判断错误条件是否满足[7],缺陷修复工具需要通过程序分析判断什么样的修改能满足被违反的规约[8]. 在程序技术中,使用最广泛的一类技术是基于抽象解释的程序分析. 基于抽象解释的程序分析通过构造合适的抽象域,使得原本不可判定或者计算量巨大的程序分析问题转变为计算量可接受的问题,从而可以迭代计算出结果. 本文主要关注基于抽象解释的程序分析. 后文中,直接用"程序分析"指代"基于抽象解释的程序分析".

传统程序分析主要基于逻辑表示[9~15]. 经过数十年的研究和发展,这种分析在取得长足发展的同时也逐渐遇到了瓶颈. 基于逻辑的表示能简洁而精确地表达分析设计者的意图,在捕捉程序语义的同时,提供严格的形式化保证. 然而,这种表达方式不够灵活,一旦程序分析被交付给最终用户,便无法改变. 这也就导致了现有程序分析无法有效表达设计阶段的不确定性,即在程序分析设计时无法预先知道的应用场景的信息. 下面举例说明这种不确定性.

(1)程序的分布:由于程序分析规则都是对程序具体语义的抽象,针对不同的代码,甚至用在不同上下文中的同样的代码,往往需要采用不同的规则才能在尽量短时间内达到最高的分析精确度. 但由于很难预先知道被分析程序的分布,导致在设计阶段很难知道应该用什么分析规则,构成了程序分布的不确定性.

(2)程序的假设:不同程序对输入空间可能有各种不同的假设,对具体的执行环境也有不同的假设. 这些假设静态分析系统很难拿到,构成了程序假设上的不确定性.

(3)用户的偏好:不同用户的要求往往不同. 假设一个程序分析两种潜在的缺陷,用户 A 希望第一种缺陷尽可能报全,对第二种缺陷无所谓;用户 B 希望第二种缺陷尽可能报全,对第一种缺陷无所谓. 这样难以预先确定的需求也构成了用户偏好的不确定性.

由于不确定性的存在,程序分析的设计者往往只能选择一个折中的设计,而这种设计在各种应用中的效果往往不能达到最优.

为了解决上述问题,研究人员提出了将概率引入到传统基于逻辑的程序分析中. Ravi 等人[16]提出了一种将概率与逻辑相结合的程序分析技术,通过结合用户的反馈,将程序分析的逻辑设计者与实际用户有效地融合在一起. Mukund 等人[17]将分析规则编码为贝叶斯网,通过计算分析结果的边缘概率,专注于解决程序分析中错误报告的排序问题,并提出了一种结合人工反馈的排序技术. 进一步地,Chen 等人[18]结合动态分析信息,提出了更为高效的报告排序技术. 在这些方法中,逻辑部分继承了传统程序分析的优势,同时引入了概率部分以处理不确定性,从而使得程序分析能够结合概率信息来捕捉不确定性以提高其实用性. 然而,这些工作仅将概率应用于特定场景,在处理一些具有不确定性的情况时存在一定的局限性. 例如,Ravi 等人仅考虑将人工反馈作为后验信息,而 Mukund 等人和 Chen 等人则仅聚焦在错误报告排序问题上. 此外,这些工作还未考虑如何在迭代过程中使用观测数据来更新概率模型. 概率与逻辑相结合的思想也在程序分析之外的领域获得了应用. 例如,Zeng 等人[19]发现,通过结合概率,可以引入语义信息来提高传统缺陷定位技术的准确性. Wang 等人[20]提出了概率差异调试技术,通过将概率和逻辑相融合,充分利用累积的反馈信息不断更新模型. 他们还设计了近似推断算法,在迭代过程中取得更好的差异调试性能.

针对已有工作只是针对特定场景,难以覆盖通用情况的问题. Zhang 等人[21]基于此提出了将逻辑与概率相融合的程序分析思想,将程序分析规则编码为马可夫逻辑网,求解在后验信息下的最可能分析结果. 但是,该框架并不完备,无法满足很多实际应用的需求. 比如,该框架限定推理结果是最可能的分析结果集合,但无法衡量分析的置信度.

在上述已有工作[16~24]的基础上,本文进一步提出了贝叶斯程序分析框架. 该框架是一个程序分析设计框架,通过实例化方法的参数,可以形成各种不同的概率和逻辑结合的程序分析方法. 贝叶斯程序分析框架的核心思想是结合程序分析和贝叶斯统计推断,通过建模和更新关于程序的概率分布来推断有关程序行为的信息. 贝叶斯程序分析框架采用概率逻辑编程来同时处理概率信息和逻辑信息,用统一的方式捕获了现有的多项不同工作,也能泛化到程序缺陷定位[19]和差异调试[20]等非传统基于抽象解释的程序分析任务上. 本文给出了贝叶斯程序分析框架的定义,展示了该框架在程序分析和相关领域的应用,并展望了未来发展方向.

2 贝叶斯程序分析的用例

本节通过三个空指针解引用分析的用例来展示贝

叶斯程序分析技术程序1中的Datalog程序表达了一个空指针解引用分析. 推导规则r_2的含义是,对任意的U和V,如果关系isNull(V)和assign(V,U)都成立,那么关系isNull(U)也成立. 将上述分析运用到图1的示意代码片段中. 首先,通过前置相关分析可以得到该代码片段对应的输入关系内容:assignNull(a)、paramPass(i,a)、paramPass(i,b)、returnPass(i,x)、returnPass(i,y)、assign(y,l)、assign(y,m)、deref(a,rep_1)、deref(x,rep_2)、deref(l,rep_3)、deref(m,rep_4). 接着,不断根据推导规则和已推导出的分析事实进行推导,直到无法推导出新的事实,具体过程如下:

(1)根据推导规则r_1和已被推导事实assignNull(a),推导出isNull(a);

(2)根据推导规则r_3和已被推导事实isNull(a)、paramPass(i,a),推导出isNull(i);

(3)根据推导规则r_4和已被推导事实isNull(i)、returnPass(i,x)、returnPass(i,y),推导出isNull(x)、isNull(y);

(4)根据推导规则r_2和已被推导事实isNull(y)、assign(y,l)、assign(y,m),推导出isNull(l)、isNull(m);

(5)根据推导规则r_5和已被推导事实deref(a,rep_1)、deref(x,rep_2)、deref(l,rep_3)、deref(m,rep_4)、isNull(a)、isNull(x)、isNull(l)、isNull(m),推导出report(rep_1)、report(rep_2)、report(rep_3)、report(rep_4);

(6)无法根据规则推导出新的分析事实,达到最小不动点状态. 上述过程中推导出的全部输出关系即为Datalog程序的运行结果.

最终,该分析报告了rep_1、rep_2、rep_3、rep_4四个错误,然而只有rep_1和rep_2是真阳性报告,rep_3和rep_4均为假阳性报告. 其原因是为了牺牲精度换取速度,该分析没有区分对一个方法的不同调用. 具体来说,该程序有两次对id函数的调用,第一次调用的参数为null,第二次调用的参数不为null. 然而根据推导规则r_4,该分析会混淆不同调用的返回,从而错误地认为y也可以为null. 由此可见,由于抽象的存在,该分析具有不确定性,并导致了假阳性报告. 而过多的假阳性报告是程序分析用户的常见痛点.

2.1　用例1:基于概率的报告排序

为了解决这个问题,让用户尽量看到真阳性报告,贝叶斯程序分析采用概率逻辑语言,通过向规则添加概率的方式来刻画规则的准确度. 比如,在上述例子中,如表1所示,通过向r_4附加0.8的概率表示其在80%的方法返回处理上都产生了正确的结果. 与此相对地,其他规则都很准确,因此向它们赋予1.0的概率. 至此,可以将上述分析表示为一个带概率的Datalog程序. 通过计算错误报告边缘概率的方式对错误报告进行排序,得到表2中每个错误报告的边缘概率及排序.

```
1   id(i) {
2     return i
3   }
4   a = null
5   b = new Class1()
6   x = id(a)
7   y = id(b)
8   l = y
9   m = y
10  a.f()   // rep1
11  x.f()   // rep2
12  l.f()   // rep3
13  m.f() // rep4
```

图1　待分析的示意代码片段

程序1　空指针解引用分析的Datalog程序

输入关系

assignNull(V):	V被赋值为一个空指针
assign(V,U):	U被赋值为v
paramPass(V,U):	方法调用时V传递为U
returnPass(V,U):	方法返回时V传递为U
deref(V,S):	S涉及对V的解引用

输出关系

isNull(V):	V可能是空指针
report(S):	S可能涉及空指针解引用

推导规则

r_1:	isNull(V)	:- assignNull(V)
r_2:	isNull(U)	:- isNull(V), assign(V, U)
r_3:	isNull(U)	:- isNull(V), paramPass(V, U)
r_4:	isNull(U)	:- isNull(V), returnPass(V, U)
r_5:	report(S)	:- isNull(V), deref(V, S)

由于rep_2、rep_3、rep_4由不精确的规则r_4间接产生,它们的边缘概率均比完全由精确规则产生的rep_1低. 经过排序,用户可以优先处理rep_1这一可信度最高的报告. 通过验证,rep_1的确是一个真实的错误报告.

表1　带概率的推导规则

r_1:	1.0:isNull(V)	:- assignNull(V)
r_2:	1.0:isNull(U)	:- isNull(V),assign(V,U)
r_3:	1.0:isNull(U)	:- isNull(V),paramPass(V,U)
r_4:	0.8:isNull(U)	:- isNull(V),returnPass(V,U)
r_5:	1.0:report(S)	:- isNull(V),deref(V,S)

表2　基于概率的报告排序结果示例

排序	报告	概率$P(\text{rep}_i)$
1	rep_1	1.0
2	rep_2	0.8
3	rep_3	0.8
4	rep_4	0.8

同时,注意到当前的概率排序并没有将真实报告rep_2和假阳性报告rep_3、rep_4区分开. 这是因为即使添加

了概率,当前分析仍然依赖基于逻辑规则定义的抽象来区分不同报告,而当前抽象的粒度并不足以区分这些报告.然而,和传统分析不同的是,贝叶斯程序分析可以非常容易地结合部署后获得的额外信息,从而获得学习与适应的能力.具体来说,这是通过将这些额外信息作为后验信息,然后计算报告的后验边缘概率达成的.下面用学习用户反馈和结合运行时信息两个用例说明这一过程.

2.2 用例2:学习用户反馈的报告排序

大多数种类的程序推理技术,包括静态分析、类型系统等,都依赖于逻辑推理来获得程序的性质.然而,基于理论上不确定性的原因以及实践上可扩展性的考虑,程序推理工具在精确推导待分析程序的性质的能力上比较受限.具体来说,当用户想通过程序推理工具最终检查得到真报告时,会同时被许多假报告所困扰.众所周知,由程序推理工具得到的报告之间是有关联的:不同的真报告经常由同一个原因所导致,而不同的假报告则常常是由于推理工具不能证明某一个中间性质产生.这就使得能够通过用户反馈来减少假报告的比例、增加真报告的比例.

用例一通过给传统程序分析的每一条推导规则附加一个概率,除了能获得一组报告以外,每一个报告都附带了一个信念分数,这个分数衡量了有多大的信念认为这个报告代表了一个真正的程序漏洞.基于此,Mukund等人[17]提出BINGO方法,可以根据从用户反馈中得到的额外信息来持续更新这些信念.通过选择需要用户人工检查的报告,以及将用户反馈的信息结合进之后的分析中,可以极大地提升程序分析工具的实际可用性.假设用户在处理了rep_1后,选择处理rep_3,并发现rep_3是假阳性报告.为了减少用户的负担,理想情况下,自动隐去或者降低相似报告的排序.在这个例子中,rep_3和rep_4都源自于分析错误的认为变量y可能为null.而在当前概率与逻辑相结合的程序分析中,可以通过计算各报告在已知rep_3=false的情况下的后验边缘概率,达到降低rep_4排序的效果.如表3所示,真实报告rep_2和假阳性报告rep_3、rep_4被区分开了.

表3 学习用户反馈的报告排序结果示例

排序	报告	概率$P(rep_i \mid rep_3=\text{false})$
1	rep_1	1.0
2	rep_2	0.8
3	rep_3	0.0
4	rep_4	0.0

BINGO选取了16个测试项目作为验证数据集,其中,4个来自于先前研究工作[25,26]的项目,4个从DaCapo数据集[27]中选取的项目,这8个都用于数据竞争分析,另外8个来自于STAMP库[28]中的污点分析项目,均通过人工标注得到真报告.用于和BINGO对比的两个方法分别是BASE-R和BASE-C.其中,BASE-R在每次迭代中从未标注的报告中等概率地随机选择一个报告提供给用户检查;BASE-C则是基于报告分类器Eugene的方法[16].在所有测试程序上,用户需要检查的报告比BASE-C和BASE-R分别平均少44.2%和58.5%.例如,数据竞争分析为ftp项目产生了522个报告,其中有75个是真阳性报告.BINGO仅仅在103轮交互中就发现了所有真报告.作为对比,BASE-C的这一轮数是368,BASE-R是520.另一个值得注意的例子是DaCapo数据集中的luindex项目,940个报告中只有2个是真报告.BINGO在14轮交互后就发现了两个真报告,而BASE-C和BASE-R分别需要101轮和587轮.然而,BINGO方法的主要问题在于处理大规模程序的效率问题,导致其在实际应用场景中的局限性,如集成进开发环境中直接和开发人员进行交互.尽管BINGO方法使用到了一些如约简约束集合、限制使用到的特征用精度换速度等优化手段,仍然无法显著提高其效率.

2.3 用例3:结合运行时信息的报告排序

近来,Chen等人[18]提出一种程序分析的报告排序方法,该方法不仅可以利用用户提供的错误反馈,还可以结合其他来源以及不同形式的反馈其中最典型的就是利用动态分析所获得的程序运行时信息.

与静态分析不同,动态分析是一种通过实际运行待分析程序来获取用户所关心的程序性质的技术.有许多工具被设计用于检测不同的运行时信息.例如,Valgrind[29]用于检查内存错误,RoadRunner[30]用于检查数据竞争错误等.动态分析之所以准确,是因为它为找到的程序性质提供了具体执行作为证据.然而,由于具体执行的次数有限,无法完全覆盖程序的全部运行空间,因此动态分析总会错失一些实际存在的程序性质.换言之,动态分析是对程序性质的一种下近似分析.

将下近似的动态分析与上近似的静态分析相结合的研究由来已久.一方面,根据传统的软件开发流程,静态程序分析通常应用于测试阶段之前.因此许多工作尝试利用静态分析的结果来引导动态分析,例如,减小动态执行的搜索空间[31]、消除不必要的插装点位[32]、生成更加高效的测试用例[33]等.另一方面,动态分析的结果也可以用于优化静态分析.Csallner和Smaragdakis[34]指出可以使用测试执行来验证分析结果的正确性.Gupta等人[35]、Naik等人[36]和Nori等人[37]提出了多种不同的方法,将测试得到的动态信息转化为分析精化策略.

这些工作大多从逻辑规则的角度将动态分析与静态分析相结合.然而,对于静态分析无法排除且动态分析无法确认的程序性质,这些基于逻辑规则的方法无

能为力. 对于这一类报告,Chen 等人[18]指出,通过结合概率与逻辑,进一步地结合动态分析信息,对这些报告进行排序,提高分析报告的可用性,即假报告被降低优先级而真报告则被用户更早检查. 而动态分析得到的运行时信息正好可以作为报告排序系统的反馈,改善报告排序系统的排序效果,并提升后续用户交互阶段的效率. 和前面的用例不同,动态分析可为分析的中间结果提供反馈,而用户反馈则主要关注最终报告. Chen 等人基于上述理论提出并实现了 Dynaboost 方法. 根据该方法,后验信息除了来源于用户的反馈,还可以来源于动态分析. 假设,用推理可能不变式的方式发现变量 y 很可能总不为 null. 为了添加这一信息,首先在分析中添加如下概率规则:

$$r_6\text{: } 0.9\text{: ! observedNull}(y)\text{: - ! isNull}(y)$$

规则 r_6 的意思是,如果动态分析发现 y 不为 null,那么有 0.9 的可能性 y 确实总不为 null. 这里,用概率刻画了动态分析的不确定性. 进一步地,可以计算在动态分析认为 y 不为 null 的条件下,各个报告的后验边缘概率,并更新报告排序,如表 4 所示.

表 4　结合运行时信息的报告排序结果示例

排序	报告	概率 $P(\text{rep}_i \mid \text{observedNull}(y))$
1	rep_1	1.0
2	rep_2	0.8
3	rep_3	0.29
4	rep_4	0.29

Dynaboost 方法在一组由 13 个 C 程序测试项目构成的数据集上以污点分析和区间分析进行验证. 结果显示,通过结合运行时信息,Dynaboost 相比于用例二中的 BINGO 方法有了显著的效果提升. 一方面,真阳性报告的初始排名得到提升,Dynaboost 排序得到的真阳性报告的平均初始排名为 160,而 BINGO 得到的平均初始排名为 251;另一方面,用户交互的效率也被改善,使用 Dynaboost 的用户平均需要 59.5 次交互来找到所有真阳性报告,而使用 BINGO 则平均需要 92.2 次交互. 尽管有显著提升,Dynaboost 仍然存在一些局限性. 一方面,在实现上,受所使用的静态分析和动态分析工具的限制,这一框架只能处理基于数据流的分析,而在静态分析领域,有许多常用的分析算法是“流非敏感”的,无法适用于 Dynaboost 方法;另一方面,从理论上看,Dynaboost 所使用的“静-动转换”和“动-静转换”需要对每一种静态分析做专门设计,缺乏通用地将静态信息与动态信息相结合的途径. 此外,Dynaboost 的“动-静转换”过程直接将运行时数据流发生的频率作为对于贝叶斯节点的观测,没有充分利用贝叶斯模型的结构性,这在理论上会导致概率模型上的贝叶斯推断不够准确.

除了用例二、三外,还有很多其他可以获得后验信息的来源. 比如,通过大数据分析,可以获得某些常见的编程范式和程序分析出错的方式. 这些信息在目前是以启发式逻辑规则的方式添加到分析中,用于去除假阳性报告. 然而,这些看似普适的规则并不总是成立的,它们在不同分析应用场景上效果各异,很多时候甚至会移除正确的错误报告. 贝叶斯程序分析可以很好地解决这一问题. 一方面,通过将这些信息以带概率的推导规则的方式添加,只会改变现有报告的排序,而不会隐去某些报告;另一方面,这些新加入的概率推导规则可以和用户反馈、程序动态信息等自动形成良好的互动,使得在分析中兼顾大数据中获取的普适知识和“小数据”中获取的个性化知识.

3　贝叶斯程序分析框架

图 2 显示了贝叶斯程序分析框架,共分为离线学习和在线推理两个部分. 离线部分构建贝叶斯程序分析主体,其中规则结构主要依赖于领域专家从专业知识中提取的或来源于经典的程序分析技术所使用的规则. 概率参数部分则通过训练获得. 参数学习引擎通过在一组已知错误报告的程序上进行训练,推理出概率部分的合适赋值,从而产生完整的贝叶斯程序分析. 在线部分中,该分析被应用在新的待分析程序上,通过结合分析中的先验信息和前端子框架提供的后验信息,产生最终的报告排序.

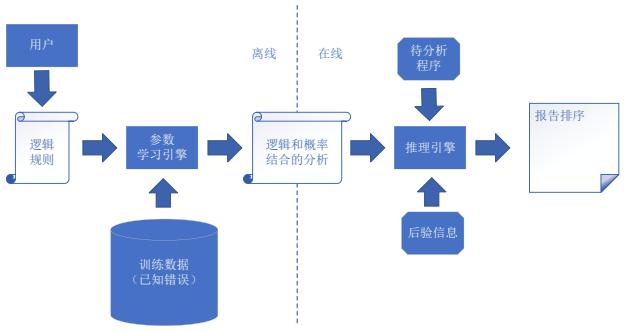


图 2　贝叶斯程序分析框架运行流程

大部分程序分析的任务都能够被转换成求解一个逻辑程序的最小不动点的形式. 逻辑语言被广泛应用于程序分析中,本文以带有概率标注的逻辑程序为实现方式,描述贝叶斯程序分析框架. 在将概率逻辑程序应用于待分析的程序时,概率逻辑程序的求解器首先通过达到最小不动点状态来生成对应的推导图和概率图模型. 随后,根据用户的查询(通常是某一事实为真的概率),推理算法用于解析查询结果. 对于未知的事实概率参数,学习算法则根据观测结果来估计这些参数值,并相应地更新模型.

3.1 贝叶斯程序分析语言

为实现贝叶斯程序分析框架,本文使用带有概率标注的逻辑语言.典型的逻辑语言,如Datalog[38],来源于数据库领域,最初是作为一种支持递归的操作语言,之后由于其良好的语言特性,被广泛应用于程序分析、计算机网络、人工智能等各个领域.在逻辑程序中,数据均用关系(类似数据库中的表)表示,而逻辑(规则)则用霍恩子句表示.给定一组输入关系,逻辑语言的求解器通过计算规则的最小不动点来得到输出关系.逻辑语言中每条规则的头部为结果,尾部(在:-之后)为条件,条件中的关系之间为合取,所有变量均带全称量词.然而,传统的逻辑语言在处理不确定性等有关问题时,存在一定的局限性.概率逻辑语言提供了自然、灵活的方式表示和推理不确定性信息.概率逻辑语言将逻辑和概率相结合,通过向规则附加概率值来表示不确定性.典型的概率逻辑语言有PRISM[39]、ProbLog[40]等.

一个概率逻辑程序主要由两部分组成:由概率事实组成的集合和概率规则组成的集合.一个概率事实,写作$p:f$,是一个带有概率p注释的事实f.一个概率规则,是一个带有概率p注释的子句,一般形式为

$$p:R_h(U_h):-R_1(U_1),R_2(U_2),\cdots,R_p(U_p)$$

其中,体部(:-之后的部分)是对事实的调用的合取式,定义了变量$U_h,U_1,U_2,\cdots,U_p$的域.

求解一个带有概率标注的逻辑程序,首先要求解其中的逻辑程序,需要累积已经实例化的约束直到达到最小不动点.给定所有输入关系的赋值I,初始化结论集合$C=I$,并将已实例化的约束初始化为输入元组的集合$\mathrm{GC}=\{\mathrm{True}\Rightarrow t|t\in I\}$.重复应用每个规则来更新$C$和GC,直到无法得到新的结论为止(达到最小不动点状态).也就是说,每当C中出现$R_1(v_1),R_2(v_2),\cdots,R_p(v_p)$,更新$C:=C\cup\{R_h(v_h)\}$,并更新$\mathrm{GC}=\mathrm{GC}\cup\{R_1(v_1)\wedge R_2(v_2)\wedge\cdots\wedge R_p(v_p)\Rightarrow R_h(v_h)\}$.在逻辑程序的最小不动点的状态上,由结论集合$C$和基于事实的子句集合GC自然地形成了一个有向图$\mathcal{G}(C\cup\mathrm{GC},E)$.其中,顶点集合为$C\cup\mathrm{GC}$.边集$E$由以下方式产生:当元组$t\in C$是子句$g\in\mathrm{GC}$的前提时,从$t$到$g$存在一条边;当$t$是子句$g$的结论时,从$g$到$t$存在一条边.

将$C\cup\mathrm{GC}$中的每个顶点看作一个随机变量,形成随机变量集合V.由此产生一个贝叶斯网络$\mathrm{BN}=(V,\mathcal{G},p)$.给定$v\in V$,用$\mathrm{Pa}(v)$表示指向$v$的边的变量集合.形式上,有

$$\mathrm{Pa}(v)=\{u\in V|u\to v\in E\}$$

随机变量v的条件概率分布是一个将$\mathrm{Pa}(v)$的赋值$x_{\mathrm{Pa}(v)}$映射到事件$v=\mathrm{True}$的条件概率的函数,将其表示为$p(v|x_{\mathrm{Pa}(v)})$.自然地,互补事件$v=\mathrm{False}$的条件概率为$p(!\ v|x_{\mathrm{Pa}(v)})=1-p(v|x_{\mathrm{Pa}(v)})$.实质上BN表示了如下的联合概率分布:

$$\Pr(x)=\prod_v p(x_v|x_{\mathrm{Pa}(v)})$$

在上式中,联合赋值x是对于每个$v\in V$的赋值x_v,而$x_{\mathrm{Pa}(v)}$是对父节点的赋值.进一步地,有变量v的边缘概率为

$$\Pr(v)=\sum_{\{x|x_v=\mathrm{True}\}}\Pr(x),\Pr(!\ v)=1-\Pr(v)$$

以及任意事件的条件概率:$\Pr(v|e)=\Pr(v\wedge e)/\Pr(e)$.

3.2 推断算法

在贝叶斯程序分析的框架下,推断引擎的主要工作就是推理出各个分析报告的边缘概率或后验概率.比如,用例一就是通过计算边缘概率的方式估计分析报告成立的概率,而用例二和用例三是在一些观察到的条件下(用例二是用户反馈,用例三是动态执行)计算分析报告成立的后验概率.

不仅如此,在学习过程中也需要反复调用推断引擎来评估候选模型参数和结构的质量.因此,需要针对程序分析领域设计快速且准确的概率推断算法.在概率统计领域,研究者们已经提出了多种不同的推断算法[41].这些推断算法从能否得出完全精确的结果来看可以分为精确推断和近似推断,而近似推断又包含采样和变分推断这两种主要路线.表5展示了主要推断算法的对比,列举出了它们的特点、优点、缺点、适用场景及主要变种.具体来看,有以下这些推断算法.

(1)模型计数法(model counting).模型计数法[42]原本是最基础的精确推断算法,通过计算满足各个条件的样本个数,就可以得到对应的条件概率.对于概率逻辑编程语言所得到的模型,计算符合条件的样本个数可以轻易地转换为SAT计数问题或SMT计数问题,从而利用求解器高效求解.

(2)变量消除法(variable elimination).这是一种精确推断算法,每次迭代基于代数规则对模型进行变形,消除一个非查询变量,直到只剩下查询变量为止.但如果概率模型中的变量存在循环依赖,无法消去,就无法使用变量消除法[43].

(3)置信传播法(belief propagation).这是一种基于消息传递的推断算法,将优化概率分布的操作转换为消息传递,在传递过程中不断优化概率分布.对于没有循环依赖的概率模型,推断结果是完全精确的[44];而对于存在循环依赖的概率模型,则有一种变种"循环置信传播"(LoopyBP)可以实现近似推断[45].

(4)马尔可夫链蒙特卡洛法(Markov-Chain Monte-Carlo,MCMC).这是使用范围最广泛的基于采样的近似推断方法,其基本思路依据马尔可夫链的性质,不断

表5　主要推断算法的对比

类型	算法名称	特点	优点	缺点	适用场景	主要变种
精确推断	模型计数法	遍历所有可能样本	结果精确;能调用已有求解器实现	时间和空间复杂度都极高;只适用于样本空间离散且有限的模型	小规模且离散的模型	—
	变量消除法	逐个消除随机变量	结果精确实现简单;理论上能够用于所有模型	重复计算多,无法复用;时间和空间复杂度受消除顺序影响很大	结构稀疏的模型,只做一次推断	—
	置信传播法	将推断操作分解为节点间的消息传递	在无环图上结果精确;计算结果可以复用	在存在环状依赖时无法精确推断	结构稀疏、无环状依赖的模型,需要多次推断时	最大乘积法
近似推断	简单随机采样	从模型联合分布中随机抽取样本来估计目标概率	实现简单	结果不精确;在联合分布复杂时效率极低	分布简单的连续模型	重要性采样
	马尔可夫链蒙特卡洛法	使用马尔可夫链来抽取样本,逼近目标概率的分布	近似精度高;适用范围广	部分情况下由于样本本身存在自相关性或难以判断马尔可夫链何时达到平衡导致不精确、高维情形下采样效率低	分布复杂的连续模型,对推断精度要求高	Metropolis-Hastings算法、Gibbs采样法
	变分推断	使用简单分布的组合近似目标概率的分布	计算速度快;适用范围广	结果存在偏差、精度受近似分布族的选取影响大	分布复杂的连续模型,对推断速度要求高	黑盒变分推断
	环状信念传播	通过消息传递不断优化目标分布(本质上也是一种变分推断)	近似精度高;可用于环状依赖情形	在结构稠密时效率低	结构稀疏、有环状依赖的模型	—

在采样过程中修正后验分布,使得每次采样的分布接近观测[46,47]. 几乎所有的概率编程语言都支持MCMC算法[48].

(5)黑盒变分推断(Black-Box Variational Inference, BBVI). 变分推断的基本思路是当模型的原分布特别复杂时,可以使用一组简单的分布来近似原分布,而BBVI利用深度神经网络来构建近似分布的模型,使得各种分布均可以使用变分推断来进行近似[49].

这些推断算法有着各自不同的适用范围,而贝叶斯程序分析框架下随着被分析程序的不同和分析问题的不同,也可能会产生不一样的概率模型,需要选择合适的求解算法来进行概率推断.

但是,程序分析的独特性也为概率推断算法带来了挑战. 首先,和大多数概率统计应用场景(比如人工智能领域)不同,为了保证分析结果的健壮性,在逻辑与概率结合的分析中,大量存在不能违背(概率为1)的硬规则(hard rules),而常见的近似推断算法由于近似结果无法保证遵循这些硬规则,因此难以应用于贝叶斯程序分析框架中. 其次,贝叶斯程序分析往往会产生非常庞大的概率模型,一般概率编程所处理的模型只包含几十个随机变量,而一个Java指针分析往往会涉及几千个甚至更多的抽象对象,对应高达10^{35}数量级的约束条件[50,51].

与此同时,程序分析的一些特性也为设计领域特化的高效推断算法带来了机会. 比如程序分析具有稀疏性和局部性,只有少部分的程序状态在程序分析中可达,且只需要一部分代码信息与分析结果相关,因此可以尝试使用逻辑制导的推断算法,只考虑逻辑规则部分所涵盖的约束条件. 再比如,程序分析由于利用程序本身的模块化特性也都带有一定的模块性,因此可以利用分块推断的技巧,将程序分析的概率模型分为多个可独立计算的小模型分别求解,从而实现组合推断.

3.3　学习算法

应用贝叶斯程序分析需要设置一系列先验参数,比如逻辑推断规则成立的概率. 直接设置这些概率值往往很困难,这时就可以用学习算法来从数据中学习出这些参数的值. 更进一步,当部分推断规则不能事先确定时,还可以用结构学习算法来学习推断规则当概率模型的参数和结构不确定时,还需要学习算法来从所有候选模型中选取最优的模型,使得目前观察到的数据的概率最大化. 学习算法可以分为参数学习和结构学习两类[41,52]. 图3展示了主要学习算法的分类,具体如下.

参数学习是指在模型结构确定的情况下选取模型上的参数. 在贝叶斯程序分析中,在逻辑规则的概率尚不确定时,需要学习引擎来为这些规则赋予概率. 这可以描述为这样的数学问题:给定一组已知错误报告的

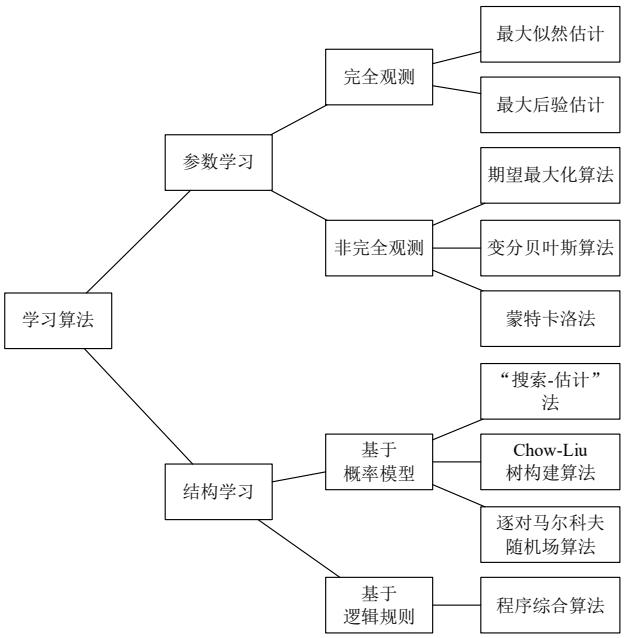


图3 主要学习算法分类

程序,和贝叶斯程序分析生成的概率模型,求出一组模型中的参数,使得这些已知报告在模型中的概率最大化.

常见的参数学习算法可以根据“是否所有的随机变量都可被观测到”分为两类. 当随机变量可以被完全观测时,可以很方便地用最大似然估计或最大后验估计来确定各个参数. 但在贝叶斯程序分析中,由于分析规则中往往会计算许多辅助信息,无法从训练数据中获取,所以必须考虑无法完全观测时,不完整数据的参数学习算法. 这些算法包括期望最大化算法[53]、变分贝叶斯方法[54]、蒙特卡洛法[55]等.

期望最大化算法(Expectation Maximization,EM)最为常用. 这一算法交替进行期望步(E-step),即根据现有参数计算所有未知变量的期望,和最大化步(M-step),即利用这些期望进行最大似然估计来重新计算各个参数,如此迭代不断更新参数直到收敛. 不过,EM算法存在容易陷入局部最优的问题.

变分贝叶斯方法(variational Bayes)则是一种贝叶斯学习方法,将模型参数也视为随机变量,通过变分推断方法,利用简单分布来近似未知变量和参数在给定观测下的后验分布. 形式上与EM算法类似,但是期望步和最大化步分别更新的是未知变量与参数的分布. 由于用分布替代了点估计,所以在选择合适的近似分布的前提下,能够学习到全局最优的参数.

此外,也可以使用蒙特卡洛法(Monte-Carlo methods)来直接对参数以及未知变量进行采样,从而近似推断参数的后验分布. 但由于模型参数空间往往是高维连续空间,很难采样得到高后验概率的参数,因此采样法的效率非常低.

结构学习则需要学习概率模型中各节点之间的依赖关系. 通用的结构学习算法包括“搜索-评估”法、Chow-Liu树[56]构建算法、逐对马尔可夫随机场算法(pairwise Markov random field)[57]等. 但在贝叶斯程序分析的框架下,有一类特殊的结构学习问题,就是学习产生逻辑规则:对于给定的分析规约(如程序及其错误报告、用户的分析描述),生成一组逻辑规则,使得这组逻辑规则表示的规约是给定规约描述的最小上近似. 现有研究提出了多种对于逻辑编程语言的程序综合算法[58-61],而将概率最优作为目标加入到程序综合的任务中,将也能够实现贝叶斯程序分析中的逻辑规则学习问题.

4 贝叶斯程序分析的拓展应用场景

通过上述用例可以发现,贝叶斯程序分析框架能够很好地结合传统基于逻辑的程序分析和后验信息,如动态信息和人工反馈等. 进一步地,本节通过其在缺陷定位和差异调试两个领域中的应用,展示其良好的可扩展性.

在缺陷定位领域,传统的技术通常专注于根据测试覆盖信息来定位程序元素的缺陷,但往往无法利用程序的语义信息. 为了应对这一挑战,Zeng等人提出了SmartFL[19],这是一种结合概率和逻辑的缺陷定位技术,也是贝叶斯程序分析框架的应用实例. SmartFL在测试执行的过程中,结合观测到的证据,能够计算各个程序元素存在缺陷的后验概率,从而实现更为准确的缺陷定位结果.

在差异调试领域,关注如何简化输入以获取期望输出的子集. 传统方法通常采用静态且固定的步骤进行简化,未能充分利用迭代过程中积累的反馈信息. Wang等人提出了概率差异调试技术ProbDD[20],该技术通过设计结合先验的概率逻辑规则,并在差异调试迭代过程中充分利用反馈信息来计算后验概率并更新概率模型,也是贝叶斯程序分析框架的应用实例. 基于上述方式,ProbDD能够在每次迭代中选择更可能成为理想输出的子集,从而提高差异调试的效率和效果.

4.1 贝叶斯程序分析在程序缺陷定位中的应用

贝叶斯程序分析能应用于软件工程等领域得很多问题,其中一个重要的应用是程序缺陷定位. 程序缺陷一直是软件开发中广泛存在、难以避免的问题. 寻找缺陷位置并加以修复往往会消耗程序员大量的时间. 近二十年来,程序缺陷定位技术一直是软件工程学术界关注的重点问题. 有效的缺陷定位工具可以很好地帮助开发者找到程序中缺陷的位置,降低开发难度和调试耗时. 然而,很多现有的缺陷定位技术只是基于概率

统计粗糙地利用了程序的动态执行信息,而忽略了细粒度的程序语义信息[62~67]. 比如,基于频谱的缺陷定位(Spectrum-Based Fault Localization,SBFL)[68]只使用测试的覆盖信息进行定位. 它的核心思想是:失败测试中出现得越多的程序元素更有可能出错,而在通过测试中出现得越多的程序元素更不可能出错,因此可以通过比较在程序元素被通过测试和失败测试覆盖的数量来对程序元素进行排序. 但是,SBFL方法也有许多本质上的问题,比如SBFL的定位最小粒度是基本块级别,即对同一个基本块内,由于覆盖信息完全一致,因此不能区分其中的各个元素. 另一方面,SBFL并没有直接考虑程序的语义信息,而通过覆盖信息去抽象测试的整个执行行为,在这一过程中,会失去数据依赖和控制依赖等重要的语义信息,这些细粒度的语义信息在很多缺陷定位的场景下是必不可少的,因而导致SBFL在很多情况下并不精确.

程序的语义可以被认为是程序计算目的的表示. 按照这一观点,程序缺陷的触发可以被理解为是错误程序元素语义在程序运行中的结果. 因此根据直观的认识,程序语义信息在缺陷定位中有非常重要的意义. 比如,语句"a++"通过一个"+1"的映射修改了变量a的值,而如果该映射有误,那么变量a的赋值就会是错误的,进而导致程序缺陷. 对另一个例子"b=a<10?b+1:b-1","a<10"的求值结果会决定变量b被如何赋值,因此如果该条件是错误的,也可能会导致程序缺陷. 如上述例子所示,像数据流依赖关系和控制依赖关系这样的程序语义信息,是和程序缺陷是否被触发直接相关的,因此如果能利用这些依赖关系的信息,将对缺陷定位有很大的帮助.

程序的语义是逻辑信息,而错误定位本质上是一个概率推断过程,因此贝叶斯程序分析可以结合两者实现更准确的错误定位. Zeng等人提出了一种新型缺陷定位方法SmartFL,通过贝叶斯程序分析技术,设计了一种编码程序中数据依赖关系和控制依赖关系等语义信息的概率模型,以推断得出各个语句的正确概率,相比传统方法大幅提升了语句级准确率. SmartFL将程序状态抽象成表示语句和变量正确性的随机变量,并将通过结合静态分析和动态分析得到的程序语义信息转化为概率约束,在效果和效率之间达到了很好的平衡.

图4展示了SmartFL的执行框架. 首先,会在目标程序上进行插桩并在插桩后的程序上执行测试进行追踪,得到Java字节码级别的源文件信息以及各个测试的执行路径信息. 在收集完相应的信息之后,SmartFL会对字节码源文件进行静态分析,以得到程序的控制依赖关系. 另一方面,SmartFL会对得到的各个测试的执行路径进行解析,通过模拟Java虚拟机的栈结构模拟运行时的情况,进而发掘数据依赖关系,并将分析的得到的控制依赖和数据依赖关系一并转化为一个统一的概率图模型. 在概率图构建完毕之后,SmartFL即会采用循环置信传播算法进行概率推断,求解出各语句变量的边缘概率. 另一方面,概率图中已经包含了数据流依赖关系和控制流依赖关系,因此SmartFL会进行动态切片,对结果进行过滤. 最终将过滤后的结果按正确概率从小到大的排序,得出定位结果.

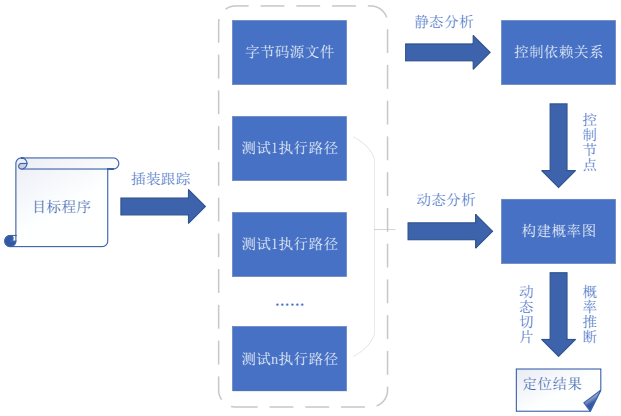


图4　SmartFL的运行流程

图5是SmartFL的一个样例程序. 该程序的错误在第3行,把条件"a<2"误写成了"a≤2". 为定位这个错误,有两个JUnit测试. 对通过的测试(pass)而言,程序中的错误条件并未影响其求值的结果,因此最终结果是正确的. 而对失败的测试(fail),错误条件影响了其求值结果,进而导致该次测试执行的分支有误,得到了错误的结果. 在这个例子中,对于像SBFL这样的缺陷定位方法,由于通过测试和失败测试的覆盖路径完全一样,因此无论其如何设计计算可疑度的公式,对被覆盖的每一行而言,得分都会是一样的,因此SBFL无法在这种情况下起作用. 其根本原因是在以本例为代表的情境中,SBFL只是粗糙地利用了程序的覆盖信息,没有对程序语义进行分析.

SmartFL和其他现有方法本质上最大的不同之处则在于,SmartFL通过贝叶斯程序分析的方式直接地使用了重要的程序语义信息. 首先,SmartFL会使用插桩技术来收集程序的动态执行信息. 从这些信息中,可以精确地提取出执行过程中的数据依赖关系. 比如,在图5的两个测试执行过程中,第3行中条件表达式"a≤2"的值依赖于第2行的函数参数"a",其运算符为"≤". 接着在第4行中,变量"a"的新赋值同样依赖第2行的"a",其运算符为"++". 另一方面,SmartFL会采用静态分析以获得程序的控制依赖关系. 其需要得知程序中各语句受哪些条件的影响,即该条件的不同取值可能会影响该语句是否被执行. 对图5中的例子而言,只有在第4行的语句"a++"会受到第3行中条件"a≤2"的影

响,而其他语句不会受到任何条件的影响.

```
1  public class CondTest{
2    public static int foo(int a){
3      if(a≤2){
4        a++;
5      }
6      return a;
7    }
8    @Test
9    void pass(){
10     assertEquals(2,foo(1));
11   }
12   @Test
13   void fail(){
14     assertEquals(2,foo(2));
15   }
16 }
```

图5 SmartFL的示例程序

在图5的例子中,SmartFL会引入输出关系valueR(T,S)去表示测试结果为T(T为pass或fail)的测试中语句S产生的值的正确性,引入输入关系paramR(S)表示语句S中测试参数的正确性,stmtR(S)表示语句S中测试参数是正确的.关于这些随机变量之间的关系,SmartFL的建模规则如下:对语句执行的输出值,其正确性取决于语句本身和语句的输入值的正确性.如果语句是否执行取决于某些条件值,则输出值的正确性也取决于这些条件值.这些随机变量之间的依赖关系来自于分析得到的程序的数据和控制依赖关系.首先,对于初始状态中的测试参数,由于这些参数都被默认为是正确的,有附带概率为1.0的事实paramR(T,s_2),即不论测试失败还是通过,语句s_2中的测试参数均是正确的.由于初始时用户对各个语句没有领域知识的先验,因此,有附带概率为0.5的事实stmtR(s_3),stmtR(s_4)和stmtR(s_6),即语句s_3,s_4和s_6正确的概率均是0.5.

SmartFL通过对程序语义的抽象设计了关于输出值正确性的概率约束.具体来说,如果语句本身、输入值和控制条件值都正确,则结果一定是正确的.这反映了缺陷定位中的一个基本原则:在正确的程序状态下,执行一条正确的语句永远不应该引入错误的程序状态.据此,会引入程序2中的概率推导规则r_1和r_2.

程序2 SmartFL中的示例

输入关系

stmtR(S): 语句S是正确的

paramR(S): 语句S中测试参数是正确的

输出关系

valueR(T, S): 测试结果为T时,语句S执行的结果可能是正确的

推导规则

r_1: 1.0: valueR(T, s_4) :- stmtR(s_4), paramR(s_2), valueR(T, s_3)

r_2: 1.0: valueR(T, s_6) :- stmtR(s_6), valueR(s_4)

接下来考虑的是错误被产生或传播的情况.SmartFL认为,如果一条语句可能产生错误的结果,则至少满足以下三个条件之一:语句本身错误、语句的输入错误或该语句不应被执行.SmartFL为了简化概率模型对这三种情况不进行区分,而是将三者统一看做“源”,并考虑“源”存在错误时表达式产生错误值的概率.此外,SmartFL认为不同类型的运算符会具有不同的性质.有些运算符对错误状态非常敏感:当“源”中存在错误时,结果很可能会是错误的.在本例中,表达式“a++”就是这样的运算.有些运算符则对错误不敏感:当“源”中存在错误时,其结果仍很可能是正确的.在本例中,“a≤2”就是这样运算.因此,当“源”中存在错误时,SmartFL为敏感的运算符的产生值设置了很高的出错概率,而为非敏感运算符设定了较低的概率.

根据以上讨论,得出程序3中r_3~r_8的概率推导关系.最终,结合程序3中所示的两项观测的证据,SmartFL根据测试结果推断每个语句是否正确的后验概率.

程序3 SmartFL中带有证据的示例Datalog程序

输入关系

observe(valueR(pass, s_6)): 在通过的测试中语句s_6中变量值是正确的

observe(!valueR(fail, s_6)): 在失败的测试中语句s_6中变量值是错误的

推导规则

r_1: 1.0: valueR(T, s_4) :- stmtR(s_4), paramR(s_2), valueR(T, s_3)

r_2: 1.0: valueR(T, s_6) :- stmtR(s_6), valueR(T, s_4)

r_3: 0.5: valueR(T, s_3) :- !stmtR(s_3)

r_4: 0.5: valueR(T, s_3) :- !paramR(s_2)

r_5: 0.99: !valueR(T, s_i) :- !stmtR(s_i), i={4, 6}

r_6: 0.99: !valueR(T, s_4) :- !paramR(s_2)

r_7: 0.99: !valueR(T, s_4) :- !valueR(T, s_3)

r_8: 0.99: !valueR(T, s_6) :- !valueR(T, s_4)

SmartFL通过程序3所示的概率逻辑程序将程序中的数据和控制依赖关系转化为概率图模型,并设置了相应的条件概率.SmartFL选择使用循环置信传播的概率推理算法推断每条语句正确的概率.对例子中的各个语句,结合给定的观测,SmartFL将成功推断出语句s_3,s_4,s_6的后验概率分别为0.29,0.73和0.87.可以看出,

第3行的语句正确概率最低,因此SmartFL在本例子中成功地定位到了错误的语句.

在实验验证方面,SmartFL和基于频谱的缺陷定位方法中的Ochiai[65]和Dstar[62],以及基于变异的缺陷定位方法中的Metallaxis[63]和Muse[69]进行了对比.使用的实验数据集为Java真实缺陷数据集Defects4J 1.0中的4个项目,共包含222个缺陷.实验采取的测量标准是语句级Top-N,即在定位结果的前N条语句中包含缺陷位置的数量,其中N取值为1,3,5,10.表6展示的是各方法的效果.

可以看到,在所有指标上,SmartFL都达到了最优,尤其在Top-1上相比其他方法有极大幅度的提升.这主要是因为SmartFL对程序语义进行了精细的概率建模.而基于频谱的方法只考虑了覆盖信息,并没有对程序语义进行建模.此外,基于变异的方法对语义的利用是较为粗糙和低效的,只是通过多次随机修改程序片段并执行测试来推测缺陷的位置.

表6　SmartFL的主要实验结果

技术	Top-1	Top-3	Top-5	Top-10
Ochiai	11(5%)	64(29%)	86(39%)	118(53%)
Dstar	12(5%)	65(29%)	86(39%)	117(53%)
Metallaxis	21(9%)	69(31%)	89(40%)	111(50%)
Muse	17(8%)	35(15%)	45(20%)	50(23%)
SmartFL	47(21%)	80(36%)	97(44%)	118(53%)

4.2　贝叶斯程序分析在差异调试中的应用

贝叶斯程序分析的另一个成功的应用是差异调试领域.差异调试问题涉及如何在保持某一特定属性的同时减小一个对象的大小.这个问题在许多应用中广泛存在,例如编译器开发、回归故障定位和缓解软件膨胀问题.差异调试问题本质上是一个搜索问题,即在庞大的求解空间中,搜索满足条件的最小解.与大部分搜索问题不同的是,由于差异调试问题要求返回的结果必须满足特定的性质,在搜索过程中需要不断调用具有一定开销的测试函数检查当前解是否满足规定的性质.

差异调试问题可以形式化地定义如下.设A是所有感兴趣对象的集合,$\phi: A \to \{F, T\}$是一个测试函数,用于提供反馈信息以确定一个对象是否具有给定属性(T)或不具有(F),$|X|$表示对象$X \in A$的大小.给定一个满足$\phi(X)=T$的对象$X \in A$,差异调试的目标是找到另一个对象$X^* \in A$,使得$|X^*|$尽可能小且$\phi(X^*)=T$,即X^*保持该属性.例如,在编译器开发过程中,给定一个揭露编译器缺陷的测试用例(一个程序),差异调试技术被用于找到一个较小规模的测试用例,在保证重现编译缺陷的同时,提高开发人员调试缺陷的效率.这里,A是程序的集合,X是一个可能很大的程序,导致编译失败,而ϕ则是用于判断是否触发相同编译缺陷的测试函数.

Wang等人注意到,许多现有方法遵循一种预定义的顺序尝试从原始对象中删除元素,并未充分利用迭代过程中积累的测试反馈.为了解决这个问题,Wang等人提出了如图6所示的概率差异调试技术ProbDD.

ProbDD中假设输入集合中每个元素之间是相互独立的,即每个元素独立地影响测试结果.设置输入关系isKEY(E)和isUnderTEST(X,E)分别表示元素E被保留在最终结果和元素E被选入待测试集合X,输出关系$\phi(X)$表示待测试集合X可能具有规定性质并通过测试函数.结合带概率标注的Datalog,可以很容易得出程序4中概率规则r_1,即当元素e必须被保留且不在待测试集合x中时,测试失败的概率是100%.

程序4　ProbDD中的示例

输入关系

isKEY(E):　元素E被保留在最终结果

isUnderTEST(X,E):　元素E被选入待测试集合X

输出关系

$\phi(X)$:　待测试集合X可能具有规定性质并通过测试函数

推导规则

r_1:　0.0: $\phi(X)$　:- isKEY(E), !isUnderTEST(X,E)

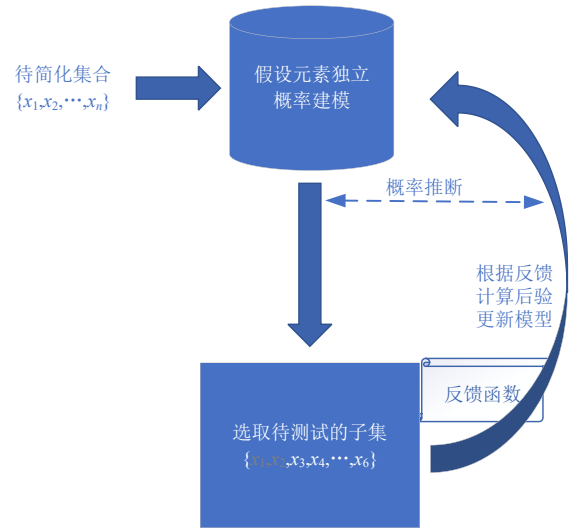


图6　ProbDD的运行流程

由此,ProbDD建立了一个概率模型来估计每个元素保留在最终结果中的概率(步骤1).在每次迭代中,ProbDD根据概率模型选择一个最大化下一次测试收益的元素子集,并测试该子集是否仍具有所需的属性(步骤2).然后,ProbDD根据测试结果计算后验概率并更新概率模型(步骤3).由于涉及利用积累的反馈信息计算后验的问题是加权模型计数问题,无法在多项式时

间内解决. ProbDD 提出利用最近一次反馈信息,近似地进行模型推断.

图 7 所示为包含 8 行语句的示例程序. 假设函数 type()存在错误,任何对它的有效调用都会导致相同的错误,在简化程序的同时,使得简化后的程序仍然能够揭露相同的错误. 假设期望减少的比率是 0.25,即期望最终该程序被简化为只包含两行语句的代码,根据 ProbDD 设置先验概率的规则,有概率事实 0.1:isKEY(s_i),$i\in[1,8]$. 在本例中,s_i 表示第 i 行语句. 结合前面提到的独立性假设,ProbDD 中待测试集合 X 通过测试的概率为 $\Pr(\phi(X))=\prod_i(1-p_i)^{1-X[i]}$,其中,$p_i$ 为第 i 个元素在最终结果的概率(初始时均为 0.1),$X[i]$ 表示第 i 个元素是否出现在待测试集合 X 中. 按照 ProbDD 中选择待测试子集的步骤,假设在第一次迭代时,选择 s_1,s_2,s_3,s_8 删除,测试集合 $x_1=\{s_4,s_5,s_6,s_7\}$,观测到证据 observe(! $\phi(x_1)$),即该次测试中测试函数返回失败. 在本例中以计算 s_1 的后验概率为例:

$$\Pr\left(\text{isKEY}(s_1)\middle|!\ \phi(x_1)\right)=\frac{\Pr\left(\text{isKEY}(s_1)\right)\times\Pr\left(!\ \phi(x_1)\middle|\text{isKEY}(s_1)\right)}{\Pr(\phi(x_1))}$$

通过将程序 4 中的逻辑程序应用于示例程序中,有 $\text{isKEY}(s_1)\wedge !\ \text{isUnderTest}(x_1,s_1)\Rightarrow\phi(x_1)$,其对应的条件概率为 $\Pr\left(!\ \phi(x_1)|\text{isKEY}(s_1)\right)=1.0$,因此,可计算出 s_1 的后验概率为 0.365 7. 类似地,经过 8 次迭代后,ProbDD 返回结果 $\{s_3,s_8\}$,由于 s_3 和 s_8 对应的概率为 1,概率收敛,算法终止. 注意到,由于涉及差异调试迭代过程中积累反馈结果的模型推断问题是加权模型计数问题,无法在多项式时间内解决. ProbDD 近似地使用最近一次反馈信息计算后验概率的优化方法极大地提高了该问题背景下模型推断的效率,最终得以提升差异调试的效率.

```
1 import tensorflow as tf
2 x = tf.constant(3.0)
3 b = 1.0
4 with tf.GradientTape() as tape:
5     tape.watch(x)
6     y=x**2
7     b = tape.gradient(y,x)
8 print(type(b))
```

图 7 ProbDD 的示例程序

Zeller 等人约 20 年前提出的 ddmin 算法[70]几乎成为了所有差异调试算法的基础,ProbDD 利用贝叶斯程序分析框架解决差异调试问题,能够在差异调试迭代过程中充分利用积累的反馈信息更新模型,解决差异调试技术中固有的缺陷,即根据固定的顺序尝试删除元素,没有充分地利用积累的测试信息以及当测试失败时的反馈. 为了比较 ProbDD 和 ddmin 的效果,在树结构和 C 程序两个应用场景中,分别利用 30 个项目进行验证. 在树结构中,采用当时树结构上的最优技术 HDD(基于 ddmin 算法,称为 d-版本 HDD),将 HDD 中的 ddmin 替换为 ProbDD(称为 p-版本 HDD). 类似地,在 C 程序中,采用当时 C 程序上的最优技术 CHISEL(基于 ddmin 算法,称为 d-版本 CHISEL),将 CHISEL 中的 ddmin 替换为 ProbDD(称为 p-版本 CHISEL). 验证结果如表 7 所示,其中,R 表示结果的几何平均标记数,S 表示对应技术每秒删除的标记数,T 表示几何平均下的处理时间(单位为 s),R_i 表示输入的几何平均标记数,↑ 表示改进,×S 表示加速比. 通过结合贝叶斯程序分析框架,通过替换技术中的 ddmin 算法,ProbDD 能够使得处理树结构最先进的差异调试技术 HDD 进一步减少 63.2% 的时间消耗,同时返回 59.5% 更小的结果;使得处理 C 语言最先进的差异调试技术 CHISEL 进一步减少 45.27% 的时间消耗,同时返回 11.51% 更小的结果. 此外,所有的 p 值都是显著的(<0.05).

表 7 ProbDD 对比 ddmin 的主要结果

场景	R_i	p-版本			d-版本			↑$_R$	p值$_R$	×S	p值$_S$	↑$_T$	p值$_T$
		R_p	S_p	T_p	R_d	S_d	T_d						
树结构	31 533	376	9	778	928	4	2 115	59.5%	0.000 0	2.3	0.000 0	63.2%	0.001 5
C 程序	64 782	8 791	31	874	9 935	17	1 597	11.5%	0.001 2	1.8	0.000 0	45.3%	0.000 0

5 相关研究工作

(1)程序分析与不确定性. 程序分析的主流方法目前多基于抽象解释[71]. 在这些分析中,一个抽象程序状态对应多个可能的具体程序(运行时)状态. 通过计算所有可达的抽象程序状态,程序分析可以推断出所有可能可达的具体程序状态. 虽然这种方法通过抽象化,将原本难以甚至无法解决的程序分析问题变得可解决,但它们传统上使用逻辑来表示抽象,无法直接处理与分析设计相关的不确定性问题. 一种流行的程序分析编写方式是使用如 Datalog[38]等的声明式逻辑编程语言[22,72,73]. 通过这样的方式,程序分析设计者只需要关心程序分析规约的设计,而不必担心算法等实现细节.

为了解决抽象单一和适用场景多样性之间的矛盾,研究人员提出了按需程序分析[22,74],其中基于反例

的抽象精化技术[75]影响最为深远.在这些分析中,分析设计者首先定义了一组不同粒度的抽象,然后通过启发式或系统搜索的方式选择最适合当前场景的抽象,以在精度和速度之间取得平衡.与前述方法不同,这些方法需要设计者事先定义所有可能的抽象,并且只能部分解决精度和速度平衡的问题,而无法处理其他由不确定性引起的问题.近年来,研究人员提出了概率抽象解释[76]来分析概率程序.在这类分析中,分析对象程序的语义本身带有不确定性(即概率),而本文关注的是分析设计中的不确定性.目前,针对不同领域中的不确定性问题,只能依赖于专家结合领域知识进行设计.针对领域问题中的不确定性建模,设计一套完备的理论是未来的一个重要研究方向.

(2)附加概率的程序分析.Zhang等人首先提出了通过在基于逻辑的程序分析中引入概率的方式,使程序分析能够从用户反馈中学习[17,77].随后,提出了基于逻辑和概率结合的程序分析的初步思想[21].尽管这一思想已在学习用户反馈用例中得到验证,但尚缺乏一个完整的通用解决方案.Nels、Benjamin和Ted等的研究团队将概率推理用于推测程序分析的规约[78~80],这可以看成是将逻辑和概率结合来解决程序分析不确定性的另一个特例.本文提出了一种概率和程序分析结合的框架,贝叶斯程序分析框架,用于解决上述应用受限的问题.除了程序分析领域,本文第4节介绍了贝叶斯程序分析框架在程序缺陷定位和差异调试领域中的应用,展示了其良好的可扩展性.

(3)逻辑与概率结合的编程语言.人工智能领域的研究人员长期以来一直在思考如何将逻辑和概率结合,以获得逻辑表达能力和处理不确定性的能力[81].这催生了概率编程[82]这一新兴研究领域,它在概率图模型[41]的基础上扩展出来.从语法角度来看,概率编程在传统编程语言中引入了概率元语;从语义角度来看,它们使用高级语言表示复杂的概率模型,每个模型对应一组概率图模型,即每个模型的采样可以看作是其中一个概率图模型的采样.这种方法使概率编程能够用更少的代码表示比传统概率图模型更复杂的概率推理问题.David Poole最早提出了在逻辑编程语言中加入概率的思想,并阐述了这种新模型和贝叶斯网之间的联系[83].此后,涌现了很多基于此思想的编程语言[84,85,39,86,40,87],然而它们要么缺乏合适的表达能力,要么缺乏针对程序分析领域的高效学习、推理算法.例如,Indepdent Choice Logic不支持规则间的循环依赖[85];Markov Logic Networks[77]和Probabilistic Soft Logic[87]则不支持最小不动点等程序分析中常用的操作.另一方面,一些语言(如Problog[40])直接在图灵完备的逻辑编程语言中引入概率[39,84,86],这些语言具备很强的表达能力,但推理和学习方面相对低效.以Problog为例,它首先将高层次的逻辑规则转化为布尔表达式,然后将推理问题转化为该布尔表达式的模型计数问题.本文提出了贝叶斯程序分析框架,将概率逻辑编程语言视为实现框架的手段.贝叶斯程序分析的效率和效果与这些语言息息相关,因此,设计更通用和高效的逻辑编程语言是未来的一个重要研究方向.

(4)纯数据驱动的软件分析.随着软件制品越来越多,过去20年一大研究趋势是应用机器学习、数据挖掘等社区研究出来的各种模型分析软件制品,形成了软件仓库挖掘、软件解析学等研究社区[88,89].这些工作大部分将程序作为文本,然后直接应用可以机器学习、数据挖掘等领域的数据驱动方法.比如,缺陷预测方法直接应用机器学习模型判断程序中是否存在缺陷.但是,这些方法完全利用数据驱动方法中的概率模型,没有基于逻辑考虑程序的语义信息,导致可能会输出不正确(unsound)的结果.相比这些工作,本文的工作主要是把概率加入到基于逻辑的程序分析规则中,仍然能保证程序分析输出正确的结果.

6　贝叶斯程序分析的挑战与机遇

贝叶斯程序分析保留了传统程序分析的优点,又提供了处理不确定性的能力.通过报告排序、结合人工反馈、结合动态信息三个用例,本文展示了贝叶斯程序分析相较于传统基于逻辑的程序分析的优势.此外,本文介绍了程序缺陷定位和差异调试领域中对贝叶斯程序分析框架的具体实现,展示了该框架具有良好的可扩展性.尽管贝叶斯程序分析在程序分析和其他领域取得了一系列进展,贝叶斯程序分析仍然存在一些挑战,限制了其可用性和应用范围.相对应地,研究人员可以从现有的挑战出发,研究针对这些挑战的解决方案,进一步提升贝叶斯程序分析的可用性和应用范围.

(1)正确性保障理论

与传统程序分析依赖抽象解释提供正确性理论基础相比,目前贝叶斯程序分析缺乏相关理论支持.比如,给定一个错误报告排序,正确报告有多大可能排在最前面?度量传统程序分析正确性使用鲁棒性和完备性这样的二元标准来评估,而贝叶斯程序分析则需要更为详细和量化的标准.同时,这些标准会和概率统计相关理论密不可分.如何建立相关理论是贝叶斯程序分析中亟须解决的问题之一.

(2)高效推断算法

推断算法在贝叶斯程序分析中起着关键作用,通常用于求解边缘概率和寻找基于当前观测结果的最优解释.此外,当概率模型的参数或结构不确定时,需要反复调用推断算法.因此,推断算法的效率问题成为影

响贝叶斯程序分析可用性和应用范围的主要因素之一. 特别是对于复杂和大规模的软件系统,构建的模型通常较为复杂,现有的推断算法很难在理想的时间内返回结果.

结合领域特定的知识,可以设计高效的推断算法,以缩短分析时间,提高贝叶斯程序分析的实际可用性. 利用近似推断方法,如变分推断或蒙特卡罗方法,来处理复杂模型的推断问题,从而在实践中实现更高效的程序分析. 基于分布式计算和并行计算技术,可以进一步提高推断算法的效率,以适应大规模软件系统的需求.

(3)应用领域扩展

虽然贝叶斯程序分析已在程序分析和其他领域取得了一系列进展,但仍然需要面对特定应用领域的挑战,限制了其可用性和应用范围. 不同领域的程序分析需求各异,需要适应不同的问题和数据特性. 进一步研究和开发面向特定应用领域的贝叶斯程序分析框架,以满足不同领域的需求. 例如,将其应用于代码补全、缺陷修复、差异调试等领域. 基于已有成功案例,为不同领域的研究人员提供指导,促进跨学科合作,进一步推动贝叶斯程序分析的应用.

如在测试用例生成领域,主要关注如何生成覆盖率高、揭错能力强的用例. 其中,被测试程序的语法语义与测试用例的覆盖率、复杂度等指标息息相关,而测试用例的揭错能力又和这些指标有关. 基于此,现有研究提出了相应的概率模型构建揭错能力强的测试用例. 然而,与已有附加概率的程序分析的工作类似,这些研究工作很难泛化到其他领域. 本文提出的框架可能可以捕捉这些工作. 在缺陷修复领域,主要关注如何自动合成参考补丁来辅助开发人员进行高效修复,其中一个重要挑战在于如何在规约不完备的情况下合成正确的补丁. 现有研究工作构建概率模型来提高缺陷修复的实用性,通过引入概率衡量缺失规约带来的不确定性. 此外,更多数据驱动的方法利用历史数据训练模型过滤错误补丁、预测正确补丁等. 一方面,针对缺失规约带来的不确定性,本文提出的框架可能可以捕捉已有工作. 另一方面,针对数据驱动的方法,本文提出的框架基于程序语义保证分析结果的正确性,一定程度上缓解数据驱动方法带来的引入不正确结果的风险. 此外,可能可以带来更多的灵活性,如在修复过程中引入反馈信息等,更好地指导修复过程.

综上所述,尽管贝叶斯程序分析在程序分析和其他领域取得了一系列进展,但仍然面临着正确性保障理论、推断算法效率以及应用领域扩展等挑战. 通过研究解决这些挑战的方法,结合机器学习、大数据和领域专业知识,贝叶斯程序分析有望进一步提高可用性和应用范围,为解决复杂软件系统分析问题提供更强大的工具和方法. 随着相关理论和技术的不断发展,贝叶斯程序分析将在更广泛的领域中发挥重要作用.

7 结束语

本文提出了贝叶斯程序分析框架,其核心思想是结合程序分析和贝叶斯统计推断,通过建模和更新关于程序的概率分布来推断有关程序行为的信息,既继承了传统程序分析的长处,又赋予了分析过程对不确定性的处理能力. 贝叶斯程序分析采用概率逻辑编程来同时处理概率信息和逻辑信息,用统一的方式捕获了现有的多项不同工作,也能泛化到程序缺陷定位和差异调试等非传统程序静态分析任务上.

当前,贝叶斯程序分析还面临一系列挑战,比如正确性保障理论、推断算法效率等. 通过深入研究正确性保障理论以及设计出表达能力强且高效的概率逻辑语言等方法,贝叶斯程序分析有望进一步提高其可用性和适用性,为应对复杂软件系统分析问题提供更为强大的理论支撑和技术依靠.

参考文献

[1] JOHNSON-FREYD P A. Introduction to Static Analysis [R]. Livermore: Sandia National Lab. (SNL-CA), 2019.

[2] 张健, 张超, 玄跻峰, 等. 程序分析研究进展[J]. 软件学报, 2019, 30(1): 80-109.
ZHANG J, ZHANG C, XUAN J F, et al. Recent progress in program analysis[J]. Journal of Software, 2019, 30(1): 80-109. (in Chinese)

[3] 陆申明, 左志强, 王林章. 静态程序分析并行化研究进展[J]. 软件学报, 2020, 31(5): 1243-1254.
LU S M, ZUO Z Q, WANG L Z. Progress in parallelization of static program analysis[J]. Journal of Software, 2020, 31(5): 1243-1254. (in Chinese)

[4] 戚晓芳, 徐宝文, 周晓宇. 一种基于程序可达图的并发程序依赖性分析方法[J]. 电子学报, 2007, 35(2): 287-291.
QI X F, XU B W, ZHOU X Y. An approach to analyzing dependence of concurrent programs based on program reachability graphs[J]. Acta Electronica Sinica, 2007, 35 (2): 287-291. (in Chinese)

[5] YAO P S, SHI Q K, HUANG H Q, et al. Program analysis via efficient symbolic abstraction[J]. Proceedings of the ACM on Programming Languages, 5(OOPSLA): 118.

[6] LATTNER C, ADVE V. LLVM: A compilation framework for lifelong program analysis & transformation[C]// International Symposium on Code Generation and Optimization. Piscataway: IEEE, 2004: 75-86.

[7] 张大林, 金大海, 宫云战, 等. 基于缺陷关联的静态分析优化[J]. 软件学报, 2014, 25(2): 386-399.

ZHANG D L, JIN D H, GONG Y Z, et al. Optimizing static analysis based on defect correlations[J]. Journal of Software, 2014, 25(2): 386-399. (in Chinese)

[8] MA W W, CHEN L, ZHANG X Y, et al. How do developers fix cross-project correlated bugs? A case study on the GitHub scientific python ecosystem[C]//Proceedings of the 39th International Conference on Software Engineering. Piscataway: IEEE, 2017: 381-392.

[9] REPS T, HORWITZ S, SAGIV M. Precise interprocedural dataflow analysis via graph reachability[C]//Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM, 1995: 49-61.

[10] COUSOT P, COUSOT R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints[C]//Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages. New York: ACM, 1977: 238-252.

[11] KAM J B, ULLMAN J D. Monotone data flow analysis frameworks[J]. Acta Informatica, 1977, 7(3): 305-317.

[12] CLARKE E M, HENZINGER T A, VEITH H. Introduction to model checking[M]//Handbook of Model Checking. Cham: Springer, 2018: 1-26.

[13] SCHMIDT D A. Data flow analysis is model checking of abstract interpretations[C]//Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM, 1998: 38-48.

[14] HUANG S S, GREEN T J, LOO B T. Datalog and emerging applications: An interactive tutorial[C]//Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. New York: ACM, 2011: 1213-1216.

[15] SHI Q K, YAO P S, WU R X, et al. Path-sensitive sparse analysis without path conditions[C]//Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. New York: ACM, 2021: 930-943.

[16] MANGAL R, ZHANG X, NORI A V, et al. A user-guided approach to program analysis[C]//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2015: 462-473.

[17] RAGHOTHAMAN M, KULKARNI S, HEO K, et al. User-guided program reasoning using Bayesian inference [C]//Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2018: 722-735.

[18] CHEN T Y, HEO K, RAGHOTHAMAN M. Boosting static analysis accuracy with instrumented test executions [C]//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2021: 1154-1165.

[19] ZENG M H, WU Y Q, YE Z T, et al. Fault localization via efficient probabilistic modeling of program semantics [C]//2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). Piscataway: IEEE, 2022: 958-969.

[20] WANG G C, SHEN R B, CHEN J J, et al. Probabilistic delta debugging[C]//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2021: 881-892.

[21] ZHANG X, SI X J, NAIK M. Combining the logical and the probabilistic in program analysis[C]//Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. New York: ACM, 2017: 27-34.

[22] ZHANG X, MANGAL R, GRIGORE R, et al. On abstraction refinement for program analyses in datalog[C]//Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2014: 239-248.

[23] HEO K, RAGHOTHAMAN M, SI X J, et al. Continuously reasoning about programs using differential Bayesian inference[C]//Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2019: 561-575.

[24] KIM H, RAGHOTHAMAN M, HEO K. Learning probabilistic models for static analysis alarms[C]//2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). Piscataway: IEEE, 2022: 1282-1293.

[25] ZHANG X, GRIGORE R, SI X, ET AL. Effective interactive resolution of static analysis alarms[J]. Proceedings of the ACM on Programming Languages, 2017, 1(OOPSLA): 1-30.

[26] SEN K. Race directed random testing of concurrent programs[C]//Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2008: 11-21.

[27] BLACKBURN S M, GARNER R, HOFFMANN C, et al. The DaCapo benchmarks: Java benchmarking development and analysis[C]//Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications. New York: ACM, 2006: 169-190.

[28] FENG Y, ANAND S, DILLIG I, et al. Apposcopy: Semantics-based detection of Android malware through static analysis[C]//Proceedings of the 22nd ACM SIGSOFT

International Symposium on Foundations of Software Engineering. New York: ACM, 2014: 576-587.
[29] NETHERCOTE N, SEWARD J. Valgrind: A framework for heavyweight dynamic binary instrumentation[J]. ACM SIGPLAN Notices, 2007, 42(6): 89-100.
[30] FLANAGAN C, FREUND S N. The RoadRunner dynamic analysis framework for concurrent programs[C]//Proceedings of the 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. New York: ACM, 2010: 1-8.
[31] CHANG W, STREIFF B, LIN C. Efficient and extensible security enforcement using dynamic data flow analysis [C]//Proceedings of the 15th ACM Conference on Computer and Communications Security. New York: ACM, 2008: 39-50.
[32] BODÍK R, GUPTA R, SARKAR V. ABCD: Eliminating array bounds checks on demand[C]//Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation. New York: ACM, 2000: 321-333.
[33] SEN K, MARINOV D, AGHA G. CUTE[J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(5): 263-272.
[34] CSALLNER C, SMARAGDAKIS Y. Check 'n' crash: Combining static checking and testing[C]//Proceedings of 27th International Conference on Software Engineering. Piscataway: IEEE, 2005: 422-431.
[35] GUPTA A, MAJUMDAR R, RYBALCHENKO A. From tests to proofs[C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2009: 262-276.
[36] NAIK M, YANG H, CASTELNUOVO G, et al. Abstractions from tests[C]//Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM, 2012: 373-386.
[37] NORI A V, RAJAMANI S K, TETALI S, et al. The yogi project: Software property checking via static analysis and testing[C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2009: 178-181.
[38] CERI S, GOTTLOB G, TANCA L. What you always wanted to know about datalog (and never dared to ask)[J]. IEEE Transactions on Knowledge and Data Engineering, 1989, 1(1): 146-166.
[39] KWIATKOWSKA M, NORMAN G, PARKER D. PRISM: Probabilistic symbolic model checker[M]//Computer Performance Evaluation: Modelling Techniques and Tools. Berlin: Springer, 2002: 200-204.
[40] DE RAEDT L, KIMMIG A, TOIVONEN H. ProbLog: A probabilistic prolog and its application in link discovery [C]//Proceedings of the 20th International Joint Conference on Artifical Intelligence. New York: ACM, 2007: 2468-2473.
[41] KOLLER D, FRIEDMAN N. Probabilistic Graphical Models: Principles and Techniques[M]. Cambridge: MIT Press, 2009.
[42] CHAVIRA M, DARWICHE A. On probabilistic inference by weighted model counting[J]. Artificial Intelligence, 2008, 172(6/7): 772-799.
[43] CHAVIRA M, DARWICHE A. Compiling Bayesian networks using variable elimination[C]//Proceedings of the 20th International Joint Conference on Artifical Intelligence. New York: ACM, 2007: 2443-2449.
[44] YEDIDIA J S, FREEMAN W T, WEISS Y. Generalized belief propagation[C]//Proceedings of the 13th International Conference on Neural Information Processing Systems. New York: ACM, 2000: 668-674.
[45] MURPHY K, WEISS Y, JORDAN M I. Loopy belief propagation for approximate inference: An empirical study[EB/OL]. (2013-01-23) [2023-09-30]. https://arxiv.org/abs/1301.6725.
[46] ANDRIEU C, DE FREITAS N, DOUCET A, et al. An introduction to MCMC for machine learning[J]. Machine Learning, 2003, 50(1): 5-43.
[47] ANDRIEU C, THOMS J. A tutorial on adaptive MCMC [J]. Statistics and Computing, 2008, 18(4): 343-373.
[48] MEENT J W V D, PAIGE B, YANG H, et al. An introduction to probabilistic programming[EB/OL]. (2018-09-27)[2023-09-30]. https://arxiv.org/abs/1809.10756.
[49] BLUNDELL C, CORNEBISE J, KAVUKCUOGLU K, et al. Weight uncertainty in neural networks[C]//Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37. New York: ACM, 2015: 1613-1622.
[50] MANGAL R, ZHANG X, NORI A V, et al. Volt: A lazy grounding framework for solving very large MaxSAT instances[C]//International Conference on Theory and Applications of Satisfiability Testing. Cham: Springer, 2015: 299-306.
[51] SUN Q, ZHAO J J, CHEN Y T. Probabilistic points-to analysis for Java[C]//Proceedings of the 20th International Conference on Compiler construction: Part of the Joint European Conferences on Theory and Practice of Software. New York: ACM, 2011: 62-81.
[52] DARWICHE A. Modeling and Reasoning with Bayesian Networks[M]. Cambridge: Cambridge University Press, 2009.

[53] NG S K, KRISHNAN T, MCLACHLAN G J. The EM algorithm[M]//Handbook of Computational Statistics. Berlin: Springer, 2011: 139-172.

[54] FOX C W, ROBERTS S J. A tutorial on variational Bayesian inference[J]. Artificial Intelligence Review, 2012, 38(2): 85-95.

[55] KROESE D P, RUBINSTEIN R Y. Monte Carlo methods [J]. WIREs Computational Statistics, 2012, 4(1): 48-58.

[56] CHOW C, LIU C. Approximating discrete probability distributions with dependence trees[J]. IEEE Transactions on Information Theory, 1968, 14(3): 462-467.

[57] SHAH A, SHAH D, WORNELL G. On learning continuous pairwise Markov random fields[C]//Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS). SanDiego: PMLR, 2021: 1153-1161.

[58] MENDELSON J, NAIK A, RAGHOTHAMAN M, et al. GENSYNTH: Synthesizing datalog programs without language bias[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2021, 35(7): 6444-6453.

[59] SI X J, LEE W, ZHANG R, et al. Syntax-guided synthesis of datalog programs[C]//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2018: 515-527.

[60] RAGHOTHAMAN M, MENDELSON J, ZHAO D, et al. Provenance-guided synthesis of datalog programs[J]. Proceedings of the ACM on Programming Languages, 2019, 4(POPL): 62.

[61] ALBARGHOUTHI A, KOUTRIS P, NAIK M, et al. Constraint-based synthesis of datalog programs[M]//Lecture Notes in Computer Science. Cham: Springer, 2017: 689-706.

[62] WONG W E, DEBROY V, GAO R Z, et al. The DStar method for effective software fault localization[J]. IEEE Transactions on Reliability, 2014, 63(1): 290-308.

[63] PAPADAKIS M, LE TRAON Y. Metallaxis-FL: Mutation-based fault localization[J]. Software Testing, Verification & Reliability, 2015, 25(5/6/7): 605-628.

[64] JIANG J J, WANG R, XIONG Y F, et al. Combining spectrum-based fault localization and statistical debugging: An empirical study[C]//2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE, 2019: 502-514.

[65] ABREU R, ZOETEWEIJ P, VAN GEMUND A J C. On the accuracy of spectrum-based fault localization[C]// Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION. Piscataway: IEEE, 2007: 89-98.

[66] LIU Y, LI Z, WANG L, et al. Statement-oriented mutant reduction strategy for mutation based fault localization [C]//2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). Piscataway: IEEE, 2017: 126-137.

[67] LIU Y, LI Z, WANG L X, et al. Statement-oriented mutant reduction strategy for mutation based fault localization[C]//2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). Piscataway: IEEE, 2017: 126-137.

[68] JONES J A, HARROLD M J, STASKO J. Visualization of test information to assist fault localization[C]//Proceedings of the 24th International Conference on Software Engineering. Piscataway: IEEE, 2002: 467-477.

[69] MOON S, KIM Y, KIM M, et al. Ask the mutants: Mutating faulty programs for fault localization[C]//2014 IEEE Seventh International Conference on Software Testing, Verification and Validation. Piscataway: IEEE, 2014: 153-162.

[70] ZELLER A, HILDEBRANDT R. Simplifying and isolating failure-inducing input[J]. IEEE Transactions on Software Engineering, 2002, 28(2): 183-200.

[71] COUSOT P. Abstract interpretation[J]. ACM Computing Surveys, 1996, 28(2): 324-328.

[72] AIKEN A. Introduction to set constraint-based program analysis[J]. Science of Computer Programming, 1999, 35 (2/3): 79-111.

[73] BRAVENBOER M, SMARAGDAKIS Y. Strictly declarative specification of sophisticated points-to analyses[C]// Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications. New York: ACM, 2009: 243-262.

[74] GUYER S Z, LIN C. Client-driven pointer analysis[M]// Static Analysis. Berlin: Springer, 2003: 214-236.

[75] CLARKE E, GRUMBERG O, JHA S, et al. Counterexample-guided abstraction refinement[M]//Computer Aided Verification. Berlin: Springer, 2000: 154-169.

[76] COUSOT P, MONERAU M. Probabilistic abstract interpretation[C]//European Symposium on Programming. Berlin: Springer, 2012: 169-193.

[77] RICHARDSON M, DOMINGOS P. Markov logic networks[J]. Machine Learning, 2006, 62(1): 107-136.

[78] BECKMAN N E, NORI A V. Probabilistic, modular and scalable inference of typestate specifications[C]//Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2011: 211-221.

[79] KREMENEK T, TWOHEY P, BACK G, et al. From uncertainty to belief: Inferring the specification within[C]// Proceedings of the 7th Symposium on Operating Systems Design and Implementation. New York: ACM, 2006: 161-176.
[80] LIVSHITS B, NORI A V, RAJAMANI S K, et al. Merlin: Specification inference for explicit information flow problems[J]. ACM Sigplan Notices, 2009, 44(6): 75-86.
[81] RUSSELL S. Unifying logic and probability[J]. Communications of the ACM, 2015, 58(7): 88-97.
[82] GOODMAN N D. The principles and practice of probabilistic programming[J]. ACM SIGPLAN Notices, 2013, 48 (1): 399-402.
[83] POOLE D. Probabilistic Horn abduction and Bayesian networks[J]. Artificial Intelligence, 1993, 64(1): 81-129.
[84] MUGGLETON S. Stochastic logic programs[EB/OL]. [2023-09-29]. https://www.semanticscholar.org/paper/Stochastic-Logic-Programs-Muggleton/454742f723d5bdeb023d4a65ede020e881d68ae4.
[85] POOLE D. The independent choice logic and beyond [M]//Probabilistic Inductive Logic Programming: Theory and Applications. Berlin: Springer, 2008: 222-243.
[86] KERSTING K, DE RAEDT L. Basic principles of learning Bayesian logic programs[M]//Probabilistic Inductive Logic Programming. Berlin: Springer, 2008: 189-221.
[87] KIMMIG A, BACH S, BROECHELER M, et al. A short introduction to probabilistic soft logic[C]//Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications. Funner: NeurIPS, 2012: 1-4.
[88] ZHANG D, DANG Y, LOU J G, et al. Software analytics as a learning case in practice: Approaches and experiences [C]//Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering. Lawrence: ACM, 2011: 55-58.
[89] VIDONI M. A systematic process for mining software repositories: Results from a systematic literature review[J]. Information and Software Technology, 2022, 144: 106791.

作者简介

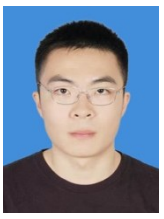
张 昕 男,1989年9月出生于湖北省武汉市. 现任北京大学新体制助理教授、研究员. 研究方向为程序设计语言和软件工程. 获得国家级青年人才项目、PLDI杰出论文奖、FSE杰出论文奖等.
E-mail: xin@pku.edu.cn

王冠成 男,1993年7月出生于江苏省连云港市. 现为北京大学计算机学院软件所博士研究生. 主要研究方向为软件测试与调试.
E-mail: guancheng.wang@pku.edu.cn

吴宜谦 男,1999年5月出生于江西省南昌市. 现为北京大学计算机学院软件所博士研究生. 主要研究方向为软件工程.
E-mail: wuyiqian@pku.edu.cn

陈逸凡 男, 1997年8月出生于江苏省南通市. 现为北京大学计算机学院软件所博士研究生. 主要研究方向为程序分析.
E-mail: yf_chen@pku.edu.cn

李天驰 男,2001年12月出生于河北省邢台市. 现为北京大学计算机学院软件研究所博士研究生. 主要研究方向为程序分析.
E-mail: litianchi@pku.edu.cn



张羿凡 男,2000年11月出生于北京市. 现为北京大学计算机学院程序设计语言研究室博士研究生. 主要研究方向为程序分析.
E-mail: 2301111981@stu.pku.edu.cn

熊英飞 男,1982年5月出生于四川省达州市. 现任北京大学新体制长聘副教授、研究员. 研究方向是程序设计语言和软件工程. 获得电子学会自然科学一等奖(排名1)、CCF-IEEE CS青年科学家奖、MODELS十年最有影响力论文奖等. 中国电子学会会员编号:E190017004M.
E-mail: xiongyf@pku.edu.cn