浙江大学

本科实验报告

课程名称：	编译原理
姓　　名：	张佳瑶
学　　院：	计算机科学与技术学院
系：	
专　　业：	软件工程
学　　号：	3170103240
指导教师：	李莹

1 前言

正确性是程序最重要的属性之一。长期以来，如何保证程序的正确性，尽可能地发现程序中各种潜在的错误，一直是计算机科学界和工业界关注的一个重要问题。随着人们对软件正确性、安全性要求的提高，随着计算机硬件性能的快速提升，相关算法及其实现技术的日益成熟，程序分析技术逐渐要求提供越来越高的精度，给出更加准确的分析结果。程序分析也在软件工程、安全、系统、网络等相关领域的实际需求驱动下不断发展。研究者们提出了很多理论和方法。本文主要介绍程序分析方面的研究。本文将介绍程序分析的基本概念、分析技术以及其实际应用等。

2 正文

2.1 程序分析简介

2.1.1 基本概念

程序分析是指对计算机程序进行自动化的处理，提供编译技术以预测可计算的一组程序在运行时动态产生的值或行为的近似集以及正确性、安全性等特性。程序分析的目的是收集程序真正在计算机上运行之前的动态属性，经典的属性包括：指针指向哪一个对象、变量是在编译时绑定还是在运行时绑定、这个程序能提高多少效率、这个变量是否在函数使用等。

程序分析的结果可以用于编译优化、提供警告信息等。一个主要的应用是让编译器避免生成冗余计算的代码。例如，通过重用可用的结果，或移动循环不变的计算出结果，或避免不必要的计算[1]。在安全性方面，分析程序指针是否为空，数组下标越界等情形。

2.1.2 程序分析的难度

程序分析的对象多种多样，可以是源代码，也可以是二进制可执行代码；可以是移动应用软件，也可以是分布式系统程序。程序设计语言具有复杂性。相比 C 程序，分析 Java 程序需要处理面向对象的结构。程序性质具有复杂性。不同的程序具有不同的特性，比如一些程序具有大量的指针，一些程序具有大量的数值计算，一些程序使用并发控制结构。针对不同性质的程序，人们关注的问题的着重点不同。比如，针对有指针的程序，人们会关注空指针、内存泄漏等问题。

程序语言、程序性质存在多样性、复杂性，且程序分析是在程序输入可用之前应用的，这显然使分析不可判定。在复杂性理论中，这被称为 Rice 定理：程序的所有非平凡属性都是不可判定的。比如，理论上，程序的终止性问题被证明为不可判定的，既不存在一种通用的方法来证明任何给定程序是否能终止。最近，Cousot[2]等人从可计算性的角度研究静态程序分析，从抽象解释的角度对 Rice 定理进行形式化地证明，得到结论：程序分析比程序验证更难；对于有效域上的程序断言，程序分析和程序验证是等效问题，而对于无限域上的程序断言，程序分析比程序验证更难。

程序分析的理论难度导致程序分析结果不精准，从而出现误报（false positive）或漏报（false negative）。误报是指报警信息提出的缺陷实际上不存在，大量的误报将导致大量的人工审查工作，带来不必要的人力、时间等资源的浪费。漏报是指程序中的某个缺陷没有被程序分析工具发现。

2.2 程序分析技术

程序分析总体可以分为静态分析和动态分析，涉及的基础理论包括抽象解释、约束解释、

自动推理等。静态分析是指不运行程序的前提下，进行程序分析。静态分析的关键技术包括数据流分析、过程间分析、符号执行等。动态分析需要通过运行程序来获取数据流信息。随着计算机技术发展，机器学习技术被用于提升各种不同的程序分析技术的性能。

2.2.1 抽象解释

抽象解释是一种对程序语义进行可靠抽象或近似的通用理论[4]。该理论为程序分析的设计和构建提供了一个通用的框架，并从理论上保证了所构建的程序分析的终止性和可靠性：所有基于上近似抽象推理得出的性质，在原程序中也必然成立。

程序设计语言制定了用该语言书写的任何程序的语义。P.Cousot 和 R.Cousot[3]指出，可以通过求解程序具体语义函数的最小不动点来研究程序的性质。如果程序不终止，程序的输入集合是无穷，这个程序的最小不动点是无法求得的；如果程序可终止，程序输入集合有限，程序的不动点需要运行该程序后求得，代价太高。因此，抽象解释理论提出，通过在抽象域上计算程序的抽象不动点来表示程序的抽象语义。在程序分析中，寻找一个抽象函数，该抽象结束函数作用于某一个特定的抽象域，考察程序的某一特定属性。抽象域的域元素可以近似程序状态集合，抽象域的域操作可以建模程序语义动作，加宽算子加速抽象域上不动点迭代的收敛速度并保证不动点迭代的终止性。抽象域不如具体域精确，规模也大大缩减，通常可以到达可计算的范围。抽象域上的最小不动点可以映射到具体域中，作为具体域中最小不动点的近似。抽象解释本质上是通过对程序语义进行不同程度的抽象，在分析精度和计算效率之间取得了权衡[5]。

目前，已经出现了数十种面向不同性质的抽象域，具有代表性的抽象域有区间抽象域、八边形抽象域、多面体抽象域等数值抽象域，还有若干开源的抽象域库，如 APRON[6]、ELINA[7]、PPL[8]等。

基于抽象解释的程序分析主要面临两方面的挑战：提高分析精度、提高可扩展性。

在提高分析精度方面：(1)使用策略迭代可以代替加宽并使用数值求解器计算精确的不动点[13]；(2)抽象加速通过将单个非迭代传递函数应用于循环头的初始状态来捕捉循环的效果，适用于任何不受限制的线性回路[14]；(3)通过加宽或变窄来加快定点迭代[15]。

在提高可扩展性方面：(1)将抽象域与 SMT 相结合来逐块分析程序[16]；(2)基于决策树等方法构造非凸抽象域，提高抽象域的析取能力。

2.2.2 数据流分析

数据流分析是静态分析中的关键技术，通过分析程序状态信息在控制流图中的传播来计算每个静态程序点（语句）在运行时可能出现的状态。数据流分析是对程序中数据的使用、定义及其之间的依赖关系等各方面的信息进行收集的过程[19]。数据流分析帮助在不实际运行程序的情况下，发现程序运行时的行为。

首先可以将一个给定的程序划分为一系列基本块，形成该程序的控制流程图。将程序作为一个整体来收集信息，并将信息分配给流图的各个基本块。通过建立和求解与信息有关的方程系统即可收集数据流信息。

从分析的精度来分类，数据流分析可以分为流不敏感、流敏感和路径敏感三种。数据流分析中最常用的分析方法是迭代分析。迭代分析是从数据流问题的初始值开始通过不断应用传输函数最终达到一个不动点。实际应用中，通常会将基本块的分析顺序进行某种排序（如逆 CFG 的逆后序），以加速算法的收敛速度。

2.2.3 符号执行

符号执行是一种广泛使用、相对精确的程序分析技术。与实际执行程序相似，符号执行技术也会执行程序。符号执行的基本思想是，用抽象的符号表示程序中变量的值，来模拟程序的执行。该方法克服了在静态测试中不能确定程序中变量的值的问题。符号执行常常在对路径敏感的程序分析中使用。用符号执行加约束求解进行程序分析的基本思想是：用 Hoare 逻辑将程序表示成{P}Q{R},其中 P 是执行程序前需要满足的条件，R 是程序执行后需要满足的条件。在程序的符号执行过程中由 P 出发，结合程序中的约束条件，可以推导出新的约束条件 $c_1 \wedge c_2 \wedge \ldots \wedge c_n$。因此有 $P \wedge c_1 \wedge c_2 \wedge \ldots \wedge c_n \rightarrow R$。可以对约束条件 $P \wedge c_1 \wedge c_2 \wedge \ldots \wedge c_n \rightarrow R$ 求解。如果有一组解满足这一约束，说明存在一组输入使运行程序的结果和规范不符。如果程序的规约正确，则程序中必定包含错误。

符号执行和约束求解方法的优点在于它可以精准地静态模拟程序的执行。由于它跟踪了变量所有可能取值，因此能够发现程序中细微的逻辑错误。但是在处理大程序时，程序执行的可能路径数目随程序尺寸的增大而呈指数级增长。在这种情况下就需要对路径进行选择，选取一定数量的路径进行分析，一般可以通过控制符号执行步数、控制循环次数、限制函数内联深度、避免递归等方法解决。放弃部分路径的分析可能导致漏报，这是符号执行的开销和分析精度的一个平衡问题。

随着 SAT/SMT 技术发展，动态符号执行出现。在实际应用中，微软公司将自己开发的二进制动态符号执行工具 SAGE 用于 Win 7 测试，发现了文件模糊测试中 1/3 的缺陷[22]。

2.2.4 动态分析

动态分析是指在指定测试用例下运行给定的程序，并分析程序运行过或结果，可以用于缺陷检测等。静态分析有不少局限性。对于程序设计语言中的动态属性（比如指针运算、动态存储分配等相关的性质），用静态分析难以奏效，而动态分析能够很好地处理这些。因此，程序动态分析技术越来越受到重视。动态分析可以分为在线和离线两种。在线动态分析是指在程序的运行过程中分析当前程序行为，离线动态分析是指在程序运行结束后分析记录下的程序运行行为。将程序运行结果和预期结果对比，可以确定程序中是否含有缺陷，但无法检

测没有反映在输出中的缺陷。通过插桩或其他监控技术分析程序运行行为，并提前定义错误的行为，查找错误，可以检测没有输出的缺陷。

2.2.5 基于机器学习的程序分析

传统的程序分析技术存在路径组合爆炸、误报率较高等问题。随着近年来计算能力提高，计算机技术发展，程序分析开始采用机器学习、统计分析等技术。

基于现有的已知结果的程序建立学习模型，学习新的可自适应的策略。由于目前约束求解的限制，符号执行很难应用于有复杂路径的程序中，如非线性约束和函数调用。为解决符号执行中的路径可达性的问题，Li[9]等人建立机器学习模型 MLB，不依赖于经典的约束求解。Kong[10]等人面向自动机验证，分析了不同的随机抽样与符号执行相结合的策略，提出了一种基于网络的、动态切换随机抽样与符号执行的算法，可以降低整体成本。Wang[11]等人展示了最佳策略可以由程序路径概率和约束求解代价确定，提出了基于马尔可夫过程的动态符号执行方法，以达到应用符号执行获取精确分析结果和应用随机测试覆盖搜索空间的平衡。针对最优策略识别的复杂性，该方法设计了一种近似最优策略的贪心算法。Xiong[12]等人提出了基于概率合成抽象框架 L2S。该框架使用四个工具：语法用于定义搜索空间和搜索步骤，约束条件用于在每个搜索步骤中删除无效的候选者，机器学习的模型用于估计每个搜索步骤中候选者的条件概率，并且使用搜索算法来找到最佳的解决方案。研究者可以基于该框架深度定制和设计相关方法。

2.3 实际应用

程序分析技术在多个领域得到了应用，包括移动应用软件、并发软件、分布式系统、二进制代码等方面。

在移动应用软件领域，研究者更多关心以安全性为核心的各种特性，常常是采用跟踪敏感数据流的动态/静态污点分析。TaintDroid[21]是一种动态污点分析技术，被应用于分析 Android 应用。TaintDroid 利用 Android 的虚拟化执行环境提供了实时分析，在应用的 Java 字节码的解释执行过程中进行动态插桩，监视敏感数据。FlowDroid 是适用于 Android 应用程序的上下文，流程，字段，对象敏感和生命周期感知的静态污点分析工具。FlowDroid 基于过程间控制流图进行静态的 Jimple 代码模拟执行，根据 Jimple 指令的语义跟踪敏感数据在潜在执行路径上的传播，从而检测分析目标应用中存在的隐私泄露等危险操作[5]。

并发软件缺陷的检测技术有静态分析、动态分析以及两者结合的混合分析。

大规模分布式系统具有较好的可扩展性和容错能力，在实际应用中受到越来越多的重视。为了提高分布式系统的可靠性，研究者对分布式系统中的各种缺陷进行了研究。CBS[23]针对 6 个开源分布式系统（Hadoop MapReduce、HDFS、Hbase、Cassandra、ZooKeeper 和 Flume）的开发和部署问题进行了全面研究，并深入研究其中 3655 个致命缺陷，详细总结了分布式

系统中出现的各种缺陷类型，包括可靠性、性能、可用性、安全性、可扩展性等方面的缺陷，并形成了一个开放的分布式系统缺陷数据集。DCatch[24]通过分析分布式系统的正确执行来预测缺陷。为了构建 DCatch，研究者对分布式云系统的各种通信和并发机制建模，建立起分布式系统中事件之间的偏序关系，静态地对这些事件进行分析，识别并发的内存访问冲突，发现可能的分布式系统并发缺陷。

3 小结

程序分析可以用于发现各类软件中的缺陷，从而改进软件质量，还可以用于软件测试、调试、维护以及缺陷预测。静态分析很难达到非常理想的效果。目前提出的各种静态分析技术试图在精确性和可扩展性之间作出平衡。采用类型推导和抽象解释的方法，分析结果不够精准；采用定义证明的方法，很难处理实际应用中的大规模程序；将符号执行和约束求解结合在一起，存在路径爆炸的问题。仅仅使用静态分析，很难既具有高效率，又能尽可能多发现错误。静态分析能够通过分析给定程序中的所有代码来带到很大的覆盖面，但是由于缺乏程序运行时的信息，其检测结果是不准确的。动态分析则是依赖程序的具体运行去检测缺陷，采集缺乏程序运行时的信息并分析，其结果相比静态分析会准确很多，但无法检测到那些没有运行或者被隐藏起来的缺陷。将动态分析和静态分析有机结合，是一个值得探索的方向。一般来说，静态分析和动态分析可以互相补充，但不能完全替代对方。机器学习等技术能够提升现有的程序分析能力。与机器学习、人工智能等技术的紧密结合，将是程序分析后续的研究趋势之一。同时，降低误报率将依然是程序分析技术的研究挑战和重点。我相信，未来，一个高精度、高效率的程序分析技术会被提出，从而给软件工程师带来更多的便利。

4 参考文献

[1] Nielson F, Nielson H R, Hankin C. Principles of program analysis[M]. Springer, 2015.

[2] Cousot P, Giacobazzi R, Ranzato F. Program analysis is harder than verification: A computability perspective[C]//International Conference on Computer Aided Verification. Springer, Cham, 2018: 75-95.

[3] Box D, Ehnebuske D, Kakivaya G, et al. Simple object access protocol (SOAP) 1.1[J]. 2000.

[4] 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. 1977: 238-252.

[5] 张健,张超,玄跻峰,熊英飞,王千祥,梁彬,李炼,窦文生,陈振邦,陈立前,蔡彦.程序分析研究进展[J/OL].软件学报:1-31[2020-06-28].

[6] Jeannet B, Miné A. Apron: A library of numerical abstract domains for static analysis[C]//International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2009: 661-667.

[7] Singh G, Püschel M, Vechev M. A practical construction for decomposing numerical abstract domains[J]. Proceedings of the ACM on Programming Languages, 2017, 2(POPL): 1-28.

[8] Bagnara R, Hill P M, Zaffanella E. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems[J]. Science of Computer Programming, 2008, 72(1-2): 3-21.

[9] Li X, Liang Y, Qian H, et al. Symbolic execution of complex program driven by machine learning based constraint solving[C]//2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2016: 554-559.

[10] Kong P, Li Y, Chen X, et al. Towards concolic testing for hybrid systems[C]//International Symposium on Formal Methods. Springer, Cham, 2016: 460-478.

[11] Wang X, Sun J, Chen Z, et al. Towards optimal concolic testing[C]//Proceedings of the 40th International Conference on Software Engineering. 2018: 291-302.

[12] Xiong Y, Wang B, Fu G, et al. Learning to synthesize[C]//Proceedings of the 4th International Workshop on Genetic Improvement Workshop. 2018: 37-44.

[13] Roux P, Garoche P L. Practical policy iterations[J]. Formal Methods in System Design, 2015, 46(2): 163-196.

[14] Jeannet B, Schrammel P, Sankaranarayanan S. Abstract acceleration of general linear

loops[C]//Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 2014: 529-540.

[15] Amato G, Scozzari F, Seidl H, et al. Efficiently intertwining widening and narrowing[J]. Science of Computer Programming, 2016, 120: 1-24.

[16] Jiang J, Chen L, Wu X, et al. Block-wise abstract interpretation by combining abstract domains with smt[C]//International Conference on Verification, Model Checking, and Abstract Interpretation. Springer, Cham, 2017: 310-329.

[17] Chen J, Cousot P. A binary decision tree abstract domain functor[C]//International Static Analysis Symposium. Springer, Berlin, Heidelberg, 2015: 36-53.

[18] 王淑栋,尹文静,董玉坤,张莉,刘浩.面向顺序存储结构的数据流分析[J].软件学报,2020,31(05):1276-1293.

[19] 苏亮. 数据流分析关键技术研究[D].国防科学技术大学,2008.

[20] Gosain A, Sharma G. A survey of dynamic program analysis techniques and tools[C]//Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014. Springer, Cham, 2015: 113-122.

[21] Enck W, Gilbert P, Han S, et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones[J]. ACM Transactions on Computer Systems (TOCS), 2014, 32(2): 1-29.

[22] Godefroid P, Levin M Y, Molnar D. SAGE: whitebox fuzzing for security testing[J]. Queue, 2012, 10(1): 20-27.

[23] Gunawi H S, Hao M, Leesatapornwongsa T, et al. What bugs live in the cloud? a study of 3000+ issues in cloud systems[C]//Proceedings of the ACM Symposium on Cloud Computing. 2014: 1-14.

[24] Liu H, Li G, Lukman J F, et al. DCatch: Automatically detecting distributed concurrency bugs in cloud systems[J]. ACM SIGARCH Computer Architecture News, 2017, 45(1): 677-691.