程序静态分析技术与工具*)

杨 宇 张 健
(中国科学院软件研究所 计算机科学重点实验室 北京100080)

摘 要 静态分析对于保证程序质量,提高软件生产率有重要的意义。本文综述了静态分析常用的策略,介绍了当前静态分析的研究现状,比较了目前已有的静态程序分析工具。

关键词 程序正确性,静态分析

Static Analysis of Programs: Techniques and Tools

YANG Yu ZHANG Jian
(Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080)

Abstract Static analysis is of significant importance to guarantee software quality and improve software productivity. In this paper, we give an overview of the strategies that are used widely in static software analysis and compare static analysis tools that have been developed.

Keywords Program correctness, Static analysis

1. 引言

长期以来,程序正确性一直受到计算机科学界和工业界的关注。自20世纪60年代起,就有很多计算机科学家对程序验证进行了研究,提出了各种理论和方法。一般说来,程序验证要求通过推理或者穷举的手段来判定程序的行为是否符合规约。由于要涵盖所有可能情况,而程序设计语言的复杂性使得程序的复杂性随着程序尺寸的增大呈指数级增长,同时证明任一程序正确与否本身是一个不可判定问题,因此程序验证目前只用于证明一些关键的核心模块的正确性而没有得到更广泛的应用。就目前而言,程序验证方法虽然可以保证软件质量,但是往往需要有一定经验的用户花费相当多的时间,因而并不一定能提高软件的生产率[1]。

另一方面,在现实的软件开发中大量的时间被用于发现和消除软件中的错误,也就是软件测试。除了一些大公司在大型软件系统开发中使用了一些自动测试手段[2],在很多情况下,软件测试仍然停留在手工测试阶段。手工测试不仅效率很低,而且容易出错。测试任务往往很繁重,在资源有限、时间紧迫的情况下测试任务常常不能充分完成。测试和调试手段的匮乏已经成为制约软件生产率和软件质量的一个瓶颈。

测试可分为动态测试和静态测试两大类。动态测试就是执行程序,再观察其行为是否满足要求。既可以由用户直接观察,也可以使用一定的辅助工具。例如,PurifyPlus[3]等通过在程序中加入代码来动态地监视程序的运行状态。

静态测试不编译运行程序,而是通过对程序源代码进行分析以发现其中的错误。程序静态分析的目标不是证明程序完全正确,而是作为动态测试的补充,在程序运行前尽可能多地发现其中隐含的错误,提高程序的可靠性和健壮性。事实上,很多相当成熟的系统中还包含着错误。只凭测试人员手工测试很难找出这些错误,而通过静态测试则已经发现了现存系统中的很多错误[4~6]。

目前关于程序静态分析的研究是软件工程研究的一个热点,也已经有一些产品面世[7,8]。本文首先介绍静态分析的主要方法,然后介绍目前已经实现的几个典型系统,并对它们作一些比较,最后说明静态分析的局限以及我们的展望。

2. 程序静态分析的方法和理论

本节简要介绍在程序静态分析工具的构建中常用的方法。这些方法并不完全相互独立,一个静态分析工具常常需要使用多种方法以取得最佳效果。

2.1 符号执行

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[9]。该方法很好地克服了在静态测试时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逻辑可以将程序表示成{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 $c_1 \wedge c_2 \wedge \cdots \wedge c_n$,因此有 $P \wedge c_1 \wedge c_2 \wedge \cdots \wedge c_n \rightarrow R$。可以对约束条件 $P \wedge c_1 \wedge c_2 \wedge \cdots \wedge c_n \wedge \rightarrow R$ 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约不符。如果程序的规约正确,则程序中必定包含错误。

约束求解工具接受的约束条件的集合以及求解能力决定了分析工具发现错误的能力和效率。从理论上讲,很多约束问题是不可解的,或者虽然可解但具有指数级的时间复杂度。尽管如此,对于实际中的很多问题实例来说,高效率的约束求解工具可以在用户可接受的时间内找到解或者断定解不存在。

符号执行和约束求解方法的优点在于它可以精确地静态模拟程序的执行。由于它跟踪了变量的所有可能取值,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,程序执行的可能的路径数目随着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。

*)本文工作得到国家杰出青年科学基金(60125207)和九七三计划(G1998030600)的支持。

2.2 定理证明

自动定理证明是基于语义的程序分析特别是程序验证中常用到的技术。但是采用消解原理的定理证明器一般并不适合于程序分析，因为它不太方便处理整数域、有理数域上的运算。在这种情况下，人们常用各种判定过程（decision procedures）来判别公式是否为定理。当然，判别的方法和公式的形式有关。对于形如 $t_1=u_1\wedge\cdots\wedge t_p=u_p\wedge r_1\neq s_1\wedge\cdots\wedge r_q\neq s_q$ 的一组等式以及不等式的合取，判定的基本方法是首先由该合取式构造成一个图，合取式中的每一个条件对应于图中的一个结点；然后利用给出的等式将对应的 t_i 和 $u_i(1\leqslant i\leqslant p)$ 顶点合并。在顶点合并的过程中我们对合取式中的不等式进行检查，如果发现存在 $1\leqslant i\leqslant q$，r_i 等于 s_i，则可以导出矛盾，该合取式不可满足。

为了能够对包含数学运算以及函数的更复杂的合取式进行推导，Nelson 和 Oppen[10] 提出了所谓的协同判定过程（cooperating decision procedure）。该方法能处理的公式中可以有抽象函数符号以及带大于号、小于号的不等式。具体的处理方法是首先将该合取式的各个分量按照一定的性质分类；然后对于每一个集合中的约束关系结合其他集合中的条件进行推理；如果有新的等价关系产生，则利用等价性传播将各个集合中的表达式的相应部分作替换，并对每一个集合中的约束进行考察，如果发现矛盾则说明此合取式不可满足。

2.3 类型推导

类型推导指的是由机器自动地推导出程序中变量和函数的类型。它在函数式程序设计语言中有着广泛的应用。最近，一些学者也提出将类型推导应用于程序静态分析。其基本思想是，程序中的数据可以依照一定的规则划分为不同的集合；如果把每一个集合作为一种类型，就可以利用类型理论中的一些算法进行分析。例如可以将程序看作一个图，各结点通过加有控制流和数据流信息的边相连。图中的每个结点包含了计算出的变量的值。若将结点依据一定的规则分组，每一组就是一个类型，由此就可以对程序中指针变量的别名等问题进行考察。

上述类型推导适用于控制流无关的分析，具有很好的可扩展性，能够处理大规模的程序。但是它不能解决所有的问题。对于和程序控制流紧密相关的特性则需要引入子类型（subtyping）的概念，来描述类型间的包含关系。例如，打开文件和读文件是两个动作。如果我们将 open File 和 read File 都作为类型，根据“文件必须先打开再读”的原则，read File 必须在 open File 之后，因此我们有 read File<open File 这样的偏序关系。read File 类型就可以被看作包含在 open File 类型中的一个子类型。

2.4 抽象解释

程序设计语言指定了用该语言书写的任何程序的语义。假定一个程序的具体语义由一个函数 F 来表示。P. Cousot 和 R. Cousot[11] 指出，我们可以通过求解这个函数的最小不动点来研究程序的性质。一个不动点是方程 X=F(X) 的解。

但是求解一个程序的最小不动点的问题常常是不可判定问题。程序可能不终止，程序的输入集合可能无穷，这都使我们无法求得程序的最小不动点。即使程序可以终止，并且输入集合有限，找一个程序的不动点也等价于运行该程序。这样的方法对于静态分析代价太高。由此人们考虑通过找一个合适的抽象函数，该抽象解释函数作用于某一个特定的抽象域，只对程序的某一特定属性进行考察。由于抽象域不如具体域精确，因此规模也大大缩减，通常可以达到可计算的范围。一旦我们找到了抽象域上的最小不动点，就可以把它映射到具体域中，作为具体域的函数的最小不动点的近似。虽然这样得出的结果集合常常比实际的最小不动点集合要大，但是用于静态分析已经可以接受。

2.5 基于规则的检查

在面向不同应用的程序中，常常隐含着各种不同的编程规则。例如在多线程程序中要求在使用某一共享变量时遵守“使用前先加锁，使用后解锁”的规则；操作系统的内核处理程序在屏蔽中断进行原子操作后必须打开中断屏蔽等。因此从经验的角度出发，人们提出了基于规则对程序进行分析的方法。采用基于规则进行分析的系统的结构是：首先由一个规则处理器处理规则，将其转换为分析器能够接受的内部表示，然后再将其应用于程序的分析。基于规则的分析方法的优点是能够依据不同的规则对不同的系统进行分析，发现大规模程序中的潜在错误。缺点是由于受到规则描述机制的局限，只能分析特定类型的错误。

2.6 模型检测

模型检测是一种验证有限状态的并发系统的方法，基本思想是对于有限状态的系统构造状态机或者有向图等抽象模型，再对模型进行遍历以验证系统的某一性质。模型检测的难点在于如何避免状态空间爆炸。为此人们提出了多种方法。其中符号化模型检测方法是将抽象模型中的状态转换为逻辑公式，然后判定公式的可满足性。还可将模型转化成自动机，同时将需要检查的公式转换为一个等价的自动机，再将此自动机取补。这两个自动机的积构成了一个新的自动机，则判定模型是否具有某一属性的问题就转换成检查这个新自动机能接受的语言是否为空。

和定理证明相比，二者目标不同，实现手段也有很大的差异。定理证明的目标是证明命题的永真性，采用逻辑规则进行推导证明；而模型检测常采用枚举所有可能状态的方法。

模型检测在硬件检测和协议验证方面已经有了很好的应用，然而对于软件检测，由于软件本身具有的高复杂度，该方法只能针对程序中的某一方面特性构造抽象的模型进行检测。

以上几种方法之间有着一定的联系。类型推导和模型检测的方法都是从不同角度对程序进行抽象，也可以说是抽象解释。符号执行和类型推导都着眼于由程序推导出一组约束，并利用约束求解工具进行求解。但是两种方法生成的约束条件的形式不同，约束求解的策略也不同：在符号执行过程中形成的约束以布尔表达式以及线性等式和不等式为主；类型推导形成的约束条件则是集合关系表达式。

静态分析方法往往基于程序的一定的抽象表示。抽象语法树、有限自动机以及有向图是常用的表示方法。

2.7 分析工具中使用的规约

机器自身无法判定程序的正确与否，一般需要由人给出一定的规约予以说明。分析工具完成的工作就是理解规约和依据规约对程序进行检查。基于规则的分析工具是将规则作为判定程序是否正确的标准。更多的分析工具[6,12]是根据程序不同的上下文嵌入不同的规约。一种方法是引入全称量词、存在量词，以一阶逻辑为基础以断言的形式描述程序的前置条件、后置条件以及某一点的约束条件。同时也可以根据需要引入和具体程序设计语言相关的表示机制。此处需要提及的是 JML（Java Modeling Language）[13]。它利用 Java 的面向对象机制定义了和集合、序列等集合论中的抽象描述等价的抽象类，以及 forall、exists 等用于描述一阶逻辑公式的关键字。

由于程序员是规约的使用者，因此分析工具中的规约应易于使用，接近程序设计语言。

3. 当前一些静态分析测试的研究项目

以下介绍当前在程序静态测试方面的一些研究项目。

PREfix[4]使用符号执行及约束求解方法对C/C++程序进行静态分析测试。PREfix的工作流程是:首先分析源代码,将其转换成抽象的语法树,然后对过程依照调用关系进行拓扑排序,再为每个过程生成相应的抽象模型,最后静态模拟执行路径并用约束求解的方法对约束集合进行检测。其中函数抽象模型的提取决定了PREfix查错的能力和精度。为了解决路径空间爆炸的问题,PREfix选择了一定数量具有代表性的路径进行分析。用PREfix对Apache,Mozilla等程序进行检查,发现了其中存在的数百个错误和安全隐患。

BANE[14,15]是一个用于构造程序分析工具的工具集,它提供了类型推导的接口并内嵌了多种不同的约束求解器。Cqual[16]是以BANE为基础构造的分析检测工具。用Cqual对Linux内核进行了分析,发现了其中关于加锁动作的错误。

Metal[5]是基于规则的程序静态分析检测工具。Metal将程序的规则以状态机的形式进行描述,对程序依照规则进行分析。Metal对Linux内核、FLASH等程序进行了检查,发现了其中隐含的不少错误。

ESP[17]也是一种基于规则的系统。它使用了多种策略,包括保守别名分析(conservative alias analysis)、数据流分析以及路径敏感的符号执行。ESP被用于检查gcc中使用的C stream库的某些性质的正确性。ESP着重于过程间程序分析,而Metal侧重于过程内的程序分析。

ESC[6,18]采用定理证明的方法进行程序分析验证。ESC的基本思想是,首先由带规约的程序生成证明条件,然后由定理证明器进行处理。如果成功则说明程序中不包含错误,如果失败则可以通过后处理程序推导出导致失败的可能原因。ESC利用最弱前置条件(weakest precondition)和最强后置条件(strongest postcondition)将规约以及约束条件转换为Dijkstra卫式命令,再转换成证明条件由定理证明器证明。ESC还可以用于检查多线程的并发程序中经常出现的一些错误。

SLAM[19]用模型检测的方法对C程序进行分析。SLAM的分析过程是一个对抽象的程序模型不断细化的反复迭代的过程。首先对程序进行抽象,建立模型,然后分析模型。如果发现能够导致错误的路径则终止分析过程并报告错误;如果已经枚举证明所有可能的路径都不可能导致错误,则对模型进行细化(refine),建立并分析细化后的模型,如此迭代直到发现错误或者超时。SLAM已经用于验证Windows XP中的一些驱动程序的接口的正确性。BLAST[20]的思想和SLAM的思想类似,都采用了模型检测的方法对程序属性进行验证。

Splint[12,21]原名为LcLint,是一种轻量级的静态检测工具。它并不将程序转换成中间表示,可以检查一些和编程风格相关的以及影响程序移植性的错误。Splint已经被广泛地应用到实际的程序开发过程,从改善程序风格的角度来提高软件的质量。

Cyclone[22]和Vault[23,24]从增强语言功能的角度考虑消除程序中的隐患。其中也用到了一些程序静态分析的手段。Cyclone和Vault都提供了更强的类型检查,并细化了程序设计语言支持的数据类型以便能够从根本上规范程序设计,避免可能导致错误的不良风格。例如Cyclone将指针类型细分为不空指针,可空指针以及可参与运算的指针等等。在编译时这些语言利用细化的类型等安全机制进行静态分析以保证程序的可靠。

表1对上述工具作一些比较。

表1 已有程序静态分析工具的比较

工具	方法、技术	被检查程序的设计语言	检查错误的类型	是否需要额外的注解	成功的例子
Cqual	类型推导,约束求解	C	在工具实现中预先定义	不需要	Linux内核C Stream库
ESC	判定过程	Java,Modula-3	空指针引用,并发程序中的简单错误	需要	
ESP	类型推导	C/C++	未指明	不需要	验证了C Stream库的正确性
Metal	基于规则的推导	C,FLASH machine code	根据规则的不同可以进行不同的检查	需要给出规则	FreeBSD, Linux内核, FLASH
PREfix	符号执行,约束求解	C/C++	限于空指针引用,内存泄漏等预先定义的几种	不需要	Windows 2000
SLAM	模型检测	C	由用户根据规约指定	需要	Windows XP中一些驱动程序的接口
Splint	基于程序风格和注释的检测	C	潜在的风格错误	需要	Wu-ftp,Apache

结束语 从上一节可以看出,程序静态分析得到相当程度的重视,有关工具也确实发现了软件中的不少错误。特别是对一些状态有限,同时对稳定性要求很高的软件,采用适当的静态分析技术显得很有必要。这是提高软件质量的一种重要手段。

当然,静态分析也有不少局限性。对于程序的某些性质(比如和指针运算、动态存储分配等相关的性质),用静态分析难以奏效。一般说来,静态分析可以作为动态测试的补充,但并不能完全代替动态测试。

由于程序设计语言的复杂性,静态分析很难达到非常理想的效果(既具有高效率,又能找到尽可能多的错误)。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,分析结果不够精确,有时不能满足软件测试工程师的要求;采用定理证明的方法。由于算法复杂性的限制,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,如果枚举所有可能的路径,也不太现实,因为路径个数随着程序尺寸的增长而呈指数级增长。

我们认为,静态分析需要更加有效的算法。在程序模型的构造、路径选择等方面,需要有更好的策略。将分析的路径限制在一定的范围,将动态测试和静态分析有机结合,这也许是一个值得探索的方向。

参 考 文 献

1 Brooks F P Jr. The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, (2nd ed.) Addison-Wesley 1995
2 Fewster M, Graham D. Software Test Automation. Addison - Wesley, 1999
3 PurifyPlus. http://www.rational.com/products/pqc/index.jsp
4 Bush W, Pincus J, Sielaff D. A static analyzer for finding dynamic programming errors. Software - Practice and Experience, 2000, 30 (7): 755～802
5 Engler D, Chelf B, Chou A, Hallem S. Checking system rules using system-specific programmer-written compiler extensions. In: Proc. of the Fourth Symposium on Operating System Design and Implementation, San Diego, Oct. 2000
6 Detlefs D L, Leino K R M, Nelson G, Saxe J B. Extended static checking: [SRC Research Report 159]. Compaq System Research Center, 1998
7 LDRA. http://www.ldra.co.uk
8 Parasoft. http://www.parasoft.com
9 King J C. Symbolic execution and testing. Comm. of the ACM, 1976, 19: 385～394
10 Nelson G, Oppen D. Simplification by cooperating decision procedures. ACM TOPLAS, 1979, 1(2): 245～257
11 Cousot P, Cousot R. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conf. Record of the Fourth ACM Symposium on Principles of Programming Languages, 1977
12 Splint. http://lclint.cs.virginia.edu
13 JML. http://www.cs.iastate.edu/~leavens/JML.html
14 BANE. http://www.cs.berkeley.edu/Research/Aiken/bane.html
15 Aiken A, Fähndrich M, Foster J S, Su Z. A toolkit for constructing type- and constraint-based program analyses. In: Proc. of the Second Intl. Workshop on Types in Compilation, Kyoto, Japan. March 1998
16 Foster J S, Terauchi T, Aiken A. Flow-sensitive type qualifiers. In: Proc. of the ACM SIGPLAN 2002 Conf. on Programming Languages Design and Implementation, Berlin, Germany. June 2002
17 Das M, Lerner S, Seigle M. Path-sensitive program verification in polynomial time. In: Proc. of the ACM SIGPLAN 2002 Conf. on Programming Languages Design and Implementation, Berlin, Germany, June 2002
18 Flanagan C, Leino K R M. Houdini, an annotation assistant for ESC/Java. FME 2001, Lecture Notes in Computer Science 2021, 2001. 500～517
19 Adams S, Ball T, Das M, et al. Speeding up dataflow analysis using flow-insensitive pointer analysis. In: 9th Static Analysis Symposium, Lecture Notes in Computer Science 2477, Sep. 2002
20 BLAST. http://www-cad.eecs.berkeley.edu/~rupak/blast/
21 Evans D, Guttag J, Horning J, Tan Y M. Lclint: A tool for using specifications to check code. In: Proc. of the ACM SIGSOFT Symposium on the Foundations of Software Engine-ering. Dec. 1994
22 Jim T, Morrisett G, Grossman D, et al. Cyclone: A safe dialect of C. In: USENIX Annual Technical Conf. Monterey, CA, June 2002
23 Vault. http://research.microsoft.com/vault/
24 DeLine R, Fähndrich M. Enforcing high-level protocols in low-level software. In: Proc. of the ACM conference on Programming Language Design and Implementation, June 2001. 59～69

(上接第170页)

```
〈soap:operation soapAction=""/〉〈/operation〉
〈soap: binding transport =" http://schemas.xmlsoap.org/soap/
http"style="rpc"/〉
〈/binding〉
〈service name="MyHello"〉
〈port name="HelloIF Port" binding="tns:Hello IF Binding"〉
〈soap:address xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
location="https://localhost:8443/security-jaxrpc/security"/〉
〈/port〉〈/service〉〈/definitions〉
```

如上服务描述语言描述了服务调用的名称、接口、地址以及调用的参数、数据模型等细节信息。客户端应用程序根据这一服务描述，生成客户端的 Stubs 对象程序，和服务器端的 Tie 对象交互，完成远程过程调用。客户和服务器之间消息传递的 XML 文档被系统屏蔽了，这一部分用户不需要自己写交互的 SOAP 消息。系统中可见的是系统自动生成的 SOAP 消息生成和解析的类。如下：

HelloIF_SayHello_RequestStruct__MyHello__SOAPBuilder;

HelloIF_SayHello_RequestStruct__MyHello__SOAPSerializer;

HelloIF_SayHello_ResponseStruct__MyHello__SOAPBuilder;

HelloIF_SayHello_ResponseStruct__MyHello__SOAPSerializer。

结论 RPC 是分布式系统的重要机制。基于 XML 的 RPC 将请求信息封装成 XML 的格式，利用 XML 的灵活性和可扩展性，简单有效地实现了异构平台上不同应用程序的互操作。基于 XML 的 RPC 同时简化了高层程序设计，最大限度地屏蔽了异构平台间的通讯机制的差异。

基于 XML 的 RPC 的通用性会造成其调度效率不能达到最好，原因之一就是消息传递时在 XML 解析、类型转换等方面时间的浪费。但是对于分布式系统，RPC 的主要问题是网络拥塞。并且，随着硬件性能的提高，解析、类型转换等花费的时间会越来越少，在调用过程中所占的资源耗费的比率会越来越低。

总的来说，基于 XML 的 RPC 是很有效的一种远程调用机制。

参 考 文 献

1 Birrell A D, Nelson B J. Implementing Remote Procedure Calls. ACM Transactions on Computer Systems, 1984, 2(1): 39～59
2 http://www.sei.cmu.edu/str/descriptions/rpc_body.html.
3 http://www.w3.org/XML/, W3C Extensible Markup Language (XML) page.
4 http://www.xml.com/pub/a/98/10/guide0.html, A Technical Introduction to XML, by Norman Walsh, October 03, 1998
5 http://www.ietf.org/rfc/rfc1831.txt. "RPC: Remote Procedure Call Protocol SpecificationVersion 2", R. Srinivasan, Sun Microsystems. Aug. 1995
6 http://www.opengroup.org/onlinepubs/009629399/, CDE1.1 Remote Procedure Call. Copyright@1997 The Open Group
7 http://www.opengroup.org/dce/info/papers/tog-dce-pd-1296.htm#intro, DCE Overview
8 http://www.itl.nist.gov/div897/staff/barkley/5277/titlerpc.html, Comparing Remote Procedure Calls. John Barkley, Oct. 1993
9 http://www.ietf.org/rfc/rfc1832.txt. "XDR: External Data Representation Standard", R. Srinivasan, Sun Microsystems. Aug. 1995
10 http://www.ietf.org/rftc/rfc1833.txt. "Binding Protocols for ONC RPC Version2" R. Srinivasan, SunMicrosystems. Aug. 1995
11 http://www.w3.org/TR/SOAP/, Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000
12 "Web Services Conceptual Architecture (WSCA) 1.0" by Heather Kreger, IBM Software Group, May 2001
13 http://www.soapware.org/
14 http://xmlrpc.com/spec. XML-RPC Specification. By Dave Winer. Tue, Jun15, 1999
15 http://weblog.masukomi.org/writings/xml-rpc_vs_soap.htm, Kate Rhodes, XML-RPC vs. SOAP".
16 http://java.sun.com/webservices/, Java Web Services tutorials 1.4