形式化方法的研究进展与趋势

CCF 形式化方法专业委员会

卜　磊[1]　陈立前[2]　陈　哲[3]　陈振邦[2]　冯新宇[1]　冯　元[4]
贺　飞[5]　李国强[6]　刘万伟[2]　马菲菲[7]　宋　富[8]　田　聪[9]
王淑灵[7]　吴志林[7]　薛　白[7]　杨鹏飞[7]　尹良泽[2]　詹博华[10]
张　民[11]　张立军[7]　张兴元[12]　赵永望[13]

[1]南京大学，南京
[2]国防科技大学，长沙
[3]南京航空航天大学，南京
[4]悉尼科技大学，悉尼
[5]清华大学，北京
[6]上海交通大学，上海
[7]中国科学院软件研究所，北京
[8]上海科技大学，上海
[9]西安电子科技大学，西安
[10]慕尼黑工业大学，慕尼黑
[11]华东师范大学，上海
[12]中国人民解放军陆军工程大学，南京
[13]北京航空航天大学，北京

摘　要

形式化方法是计算机科学的一个传统研究方向，是保证计算机软硬件系统正确性与安全性的重要途径。本报告将对形式化方法各个方面最近 5 年的进展进行相对全面的总结，包括定理证明、形式模型、形式语义与形式建模、形式规约、形式验证技术与方法、形式验证工具及应用，以及量子程序的分析与验证。最后，本报告对形式化方法的发展趋势进行展望。

关键词：交互式定理证明，SMT，自动机，形式语义，分离逻辑，时序逻辑，演绎推理，抽象解释，模型检测，符号执行，量子程序的分析与验证

Abstract

Formal methods is a traditional research field in computer science, and is an important approach to guaranteeing the correctness, safety and security of computer hardware and software systems. This report will summarise the progress of different aspects of formal methods during the last five years, in a relatively complete way, including: theorem proving, formal models, formal semantics and formal modeling, formal specification, formal verification techniques and methodologies, formal verification

tools and applications, the analysis and verification of quantum programs. This report will also discuss the perspectives of formal methods in the future.

Keywords: Interactive theorem proving, SMT, automata, formal semantics, separation logic, temporal logic, deductive verification, abstract interpretation, model checking, symbolic execution, the analysis and verification of quantum programs

1 引言

随着信息技术的广泛应用，特别是信息技术与物理世界和人类社会的高度融合，整个社会的信息化程度不断提高，人们对计算机系统的可靠性与安全性提出了更高的要求。计算机系统的一个小小的错误或者安全隐患可能影响到国家安全或者人民群众的生命财产安全。

形式化方法是保证计算机系统正确性与安全性的一种重要方法，其采用数学（逻辑）证明的手段对计算机系统进行建模、规约、分析、推理和验证。

形式化方法是计算机科学的一个传统研究方向，迄今为止，共有 12 位学者因为或者部分因为形式化方法的研究工作而获得图灵奖。形式化方法是一个基础性和交叉性很强的研究方向，它与计算机科学的其他分支如计算理论、编程语言、软件工程、计算机安全等都密切相关；同时，近年来与生物信息、控制理论、电子自动化设计等多个学科交叉明显。

形式化方法越来越受到国际/国内计算机学术界的重视，比如国际组织 IFIP 下面至少有三个工作组（working group）与形式化方法相关，欧洲有专门的形式化方法组织 FME，美国计算机学会（ACM）在 2014 年成立了 SIGLOG，即逻辑与计算专委会（Special Interest Group on Logic and Computation），涵盖计算逻辑、自动机理论、形式语义、程序验证等方向，而中国计算机学会则在 2015 年成立了形式化方法专业组，今年刚刚正式更名为“形式化方法专业委员会”。

形式化方法方面的国际旗舰会议包括关注形式化方法的理论基础的 LICS（LogicIn Computer Science）、关注形式化方法的理论和应用结合的 CAV（Computer-Aided Verification）和 FM（International Symposium on Formal Method）、关注自动推理的 IJCAR（International Joint Conference on Automated Reasoning）等。

由于形式化方法在保证计算机软硬件系统的正确性和可靠性方面的有效性，它已经被许多国际标准化组织列为安全攸关系统必备的技术手段。例如，国际航空软件标准 DO178B、DO178C 中明确要求开发安全可靠的航空软件必须使用形式化方法；又如，在软件安全等级 SIL1-4 中，安全级别最高的 SIL3 和 SIL4 要求必须使用形式化方法。而且，形式化方法已经逐渐为 IT 产业界所接受和采纳，例如：

- 微软开发了一系列影响深远的形式化验证工具，比如 Z3、Dafny、F^*、SLAM 等，用于对其开发的众多软件的正确性进行分析。

- Facebook 将基于分离逻辑的验证工具 Infer 广泛用于其 Android 应用的开发过程中。
- 亚马逊在 2014 年成立了自动推理组，用 SMT 求解器去验证其 Web 服务的正确性。
- 国内的华为公司最近建立了形式验证团队，使用定理证明辅助工具来验证操作系统内核的正确性和安全性。

形式化方法使用数学（逻辑）证明的手段对计算机系统进行建模、规约、分析、推理，其主要涵盖以下几个研究方向：

- 定理证明：对逻辑推理过程进行研究，尽可能提高逻辑推理过程的自动化程度。
- 形式模型：对形式模型的理论性质和各种判定问题进行研究，提出高效的判定算法，并开发原型工具。
- 形式语义与形式建模：为了对计算机系统的行为进行推理验证，需要对编程语言的语义进行严格的定义，使用形式模型对计算机系统的行为进行建模。
- 形式规约：为了对计算机系统的行为进行推理验证，也需要对程序行为所满足的性质用某种形式语言（逻辑、自动机等）进行严格的定义。
- 形式验证技术与方法：在形式语义/形式建模以及形式规约的基础上，将计算机系统的分析与验证问题转化为逻辑推理问题或者形式模型的判定问题，用定理证明工具/求解器或者某个形式模型的原型工具来进行验证。
- 形式验证工具以及应用：基于形式验证理论、方法和技术上的研究成果，开发自动形式验证工具并在工业级系统中进行应用。
- 量子程序分析与验证：针对量子程序的特点发展其分析与验证理论、方法、及工具。

本报告将对形式化方法最近 5 年的国内外研究进展进行相对全面的介绍，并对发展趋势进行展望。

2 国际研究现状

2.1 定理证明

定理证明方面的研究可以分成两部分：交互式定理证明（interactive theorem proving）和自动推理（automated reasoning）。

交互式定理证明的目标是通过用户和计算机相互协助来完成一个形式化证明。这里所用到的工具叫作证明辅助工具（proof assistant）。现在最常用的两个证明辅助工具是 Coq 和 Isabelle。其他的证明辅助工具包括 HOL Light、HOL4、PVS、ACL2、Lean、Mizar 等。由于在证明的过程中用户可以向计算机提供各种帮助，因此这种方法可以用来验证

非常复杂的定理。但同时，现在的证明辅助工具操作起来还比较烦琐，所以使用的门槛较高。

自动推理的目标是让计算机完全自动地证明一个数学命题。目前这个领域最常用的算法包括 SMT（Satisfiability Modulo Theory）和归结（resolution）。基于 SMT 的证明器包括 Z3、CVC4 等。基于归结原理的证明器包括 SPASS、Vampire 等。这些算法的使用不需要（或需要更少的）人为干涉，所以在计算机系统的推理和验证方面得到了广泛的应用，但另一方面，能用这种方法证明的数学命题比较有限。

下面对交互式定理证明和自动推理这两方面近 5 年的国际研究现状进行总结。

2.1.1 交互式定理证明

交互式定理证明的自动化传统上依靠在证明辅助工具里实现证明策略（tactic）。几乎所有常用的证明辅助工具都提供了实现证明策略的语言。如何改善这些语言，让用户更直观地实现证明策略是一个长久的问题。在许多证明辅助工具里，证明策略可以用实现这个工具的语言（比如 OCaml 或 ML）来实现。在 Coq 里，Ltac 提供了一个更高层次的证明策略语言。Eisbach[16]尝试在 Isabelle 里面实现类似的功能。Lean[17]是一个最新的证明辅助工具，其中的一个设计重点是允许更有效的证明策略的实现[18]。

另一种自动化方法是调用证明辅助工具之外的自动证明器[1]。外部证明器的调用可以分成以下几个步骤。首先，根据要证明的子目标，从已有的定理中筛选一部分相对更有可能用到的定理（premise selection）。然后，这些筛选出来的定理和子目标一起被转换成无类型的一阶逻辑，交给自动证明器求解。最后，如果自动证明器能够证明命题，证明辅助工具可以直接信任这个结果，或通过自动证明器反馈的信息在自己的系统里重新合成证明。

很多证明辅助工具都在使用或尝试使用这种自动化方法。其中，Isabelle 里的 Sledgehammer 最具代表性，现在已成为 Isabelle 大多数应用中必不可少的工具。最近为改善 Sledgehammer，有以下一些研究：文献［2］加入了 SMT 证明器的使用，文献［19］改善了从 Isabelle 里的类型到无类型逻辑之间的转换，文献［20］用机器学习的方法改善了筛选已有定理的过程。自动证明器的使用在其他一些证明辅助工具中也进行了尝试，比如 HOL Light[21]、Coq[22]等。

使用外部自动证明器的一个弱点是无法处理涉及高阶逻辑的命题。虽然存在一些从高阶逻辑的命题转换到一阶逻辑的算法，但转换的过程可能会让命题变得非常烦琐。由 Jasmin Blanchette 主持的 Matryoshka 项目希望在已有的自动证明器的理论基础上加上对高阶逻辑的支持。最近的成果包括文献［1-12］。

在证明辅助工具的设计中，逻辑基础是一个非常重要的课题。除了保证可靠性以外，选用的逻辑基础应当能够有效地表达需要证明的理论，并且让实际的证明过程尽可能方便。Isabelle/HOL、HOL Light、HOL4 等工具使用一种简单的（不包含依赖类型的）类型论（simple type theory，或者叫 higher-order logic）。Coq 和 Lean 使用依赖类型论（dependent type theory）。逻辑基础的一个最新进展是 homotopy type theory[29]。这是一个基于依赖类

型的更复杂的类型论。除了这个逻辑在数学基础上本身的意义外，它也允许用相对简单的方式定义和计算代数拓扑里的基本群（和同伦群）[30]。在另一个方向，也在尝试选用无类型的集合论作为逻辑基础。文献［31］用这个基础验证了一些包括基本群的传统数学理论。

数据类型是定理证明器的逻辑基础中最重要的部分之一。Coq 等基于依赖类型论的定理证明器，在引入数据类型时需要对其逻辑演算进行扩展，此类扩展的正确性由逻辑学家的元理论分析作为支撑。与此不同，在 HOL4、Isabelle/HOL 等基于简单类型论的定理证明器中，数据类型的引入所采用的是“定义扩展”（definitional extension）的方式。自 2011 年开始，Isabelle/HOL 中的数据类型定义机制发生了很大变化[27]。基于范畴论构造的新数据类型包替换了原有的数据类型包，由此获得了更好的开放性和更加灵活多样的表达方式。这方面最新的进展是文献［28］在 HOL 中引入了“nonuniform 数据类型”。

数学理论的形式化既是交互式定理证明的原始目标，也是验证计算机系统的基础。这方面有两个大型项目在最近几年完成：奇序定理（odd order theorem）[32]和开普勒猜想的证明[33]。这两个项目各自都耗费了超过 20 人年的时间。奇序定理是群论里面的一个结果，由 Feit 和 Thompson 在 1962 年证明，论文长达 255 页，其中涉及表示论、伽罗瓦理论等比较深奥的数学理论。开普勒猜想在 1998 年被 Hales 证明，其中涉及大量在计算机上进行的计算，所以证明的正确性也曾经存在争议。

除了这两个大型项目外，形式化数学在各个基础领域都有进展。在数学分析方面，文献［6］验证了常微分方程的一些基本理论。这项工作之后被用来验证 Smale’ s 14th problem 的一部分证明[7]。同时，它也被用来验证一个混成系统的逻辑基础[13]。在概率论方面，文献［8］在 Isabelle 里建立了基本理论。它在文献［9］里被用来验证关于马尔可夫链和马尔可夫决策过程的基本特征。在线性代数方面，最近被验证的结果包括 Jordan Normal Form[24]和 Perron-Frobenius 定理[25]。线性代数最近被用于证明了一个关于深度学习表达能力的定理[26]。

形式化数学也可以用于验证各种数值和符号算法。最近的工作包括在 Coq 里验证一个数值计算定积分的算法[5]。这项工作实现并验证了一个完全可信的算法，并在测试的过程中找到了一些已有的实现上的错误。另一个主要的验证目标是多项式问题判定的算法。文献［15］在 Isabelle 里完成了一元多项式问题判定得到的证书（certificate）的验证。验证多元多项式判定证书所需要的柯西留数定理也已经被验证[14]。这一系列工作的最终目标是将实数定理的专用证明器 MetiTarski 整合到 Isabelle 系统中。

其他一些最近被验证的定理包括哥德尔不完备定理[3]、中心极限定理[4]、格林公式[10]、基本的纽结理论[39]等。

Alexandria[23]是由 Larry Paulson 主持的形式化数学方面的项目。它的目标包括整理已有的工作并填补空隙，以得到一个覆盖所有大学水平数学的形式化数学库。同时，它也希望提高 Isabelle 系统在形式化数学方面的可用性。这里包括提供各种定理搜索功能，以及自动向用户推荐证明方法。这个项目的另外一个目标是在 Isabelle 里融入和整理各种可以被验证的数值和符号算法。

2.1.2 自动推理

在理论计算机科学和人工智能领域中，逻辑公式的可满足性问题一直是一个重要的问题。世界各国学者在这方面做了大量研究工作，特别是在命题逻辑的可满足性问题（SAT）上取得了很大进展。SAT 是命题逻辑推理中的一个经典的判定问题，也是第一个被证明为 NP 难的问题，一直备受关注。但是随着研究的深入，人们发现 SAT 的表达能力和应用范围有很大的局限性。因此，SMT 近年来开始引起人们的极大兴趣。SMT 的研究对象是各种领域知识的逻辑组合，经常表达为带等词的一阶逻辑公式[50]。与 SAT 判断布尔公式的可满足性不同，SMT 判断的是理论组合的可满足性，它的抽象层次更高，表达能力也更强。它可以看作是对 SAT 的扩展，将 SAT 中的某些布尔变量用理论谓词取代。例如，$x+2=y \vee x=f(y-x+1)$ 是 SMT 的一个实例，它包含了两个理论：线性算术、未解释函数。如果用布尔变量 $P1$ 表示线性理论上的谓词 $x+2=y$，$P2$ 表示包含未解释函数的等式 $x=f(y-x+1)$，则此 SMT 公式将变为一个命题逻辑公式 $P1 \vee P2$。SMT 涉及的理论是一些数学理论和计算机领域内用到的数据结构理论，包括差分逻辑（difference logic）、线性算术理论（linear arithmetic，包括线性实数算术（LRA）和线性整数算术（LIA））、位向量（bit vector）、数组（array）、未解释的函数（uninterpreted function）等[51]。随着 SMT 应用范围的扩展，新的理论也被不断地加入，例如近年来兴起的集合理论。为了表达方便，自动推理领域使用记号 SMT(T) 来表示理论 T 上的 SMT 公式，如线性实数算术理论上的 SMT 公式记为 SMT(LRA)，位向量上的 SMT 公式记为 SMT(BV) 等。

对 SMT 求解的研究不但是自动推理、约束求解领域主流国际会议 IJCAR/CADE、CP 和 SAT 的主题之一，在验证和分析领域的著名国际会议 CAV、TACAS 上也是非常受关注的议题。SMT 也受到产业界的高度关注。微软、Intel、Cadence、NEC 等公司的研究院或实验室都在开展与 SMT 相关的研究项目。自 2005 年以来，自动推理学界每年都举办 SMT 求解器比赛（SMT-COMP）[52]，以促进研究者不断改进 SMT 求解器的性能。目前，比较有代表性的 SMT 工具包括美国爱荷华大学的 CVC4[53]、SRI International 的 Yices[54]、微软的 Z3[55]等，它们被集成于多个研究领域的自动工具之中，如高阶逻辑的交互式理论证明器 HOL、Isabelle、ACL2，模型检测工具 BLAST、MAGIC，程序分析工具 KLEE 等。

SMT 是理论谓词的逻辑组合，其可满足性涉及两个层面：命题逻辑层次与理论层次。根据如何处理这两个层面上的可满足性，SMT 求解的主流算法经历了从积极（eager）类算法到惰性（lazy）类算法的演变。

SAT 求解技术取得的重大进步推动了积极算法的发展，积极算法是早期的 SMT 求解器采用的算法，也称为 bit-blasting。它是将 SMT 公式转换成可满足性等价的命题逻辑公式，一般为 CNF 形式，然后用 SAT 求解器求解。这种方法的好处是可以采用高效的 SAT 求解器，同时它的求解效率也依赖于 SAT 求解器的效率。对于不同的理论，积极算法需要用不同的转换方法和改进方法以提高转换和求解的效率。例如，对于 EUF 一般用 per-constraint 编码，对于 DL 通常用 small-domain 编码等。积极算法的正确性依赖于编码的正

确性和 SAT 求解器的正确性，而且对于一些大的例子来说，编码成 CNF 公式很容易引起组合爆炸，也就是公式的长度指数级增长，因此这类方法在实际应用中效果不是很好，一般解决不了规模较大的工业界实例。但由于位向量理论上的 SMT 公式（SMT(BV)）与命题逻辑公式较为相似，因此 bit-blasting 仍为求解 SMT(BV) 的主要算法。

惰性算法是目前主流的 SMT 求解方法，也是当前研究最多的算法。这种算法框架被称为 DPLL(T) 算法[56]。其基本思路是：将 DPLL 风格的 SAT 判定过程与特定理论的求解器（T-solver）紧密结合，基于 DPLL 的 SAT 求解器负责命题逻辑部分的推理，得到对布尔变量的（部分）赋值，而理论求解器负责判断 SAT 求解器得到的赋值——理论谓词的合取的可行性。这里，DPLL(T) 中的 T 代表理论。如果 T 是线性算术理论，T-solver 可以是 Simplex 算法或它的变种。DPLL(T) 这一名称的含义是，这是一个集成了 DPLL 算法的框架，可以结合不同理论及其判定过程而实例化。早先的 DPLL(T) 算法是将 SAT 求解器（DPLL 部分）当作黑盒，理论求解器检查模型的一致性要等到 DPLL 算法给出一个解之后才进行，这种方法的缺点是理论求解器很少参与 DPLL 的求解过程。后来出现了 online 方法对其做了改进，使得理论求解器参与 DPLL，从而提高了求解效率。这些改进主要有理论预处理、选择分支、理论推导、理论冲突分析和引理学习等。为了 DPLL 求解与理论求解更为有效地结合，理论判定过程一般要设计为增量式的（incremental）和可回溯的（backtrackable）。

DPLL(T) 算法是完备求解算法。为了对 DPLL(T) 的求解能力形成补充，近年来，研究人员开始尝试用非完备算法求解 SMT。Fröhlich 等人针对位向量理论上的 SMT 公式（SMT(BV)），提出了在理论层面设计局部搜索算法，将 SAT 的局部搜索算法中的一些先进技术提升到理论层面上[57]。Niemetz 等人进一步将位向量上的推理机制与局部搜索算法相结合[58-59]，从而大幅提高了局部搜索的效果。特别是当将局部搜索与基于 bit-blasting 的完备算法混合使用时，对很多 SMT(BV) 实例的求解可以加速 1 ~ 3 个数量级。但位向量是比较接近于 SAT 的理论，在算术理论等其他理论上如何设计局部搜索算法仍是一个具有挑战性的问题。除了局部搜索之外，研究人员也尝试了其他新颖的非完备算法来求解 SMT 问题。在线性整数理论方面，Bromberger 等人提出了用空间超立方体探测 SMT(LIA) 公式解空间内部的方法，该方法适用于含整数解数目众多的 SMT(LIA) 公式，对于 SMT-LIB 中部分公式的求解速度比目前的 SMT 求解器快了几个数量级[60]。在浮点数理论方面，Fu 等人将 SMT 转成一个无约束的优化问题，然后用蒙特卡罗马尔可夫链（MCMC）的方法求解，在一些实例上获得了比 Z3 等 SMT 求解器快 700 倍以上的效果[61]。

2.2 形式模型

在计算机科学中，形式模型一般指用来对特定结构（比如串和树）进行接受、生成和变换的数学模型。形式模型包括自动机、文法和重写系统等。对形式模型的研究贯穿计算机科学的发展历程，它奠定了计算机科学的很多分支的理论基础。

图灵机——现代计算机的理论模型，是最早提出和被研究的形式模型之一，它被证明和其他几种形式模型（比如递归函数、λ 演算等）是等价的。而有限自动机和上下文无关文法是编程语言的词法和语法分析的理论基础。

由于篇幅的限制，本报告主要集中在自动机模型上。经典的自动机模型包括有限自动机、下推自动机、图灵机等。为了将对系统行为的推理从有限状态扩展到无穷状态，从定性推理扩展到定量推理，研究人员考虑了经典自动机模型的各种扩展，比如带概率的自动机模型、时间自动机模型、混成自动机模型等。近 20 年来，在工业界尤其在硬件的分析和验证上取得很大成功的自动验证（模型检测）工具，比如 SPIN、UPPAAL、PRISM 等，分别建立在无穷串上的自动机、时间自动机以及概率自动机的基础上。

下面对概率自动机模型、混成自动机模型以及无穷状态并发系统模型近 5 年的国际研究进展进行介绍。

2.2.1 概率自动机模型

为了对概率系统的性质进行推理与验证，研究人员提出了很多概率自动机模型，包括：离散时间马尔可夫链（DTMC）[86]，连续时间马尔可夫链（CTMC）[71]，离散时间马尔可夫决策过程（MDP）[75]，连续时间马尔可夫决策过程（CTMDP）[89]、概率自动机（probabilistic automata）[70]。离散时间马尔可夫链和离散时间马尔可夫决策过程可以看成概率自动机的子模型。

概率自动机模型的本质是在经典自动机模型中加入相同行为下的不确定性选择和对应的概率行为，从而在行为标记上同时表达非确定性和概率行为。文献［103］指出，概率模型中的非确定性选择非常适合刻画并发和分布式系统。概率系统的性质多用经典逻辑的概率扩展来进行规约，比如概率分支时序逻辑（PCTL）[113] 概率线性时序逻辑（PLTL）以及它们的结合 PCTL* [76]。概率模型检测的本质是将验证问题转化为数学问题。比如，离散时间马尔可夫链上的 PCTL 模型检测问题可归约到线性方程组的求解，其时间复杂度是多项式的[113]；而其上的 PLTL 模型检测问题则更为复杂，是 PSPACE 完全的；该问题在马尔可夫决策过程中则更为复杂，时间复杂度是双指数的[78]。在连续时间马尔可夫链上，文献［71］中最先提出了连续随机逻辑（CSL）的验证算法，随后，文献［73］中提出了高效的近似算法。在概率自动机上添加时间属性可以将其扩展为概率时间自动机，概率时间自动机的模型检测问题最早在文献［97］中得以研究，其验证基于度量时序逻辑（MTL）[90]。因其直观易用，文献［100］对其进行了系统的研究，尽管 MTL 本身不可判定，但它的很多子逻辑都是可判定的。

和经典的模型检测问题一样，概率模型检测同样面临着状态空间爆炸的问题。针对这一问题的主要优化思路是状态约简和符号化方法。在状态约简方法中，互模拟技术被应用得最多，其主要思想就是把行为一致的状态放在一起，通过求得商系统来缩小状态空间。在概率自动机上，文献［132］中首先提出了概率自动机上基于状态的互模拟关系，随后，文献［81］中提出了基于分布的互模拟关系，［87］中提出了基于分布的互模拟关系的高效算法。概率互模拟在逻辑刻画、度量刻画、算法研究以及弱互模拟关系

等方向的工作非常广泛，比如文献［98，79，101，85，117］，在理论层面还有一些基于范畴论的语义和互模拟、模拟关系的刻画，比如［106］。而符号化方法则是用二叉决策图（BDD）[77]节省模型的存储，加快运算效率，近些年多端二叉决策图（MTBDD）在概率模型检测方面有着相当不错的结果[84]。基于概率自动机及其他概率模型的概率模型检测在验证实际软件、系统和协议中起到了关键作用。概率模型检测工具 PRISM[88]上已经有大量的应用案例，包括随机的分布式算法、通信协议、网络协议、多媒体协议、攻击下的安全性、量子加密协议、仿生算法、控制软件、博弈、性能评估和可靠性、电源管理等。研究人员已经使用概率时间自动机和概率 I/O 自动机验证了许多在实践中被广泛应用的标准协议，比如 IEEE 802.3 载波监听多点接入/碰撞检测协议[95-96]、IEEE 1394 火线根竞争协议[94]、IEEE 802.11 无线局域网协议[93]、IPv4 Zeroconf(零配置网络服务规范）协议[92]和 IEEE 802.15.4 载波侦听多路访问/冲突避免协议（ZigBee）[82-83]。

2.2.2　混成自动机模型

混成系统是一种嵌入在物理环境下的实时系统，一般由离散组件和连续组件连接组成，组件之间的行为由计算模型进行控制，其演化过程由离散事件跳变和连续时间动态过程共同完成。而混成模型是针对混成系统的数学模型。

为了验证混成系统的性质，提出了多种混成模型，如混成自动机（hybrid automata）[130]、混成 Petri 网[131]、混成程序[132]等，其中混成自动机得到了最为广泛的认可与应用。相较于复杂度高的非线性混成自动机，线性混成自动机复杂度低，可应用于大规模系统形式化验证的优点使其成为过去 20 年混成模型形式化方法研究领域的热点，其理论及方法已颇为成熟，如基于线性混成自动机的验证工具 HyTech[133]、HyperTech[134]、d/dt[135]、SpaceEX[136]等。最近几年，能更准确刻画混成系统行为的非线性混成自动机的研究也已经引起了学者们的广泛关注，产生了丰富的研究理论及成果，如泰勒模型方法[137]、边界可达性分析方法[138-140]、凸优化方法[141-144]、Hamilton-Jacobi 偏微分方程方法[145-146]等。

以上提及的主要是针对混成系统中连续动态行为由常微分方程描述的混成模型。而对连续动态行为由时滞微分方程描述的时滞混成模型的形式化验证，自 2004 年文献［147］提出基于栅栏函数（barrier certificate）对时滞模型进行形式化验证的方法后，时滞混成模型的形式化验证研究基本处于停滞状态。而在近 5 年，时滞混成模型的形式化验证取得了重大的突破。文献［148］通过将时滞变量视作扰动因素，提出了非线性时滞动态网络的有界验证方法。文献［149］提出了一种基于区间泰勒模型近似的方法，通过计算可达集的过高估计来验证一类最简单的时滞微分方程的安全性和稳定性。文献［150］进一步将此方法推广到了验证时滞微分方程的行为是否满足用时序逻辑描述的安全性质中。文献［151］提出了基于欧拉方法计算时滞微分方程可达集过高估计的数值方法，同时能够给出严格的误差上界。文献［152］中利用敏感性分析给出了一类其解具有同胚性质的时滞微分方程，将常微分方程中利用边界计算可达集的方法推广到了此类时滞微分系统，提出了计算其可达集的过高和过低估计的方法。文献［153］将常微分方程可达集计算中的泰勒模型方法推广到了时滞模型中，计算了可达集的过高和过低估计。

2.2.3 无穷状态并发系统模型

无穷状态并发系统模型可以统一在进程重写系统（Process Rewrite System，PRS）中[154]。在进程重写系统框架下，并发系统模型的分类如图1所示，其中有我们比较熟悉的有限系统（FS）、为递归程序建模的系统下推系统（PDA）、为并发程序建模的系统Petri网（PN）、以及为并发交互系统建模的进程代数（PA）等。

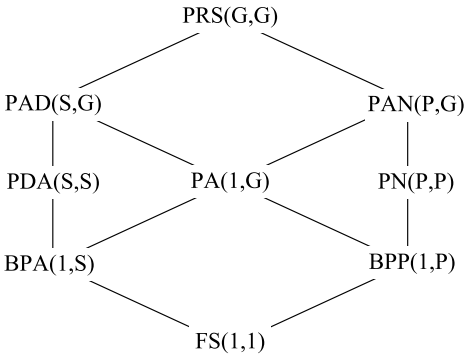


图1 并发系统模型的分类

在进程重写系统的等价性研究中，首先研究的是强互拟等价性问题，该研究起源于20世纪80年代。Baeten、Bergstra和Klop在1987年证明了BPA强互模拟是可判定的[155]。这在当时是一个相当令人惊奇的结果，因为众所周知的是，BPA的语言等价问题是不可判定的。接着，Christensen、Hirshfeld、Moller证明了BPP上的强互模拟等价性验证是可判定的[156]。Hirshfeld和Jerrum进一步将这两个结论加以推广，他们证明了一个包括BPA和BPP的模型——PA上的强互模拟也是可判定的[157]。Stirling证明了PDA上强互模拟等价性验证是可判定的[158]。Jančar通过归约Minsky机停机问题证明了PN上强互模拟验证问题是不可判定的[159]。因此在进程重写系统中，所有PN之上模型的强互模拟等价性验证都是不可判定的。

除了可判定性的研究外，强互模拟等价性验证问题的算法和复杂性也是这一类研究关注的焦点。Balcázar、Gabarró和Sántha首先证明了有限状态系统判定互模拟等价是P-难的[160]，因此进程重写系统上所有模型的强互模拟等价性验证都有P-难这个下界。Hirshfeld、Jerrum和Moller在normed BPA（nBPA）和normed BPP（nBPP）上，证明了这两个模型的强互模拟等价性验证有多项式时间算法[161-162]。一个模型是normed，指的是这个模型中的所有进程都能有限终止。在一般BPA上，最近，Kiefer证明了强互模拟等价性验证有EXPTIME-难下界[163]。因此该问题目前的复杂性介于EXPTIME和2-EXPTIME之间。对于一般BPP，Jančar证明了该问题是PSPACE-完备的[164]。最近，Benedikt等证明了PDA上强互模拟的下界是Non-Elementary[165]。

如果考虑带内部动作的互模拟等价性验证研究，我们会考虑弱互模拟以及分支互模拟这两类等价性研究。针对弱互模拟，目前公开的问题非常多，我们知道：其一，在PRS的绝大部分模型上，如在BPA或BPP以上的模型，弱互模拟等价性验证是不可判定的；其二，在非常基本的无限状态系统——BPA和BPP，乃至nBPA和nBPP上，弱互模拟等价性验证的判定性至今仍然是公开的。分支互模拟等价是van Glabbeek和Weijland定义的一种比弱互模拟等价更精细的观测行为等价[166]。2011年，Czerwiński、Lasota和Hoffman证明了在nBPP上分支互模拟是可判定的[167]。Czerwiński和Jančar证明了nBPA上分支互模拟验证是NEXPTIME可解的[168]。

表1中列出了进程重写系统定义的5个模型到目前为止的已知判定结果。其中，

“~”“≅”“≈”分别表示强互模拟、分支互模拟和弱互模拟，“√”表示可判定，“×”表示不可判定，“?”表示未知。

表 1　进程重写系统定义的 5 个模型已知判定结果

	BPA	BPP	PDA	PN	PA	nBPA	nBPP	nPDA	nPN	nPA
~	√	√	√	×	?	√	√	√	×	√
≅	?	?	×	×	×	√	√	×	×	×
≈	?	?	×	×	×	?	?	×	×	×

2.3　形式语义与形式建模

对计算机系统进行验证的基础是编程语言的形式语义和对系统行为的形式建模。

形式语义学（formal semantic）是研究程序设计语言的语义的学科，以数学为工具，利用符号和公式，精确地定义和严格地解释计算机程序设计语言的语义，从而消除语言设计者、开发者和使用者之间理解的差异性。形式语义学主要分成四大流派：操作语义、代数语义、指称语义和公理语义。其主要作用是帮助理解语言，支持语言标准化，指导语言设计，帮助编写编译器和语言系统，支持程序验证和软件可靠性，有助于软件规范化。4 种语义的主要区别如下所述：

1）操作语义着重模拟数据处理过程中程序的操作。

2）代数语义基于代数，用代数结构刻画程序语法实体，用代数公理刻画实体含义及语法实体间的关系。

3）指称语义主要刻画数据处理的结果，而不是处理的细节。

4）公理语义用公理化的方式描述程序对数据的处理，主要用于程序的推理和验证。

操作语义、代数语义、指称语义和公理语义之间的关系通常采用伽罗瓦连接理论。不同的语义模型之间的伽罗瓦连接理论可以抽象为从一个语义模型元素到另一个语义模型元素的函数，可用于证明程序语言不同语义之间的一致性。

形式建模将计算机软硬件系统的行为和性质用某种形式模型精确地刻画。形式建模一般采用通用形式建模语言（比如 Petri Net、Event-B、Pi-演算、CSP、SysML、Lusture 等）或者专用形式模型（比如有限自动机、下推自动机、概率自动机等）来进行。

早期程序语言的语义只是在论文中给出，这样的语义作用非常有限，既不能用计算机测试语义的正确性和一致性，也不能用于程序的验证和分析，比如文献［183-184］。

中期的语义一般采用定理证明器（如 Coq、ACL2、Isabelle、HOL 等）的元语言实现，此类语义可用于程序语言语义和程序性质的手动或半自动化验证[185-189]。另一种技术方法是采用函数式语言实现程序的语义，如文献［188］中采用 Haskell 实现 C99 的部分语义，这样的语义可以作为解释器执行 C99 程序。

K 框架是最近流行的一种定义语言的形式语义的途径，其基于重写逻辑，通过定义语言的操作语义，自动生成对应程序的形式分析与验证工具[190]。2012 年，Ellison 和

Rosu 定义了 C99 标准中的大部分语言成分的语义和标准库函数[182]，是迄今为止最完备的 C99 语言的语义，该语义在 K 框架中进行了实现，跟文献［188］中的语义类似，可作为解释器执行程序，并用 GCC 的压力测试程序进行大规模的测试验证，保证其语义正确性。同时，利用 K 框架的集成工具，可以根据语义快速生成 C99 程序的各种验证工具，如运行时验证、模型检查器、推理验证器等。文献［218］对 C 程序语义的定义和应用，给出了一种新型的程序语义的定义方法和程序语义应用模式。基于 K 框架，Java1.4[191]、JavaScript[192]、PHP[193]、Python 3.3[194]、Rust[195]等工业级编程语言的语义也陆续被实现和应用，这些语义大部分都进行了一致性测试，并生成了一些工具用于进行实验性的程序分析与验证。

形式建模方面，函数式程序可以用树自动机[207]、高阶下推自动机[208]来建模。尽管理论上 Ong 等人已经证明高阶下推自动机的模型检测问题的复杂性是 n-EXPTIME 完全的[209]，但以东京大学 Kobayashi 团队为代表的研究小组相继提出一些高效的算法，大大缩短了验证的时间[210]。另外，实时嵌入式系统一般使用时间自动机来建模，然后进行验证，这方面的主流工具为 UPPAAL[211]。此外，一种基于 UML 的实时嵌入式系统建模语言 MARTE 近来逐渐成为研究的热点，并成为国际对象管理组织认定的标准建模语言[212]。MARTE 为专门的系统标准（如 OSEK、ARINC 和 POSIX）提供相应的建模插件，以支持相应系统的建模。

自适应系统与多智能体一般使用 Petri 网[206]、UML[213]、Z[214]以及马尔可夫模型[215]等来建模。最近，一些针对自适应系统的新的形式化建模方法被相继提出，如自适应抽象状态机模型[216,218]和时间 Petri 网[217]，前者通过 MAPE-K 控制循环描述系统分布式和去中心化的自适应行为，后者通过时间 Petri 网模型来描述系统的实时自适应行为。

最近，深度神经网络的形式建模迅速成为研究热点。神经网络常被形式化表示为一组在整数域或实数域上的线性算术方程组与激活函数对应的非线性方程组的集合[219]。由于非线性方程的存在，无法利用已有的线性优化算法或者 SMT 求解器进行求解。目前比较普遍的方法是根据具体的非线性方程对线性优化算法或 SMT 求解器进行扩展，验证神经网络的安全性，或找到对抗干扰实例[219-220]。

2.4 形式规约

除了需要对计算机系统进行形式建模之外，我们还需要对系统应满足的性质进行准确的规约。一般使用逻辑公式对系统的性质进行规约。经典的规约语言包括针对串行程序的一阶逻辑及其扩展分离逻辑、针对并发程序的时序逻辑等。

一阶逻辑在命题逻辑（布尔逻辑）的基础上加入关系（函数）符号和一阶量词。Floyd-Hoare 逻辑将一阶逻辑用于对串行程序的行为进行规约和推理[231]。Floyd-Hoare 逻辑未提供对内存布局的描述手段，不能刻画“指针”等程序结构。为此，分离逻辑（separation logic）在 2000 年左右被提出[232]。该逻辑在一阶逻辑的基础上引入了分离算子，即分离合取 * 和分离蕴含 *。分离算子可以非常方便地表示程序不同组成部分所操作

内存区域的独立性，并在推理规则中将 Floyd-Hoare 逻辑的不变式（invariance）规则替换为框架（frame）规则，从而实现对程序行为的局部推理，而分离逻辑中的归纳谓词则可以非常直观地描述链表和树等动态数据结构。分离逻辑提出后，有一大批在其之上的后续工作，大大推动了程序验证的发展。

Floyd-Hoare 逻辑和分离逻辑比较适合对串行程序终止时的输入和输出应该满足的关系进行规约，而并发程序一般不终止，对其进行规约需要不同的规约语言。时序逻辑是目前使用的主流的并发程序的规约语言。按照对时间的不同解释，时序逻辑大体可以分为线性时序逻辑和分叉时序逻辑两大类，前者如 LTL(Linear-time Temporal Logic) 等，后者如 CTL(Computation Tree Logic)、CTL* 等。

针对串行程序的形式规约语言近 5 年的进展集中在分离逻辑方面。研究人员近 5 年对于带归纳谓词的分离逻辑的各种片段的可判定性、复杂性、判定算法以及启发式算法进行了广泛的探讨。分离逻辑不仅可以描述形状性质，比如链表，也可以描述数据约束，比如有序性。

研究人员首先对不含数据约束的带归纳谓词的分离逻辑的判定算法进行了探讨。Cook 等人证明了含有单链表片段归纳谓词的分离逻辑子集的可满足性问题和蕴含问题是多项式时间可判定的[233]。Antonopoulos 等人证明了带归纳谓词的分离逻辑的蕴含问题一般是不可判定的[234]。另一方面，Iosif 等人将分离逻辑嵌入树深度受限的图上的一元二阶逻辑中，证明了带某一类归纳谓词的分离逻辑的蕴含问题是可判定的[235]。而且，Brotherston 等人对于带比较一般的归纳谓词的分离逻辑的可满足性问题进行了深入的研究，他们证明了该逻辑的可满足性问题是 EXPTIME-完全的[236]。Iosif 等人将带归纳谓词的分离逻辑的蕴含问题归约成树自动机的语言包含问题，开发了分离逻辑蕴含问题求解器 SLIDE[237]。

研究人员在含有可以描述数据约束的归纳谓词的分离逻辑方面也进行了广泛的探讨。Chin 等人和 Qiu 等人考虑了带归纳谓词和一般数据约束（比如算术约束、集合约束、多重集约束等）的分离逻辑，提出了求解蕴含问题的启发式算法[238-239]。他们还对带线性算术数据约束的归纳谓词的分离逻辑的可满足性问题进行了研究，提出了启发式算法，并对这些算法在哪些子类上是完备的进行了探讨[240-241]。Piskac 等人将分离逻辑公式嵌入经典一阶逻辑的含有集合变量和可达谓词的某种扩展，使得可以对序关系和集合数据约束进行描述推理[242-243]。Enea 等人提出了可组合归纳谓词的概念，简化了对含有归纳谓词和复杂数据约束的分离逻辑公式的求解[244]，但他们提出的算法是不完备的。

针对并发程序的规约逻辑 LTL 由 Pnueli 提出[245]，其在命题逻辑的基础上引入了 Next(X) 和 Until(U) 两个时序算子。LTL 由于具有简洁、直观的特点，同时有具有相对较强的表达能力，因而在验证领域被广泛使用。LTL 能够表达全部 star-free 的 ω - 正规性质，其模型检测问题及可满足性判定问题都是 PSPACE-完全的[246]。CTL 由 Clarke、Emerson 等人提出[247]。同 LTL 相比，CTL 中引入了全称路径量词 A 和存在路径量词 E(二者互为对偶关系)。CTL 的模型检测问题可在多项式时间内完成[248]；CTL 的可满足性问题是 EXPTIM-完全的[250]。CTL* 是 LTL 与 CTL 的共同超集。CTL* 公式的模型检测

问题也是 PSPACE-完全的[248,251]，其可满足性问题是 2EXPTIME-完全的[249]。

时序逻辑方面近 5 年的进展集中在经典时序逻辑 LTL、CTL 和 CTL* 的各种扩展上，包括博弈扩展、概率扩展、实时扩展、量子扩展等。

首先，研究人员提出了用于描述多主体系统性质的时序逻辑的博弈扩展，比如 ATL、ATL*[252]、Strategy Logic[253]等。ATL 的模型检测问题是 PTIME-完全的[252]；其可满足性问题是 EXPTIME-完全的[254]。ATL* 是 ATL 的扩展，ATL* 的模型检测问题以及可满足性问题的复杂度均是 2EXPTIME-完全的[252,255]。Strategy Logic 具有非常强的表达能力，即使是其一次交错片段（否定范式公式中量词 ∀ 和 ∃ 最多只能嵌套交错一次的公式构成的片段），也能够编码全部的 ATL* 公式。该逻辑可以描述纳什均衡、安全均衡等重要的性质。Strategy Logic 的模型检测问题和可满足性问题是非初等可判定的[256]。但是，对于其一些简单的片段，如一次交错片段，其模型检测问题是 2EXPTIME-完全的[256]。

时序逻辑的概率扩展主要包括 PCTL、PCTL* 等，它们的语义定义在离散马尔可夫链或者离散马尔可夫决策过程（MDP）上。对于离散马尔可夫链模型，PCTL 的模型检测问题可在多项式时间内完成[113]，而 LTL 和 PCTL* 的模型检测问题仍是 PSPACE-完全的[257-259]。对于离散马尔可夫决策过程，PCTL 的模型检测问题也可以在多项式时间内完成，但 LTL 的模型检测问题却是 2EXPTIME-完全的（即使是定性验证）[78]。PCTL 的可满足性问题比较复杂，目前还没有完全得到解决。

时序逻辑的实时扩展，典型包括 MTL 和 TCTL：前者定义在线性结构上，后者定义在分支结构上。MTL 可以看作是 LTL 的扩展，它取消了 X 算子，并在 U 算子上附加一个区间下标 I——其 Unitl 公式形如 $f_1 U_I f_2$，它显式地指明了 f_2 成立的时刻应当位于区间 I 内[260]。TCTL 在 CTL 的基础上引入了时钟变量，并取消了 X 时序算子[260]。这两类逻辑的语义均定义在实时系统上（如时间自动机）。MTL 的可满足性问题和针对时间自动机的模型检测问题在连续时间上是不可判定的[261]，在离散时间上是 EXPSPACE-完全的[262]。

时序逻辑的量子扩展主要包括 QCTL、QCTL* 等，其语义主要定义在量子马尔可夫链上。QCTL 在量子马尔可夫链上的验证可在多项式时间内完成[272]；同样，该逻辑的可满足性的判定问题目前仍然没有解决。关于量子马尔可夫链上的 LTL 模型检测问题，国内外学者仍在进行研究。

另外，研究人员针对计算机安全的背景，提出了时序逻辑的“超性质”的扩展[263]，主要包括 HyperLTL、HyperCTL* 等。HyperLTL 的模型检测算法在最坏情况下是非初等可判定的，同时这两种逻辑的可满足性问题也是非初等可判定的，但对于其某些特定片段却存在较为高效的算法[263]。Finkbeiner 等人给出了关于 HyperLTL 和 HyperCTL* 的模型检测的实现[264]。

2.5 形式验证技术与方法

形式验证包括 4 种技术与方法，即演绎推理、抽象解释、模型检测和符号执行。

演绎推理的基本思想是，将程序满足其形式规约的证明问题转化为一组数学命题的证明。这组数学命题被称为证明义务（proof obligation）。若这组证明义务的证明确实能够蕴含“程序满足其规约”这一命题，那么我们就说该证明系统是正确（sound）的。Folyd-Hoare 逻辑是一种经典的基于演绎推理的验证系统。

抽象解释理论是一种对程序语义进行可靠抽象（或近似）的通用理论[331]。该理论为程序分析的设计和构建提供了一个通用的框架[332]，并从理论上保证了所构建的程序分析的终止性和可靠性（即考虑了所有的程序行为）。抽象解释本质上是通过对程序语义进行不同程度的抽象以在分析精度和计算效率之间取得权衡。这种由（面向某类性质的）语义抽象及其上的操作所构成的数学结构称为抽象域。到目前，已出现了数十种面向不同性质的抽象域。其中，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。

模型检测的基本思想是通过遍历系统的状态空间以验证系统模型是否满足给定的关键性质，并在不满足性质时给出具体反例路径。因此，如何对模型状态空间进行快速遍历对于模型检测至关重要，而状态空间爆炸问题则自然成为模型检测技术面临的主要问题。与模型检测技术取得成功的硬件领域相对比，软件系统的状态空间复杂性大幅提高。由于模型检测覆盖范围较广，鉴于篇幅所限，本报告中我们将以软件模型检测为代表，来对软件模型检测近期技术的进展做一个总结。

符号执行[393-396]提供了一种系统性遍历程序路径空间的手段，符号执行中的程序路径精确刻画了这条路径上的程序信息，可基于路径信息开展多种软件验证确认阶段的活动，包括自动测试、缺陷（包括功能缺陷）查找以及部分程序验证等。理论上，相比于需要固定程序输入的分析方法，符号执行通过符号分析，能覆盖更多的程序行为。另一方面，符号执行技术依赖于 SAT/SMT 技术，求解器的能力是决定符号执行效果的关键因素；同时，符号执行中程序路径空间大小随着程序规模的增大而呈指数级增长，例如，单就串行程序来讲，一个具有 n 个条件语句的程序段，就有可能包含 2^n 条路径，这也是制约符号执行能力的关键因素。

2.5.1　演绎推理

近年来基于演绎推理的程序验证技术方面有大量突破性进展，在此很难一一介绍，我们仅仅介绍其中一些最具代表性的工作。

分离逻辑是近年来程序验证领域的重大突破之一，它由 Reynolds、O’ Hearn、Ishtiaq 和 Yang 等人在 2000 年前后提出[274-275,196,232]。并发分离逻辑是 Brookes[278] 和 O’ Hearn[279] 对分离逻辑的扩展，支持对共享内存的并发程序的验证。它的基本思想是要求并发任务对共享资源（包括内存）的访问在互斥的临界区中进行。而在临界区之外，并发任务只能访问自己的私有资源。要实现这一点，必须有效保证共享资源和私有资源的分离，以及不同任务各自的私有资源的分离。而分离逻辑中的分离合取恰恰能够用来方便地描述这一点。并发分离逻辑主要针对良好同步（properly synchronized）的程序的验证，之后有大量工作对并发分离逻辑进行扩充，以支持细粒度并发或者无锁并发程序的验证。在 Brookes 和 O’ Hearn 的综述文章中[280]，对基于并发分离逻辑开展的工作有完整的介绍，

在此不再赘述。

并发分离逻辑及其扩展大多针对串行一致性内存模型上的并发程序开展验证。然而，由于处理器体系结构中以及编译器中引入的种种优化，导致并发程序的行为并不满足串行一致性。这些行为统称为弱内存模型。近年来，Viktor 等人对并发分离逻辑进行扩展，提出一系列在 C11 内存模型上的程序逻辑[281-287]，证明 C11 内存模型（子集）上程序的正确性。

关系型程序逻辑可以验证两个程序之间的关系，或者一个程序在两种输入下的行为之间的关系。前者可用于程序精化的验证，而后者则可用于安全性质，特别是信息流控制（information flow control）机制的验证。Benton[288] 和 Yang[289] 较早提出关系型程序逻辑。Beringer 和 Hofmann 提出将关系型程序逻辑应用于信息流控制[290]。Barthe 等在关系型程序逻辑方面开展了大量研究，主要将其应用于安全性质验证[291-292]。Turon 等[293] 和 Liang 等[294] 提出了关系型程序逻辑来开展并发程序的精化验证。作为对关系型程序逻辑的扩展，Sousa 和 Dillig 提出笛卡儿霍尔逻辑，用于验证 k-safety，即程序 k 次不同执行之间的关系[295]。

近年来，有大量的基于演绎推理验证技术对实际系统在代码级的验证。其中比较有代表性的包括操作系统内核验证和编译器验证。操作系统内核验证相关的工作包括澳大利亚 NICTA 对 seL4 的验证[296]、耶鲁大学团队对 CertiKOS 的验证[297]、中科大团队对 μC/OS-II 的验证等[40]。编译器验证工作中最著名的是 INRIA 对 CompCert 的验证[298-299] 以及后续工作[300-302]。此外，还包括对分布式系统的验证[303-304]、安全系统的验证[305-308]、文件系统的验证[309-310] 等。

2.5.2 抽象解释

近年来，抽象解释的主要研究进展包括提高分析精度、可扩展性、可行性 3 方面。

在提高分析精度方面，抽象域本身表达能力的局限性是当前面临的主要问题。为了弥补抽象域表达能力的局限性，最近的研究进展可分为两类：（1）通过结合符号化方法来提高分析精度，利用 SMT 求解器[347,342]、插值[334] 等技术来计算程序语句迁移函数的最佳抽象，以改进抽象域在语句迁移函数上的精度损失；（2）提高抽象域的非线性表达能力，如基于组合递推分析[345-346] 将符号化分析与抽象解释结合起来了以生成多项式、指数、对数等形式非线性不变式，基于椭圆幂集来生成二次不变式[323] 等。另外，在面向特定应用改进精度方面，最近有研究[326] 针对实际计算机程序，尤其是嵌入式系统中数值都是有限二进制位数表示的问题，提出了改进策略来提高基于抽象域的分析的精度。

在提高可扩展性方面，如何有效降低存储开销和提高计算效率是目前考虑的主要问题。在这方面，近年来的研究进展包括：

1）利用变量访问的局部性原理，降低当前抽象环境中所涉及的变量维数，并根据数据流依赖的稀疏性，降低抽象状态的存储开销和传播开销。基于该思想，Oh 等人[354-355] 提出了一种通用的全局稀疏分析框架，在不损失分析精度的前提下能够显著降低时空开销，并在商业化静态分析工具 Sparrow 上进行了应用，取得了显著的可扩展性提升效果。

2）利用矩阵分解等在线分解优化策略来对抽象域操作的实现算法进行优化。基于该思想，最近 Singh 等人对实际静态分析工具中常用的八边形抽象域的实现进行了优化，优化后八边形分析的性能提升达 140 多倍[359]；对多面体抽象域的实现进行了优化，优化后多面体分析的性能提升了 2 ~ 5 个数量级，并且很多原有实现分析不出来的规模较大的实例采用优化后的方法能够分析出来[360]；在此基础上，Singh 等人还提出了一种通用的基于分解的优化策略[361]，能够在不改变基抽象域的基础上自动实现分解的优化，从而不需要人工重新实现基抽象域，并基于这一思想实现了开源抽象域库 ELINA。这种基于在线分解的方法在提高抽象域的分析效率时不会造成精度损失。最近，Singh 等人[362]还提出了一种基于强化学习来加速静态程序分析的方法，在每次迭代中，利用强化学习来决策选哪个转换子，以在精度和不动点迭代收敛速度之间进行权衡。在抽象域编码实现上，Becchi 等人[324-325]最近改进了未必封闭多面体域（支持严格不等式约束）的双重描述法，在表示中避免了松弛变量的引入，极大地提高了分析效率，并在原有广泛使用的多面体抽象域实现库 PPL 的基础上开发了新的开源实现 PPLite。为了提高形态分析的分析效率，Li 等人[343]提出一种方法对程序的抽象状态做二次抽象以合并相似的抽象状态，从而降低形态分析中析取抽象状态数，提高了分析效率。

在提高可行性方面，复杂数据结构自动分析的支持、不同谱系目标程序的支持、活性性质分析的支持是目前主要的关注点。在复杂数据结构的自动分析方面，最近的研究重点关注针对数组内容的精确分析[348]、混杂数据结构的建模[344]、数值与形态混合的程序分析[330,337]、关系型形态分析[340]。在支持不同谱系目标程序方面，最近的研究重点关注多线程程序的自动分析[349]、中断驱动型程序的自动分析[365-366,363]、概率程序的分析[327]、操作系统代码的安全和功能性分析[356,344]、JavaScript 等动态语言的分析[458]。在目标性质支持方面，近年来在抽象解释领域出现了一些新的用来分析时序性质和终止性的方法[351-352]。

2.5.3　模型检测

将状态空间符号化表达，并在符号化后的空间上进行计算和遍历是软件模型检测的基本方法。然而，即使是符号化后的状态空间，其验证也并不是一个简单的问题。因此，如何对复杂状态空间进行抽象简化一直是相关研究的一个重要方向。

反例制导的抽象精化方法（Counter-Example Guided Abstraction Refinement，CEGAR）[385]是当前复杂系统模型检测的主要手段之一。其基本思路为当抽象后模型违反给定性质时，将相关反例在原系统上进行确认，如确认成立则发现反例，否则分析出抽象过程中导致此虚假反例出现的因素，从而对上次抽象进行精化。在 CEGAR 框架当中如何对相关“反例”进行确认，并进一步对抽象过程进行精化是其中一项重要问题。近年来，此方向上最主要的一个研究方法是基于插值（interpolant）来对抽象谓词进行精化的研究，Wolverine[392]、UFO[378]，以及文献［383］等均是近年来相关方向的典型工作。以上基于谓词抽象的软件系统模型检测方法在相关领域中发挥了重要作用，比如提高了适用系统规模，可生成实际反例，不会产生误报等。然而，由于软件系统的高复杂性，相关方

法目前可解决问题的规模仍然受到一定限制。

相对应地，在模型状态空间过于复杂的情况下，有界模型检测（Bounded Model Checking，BMC）[381]思想被提出。BMC 思想也被应用在了软件模型检测方向上。自 CBMC[386]、F-Soft[390]等以来，近 10 年有大量工具出现并得到有效应用。目前相关方向的工作主要集中于如何对代码中各种数据结构，如数组、位向量、堆等进一步提供编码机制，如何对给定深度内行为空间进行有效编码及剪枝等来提升可验证系统规模，并提高验证效率等[387-388]。

上文大量 BMC 工作通过对状态间迁移关系进行一定深度展开后再进行 SAT/SMT 编码及求解的方式进行。Bradley 在 2011 年提出了 IC3 方法[382]，采用逐步展开并推导证明的方式进行求解，以其优异性能及可扩展性在硬件验证领域迅速引起了广泛关注，相关技术也被应用到软件验证中[384]。除了 SMT、IC3 之外，将验证问题以 Horn 子句形式进行编码并求解也是近年来相关方向研究热点之一。随着相关约束求解技术的提升，SeaHorn[389]、VeriMap[379]、TRACER[391]等基于 Horn 子句的软件验证工具快速涌现。

在上述几大方向的基础上，通过多技术深度融合来进行代码验证是近年来的重要趋势之一。如将抽象解释（Abstract Interpretation）与模型检测相结合，将插值技术与 SMT 结合等。其中最具有代表性的工作即为近年来引起广泛影响的 CPAchecker 工具[380]，其在上述工作基础上提出了可配置验证框架，通过统一接口集成多种求解器和技术来对软件系统进行验证，以期能够得到更好的性能和可扩展性。

在模型检测方面，随着研究工作的逐步深入，可处理问题种类逐步增加，处理效率也得以稳步提升，相关领域引起了越来越广泛的关注。自 2012 年起召开的一年一度的 Software Verification Competition 更是会对现有各类验证工具的实用性和处理问题能力进行激烈竞赛和详细分类评估。我们推荐对此领域感兴趣的读者对该竞赛进行关注，从而对软件模型检测的当前处理能力及热点研究现状有一个快速的了解。

2.5.4 符号执行

目前，符号执行技术仍面临提高可扩展性（scalability）与可行性（feasibility）这两方面的挑战。在可扩展性方面，路径空间爆炸和复杂路径约束是导致此挑战的主要原因，因此相关工作主要集中在缓解路径空间爆炸和约束求解优化两个方面；在可行性方面，相关工作则主要集中在环境建模和多形态分析目标两个方面。下面分别对这两方面国际上的最近的研究进展进行介绍。

在缓解路径空间爆炸方面，目前已有工作基本分为两个思路：首先是在具体目标下提供高效的搜索策略，使符号执行分析更快地达到目标，包括提高程序的覆盖率[459-461]、判断某个程序点是否可达[462]、产生满足正规性质的程序路径[463]、探索程序不同版本的差异部分[464-465]等；然后是通过约束输入范围、削减或合并路径来减小程序的路径空间，包括基于输入模板[400-401]、程序切片[402,406,404]、程序抽象（包括摘要技术）[408-415]、偏序约简[416-417]、条件合并[418-419]以及等价路径约简[420,443,421,407]等方法。此外，还有一些并行符号执行[466-467]的工作，其基本思想是在多台机器上并行探索程序路径空间，目标也

是缓解路径空间爆炸，不同工作的差别在于探索任务的分配方式及并行算法。

在约束求解方面，已有工作也可分为两方面：在调用求解器前对路径条件的查询进行优化，以减少求解器的调用次数或降低求解时间；支持复杂程序特征的高效编码。第一个方面包括查询缓存和重用[422-423,425]、基于约束独立性的优化[401,399]、增量式求解[401,424]等；第二个方面包括对机器数[403-404]、数组[404,397]、浮点[426-430]、字符串[431]、动态数据结构[432-434]等的支持。

在环境建模方面，已有工作基本都是在分析的精确性、可靠性、建模工作量以及可扩展性之间进行权衡和折中，包括手工建模[401,468]、自动综合[435]、动态执行[398]、全栈执行[402]等。

在多形态分析目标方面，主要是应对多语言和多应用领域的复杂性，包括 C ++ 程序[469-470]、高级语言中多态的处理[471]、二进制程序[436,405,437]、脚本语言程序[439]、分布式程序[438]、数据库操作程序[440]、无线传感器网络程序[441]、并发或并行程序[442-444]、嵌入式程序[445]、PLC 程序[446]等的符号执行方法。

同时，面向新的分析需求，也有一些新的符号执行技术出现，其中比较有代表性的是概率符号执行技术[449-451]，其基本思想是通过符号执行来得到程序的路径，然后使用 SMT 解空间体积计算技术来计算每条路径的路径条件的解的个数，通过每条路径对应的解的个数，可以计算每条路径的概率。基于此，可以开展包括低概率缺陷查找、低概率路径的测试用例生成、生成程序的性能分布[452]、刻画代码变迁[453]等活动。此外，符号执行技术与其他技术之间紧密融合，以提高分析的效果，也是目前新的发展趋势，包括与模型检测[447]、抽象解释[455]、模糊测试[448]、随机测试[454]等技术结合，以提高程序的覆盖率或缺陷发现效率。

2.6 形式验证工具以及应用

形式化方法早期主要在硬件验证和协议验证中取得应用。近年来，随着形式化技术的快速发展，形式化方法在越来越多的软件系统验证和嵌入式系统验证中得到应用，并取得显著成效。本报告主要围绕软件形式验证工具和面向嵌入式系统的混成系统验证工具进行介绍。

2.6.1 软件形式验证工具

目前软件形式化验证工具的主流技术有两种：基于抽象的技术路线和基于循环/递归展开的技术路线。前者利用抽象技术将程序状态空间映射到一个有限且规模较小的抽象状态空间，再通过遍历所有可达的抽象状态来判断系统是否正确。采用抽象路线的软件形式化验证工具包括 SLAM[476]、BLAST[472]、CPAChecker[475]、UAutomizer[477]等。后者将程序中所有循环和递归按给定的深度展开，得到一个不含循环和递归的简单程序。这类简单程序的验证问题被归结为一个逻辑公式的可满足性问题，再借助已有的 SMT 工具（如 Z3、MathSAT、Yices）进行求解。采用循环/递归展开技术路线的软件形式化验证工

具包括 CBMC[474]、ESBMC[479]、2LS[473]等。

以抽象解释为代表的程序分析工具近年来也得到了长足发展。相关工具不断涌现，出现了 PolySpace[480]、Astrée[481]、aiT WCET Analyzer[482]、CodeHawk[483]、Sparrow[484]、Zoncolan(Facebook)、Julia[485]等商业化工具和 Frama-C Value Analysis[486]、CCCheck(Code Contract Static Checker)[487]、Interproc[488]、crab-llvm[489]、IKOS[490]、MemCAD[491]、Fluctuat[492]、Jandom[493]、JsCFA[494]、MuJS[495]等学术界工具。Miné 等人基于 Astrée 开发了面向异步实时程序的扩展版本 AstréeA，以支持多线程 C 程序中运行时错误、数据竞争、死锁等错误的检测，并成功分析了 ARINC 653 航空应用（约 220 万行代码）[525-526]。

软件形式化验证工具的一个发展趋势是模型检测技术与抽象解释等程序分析技术的融合。Beyer 等人在 2007 年提出可配置程序分析（Configurable Program Analysis，CPA）[478]的思想，实现了静态分析与模型检测在一个理论框架下的融合。基于可配置程序分析的软件验证工具 CPAChecker 曾 4 次（2012、2013、2015 和 2018 年）夺得国际软件验证大赛总成绩的冠军。Heizmann 等人于 2013 年提出了一种融合自动机理论[477]的软件验证方法。他们基于该方法实现的 UAutomizer 工具连续夺得 2016 年和 2017 年国际软件验证大赛总成绩第一名。

作为程序的一种重要形态，并行程序验证成为目前形式化验证的一个热点领域。国际上出现了较多的并行程序验证工具。这些工具分别使用了偏序规约、限界模型检测、抽象精化、组合验证等技术，支持 PThread、OpenMP 以及 MPI 等并行框架，可以对常见静态属性和用户自定义断言属性进行验证。由于并行程序的行为空间异常复杂，目前这一领域的工具开发还处于学术研究阶段，国际上针对并行程序验证的主流工具有 CBMC[507]、ESBMC[479]、Lazy-CSeq[496]、SMACK[497]、以及 CPA-Seq[498]等。

随着软件形式化验证工具的日益完善，软件形式化验证在工业界软件，尤其是操作系统、驱动程序、嵌入式程序的分析与验证中得到了成功应用。耶鲁大学采用形式验证开发了一个全新的操作系统 CertiKOS[499]，号称是世界上第一个“没有 bug”的反黑客攻击操作系统。美国国防高级研究计划局（DARPA）在 2012～2017 年实施了一个 HACMS 项目（High-Assurance Cyber Military System）[500]，采用了形式验证方法，在开发一套开源的、高安全确保的操作系统和控制系统组件。微软研究院目前正和卡内基梅隆大学等合作，开展一项雄心勃勃的程序证明项目，即“珠穆朗玛峰（Everest）”项目[501]。亚马逊针对其云计算 IaaS 和 PaaS 平台的 S3 对象存储系统、DynamoDB NoSQL 数据库服务、EBS 弹性块存储服务等 10 个系统上使用了形式化验证，显著提高了系统的安全可靠性[502]。

2.6.2 混成系统验证工具

近年来，相关领域国际科研工作者在非线性混成自动机模型检测工具上投入了大量的精力并取得了一定进展。特别是泰勒模型（taylor model），支持函数（support function）等数学模型被应用到了混成系统状态域的表达与计算当中。近年出现的工具 Flow*[505]就是一个成功应用泰勒模型对非线性混成系统进行验证的示例。Flow*要求混成自动机的流条件必须由多项式微分方程描述。用户给定初始区间与一个固定的基本时间步之后，可利

用泰勒模型来分析它的可达区间。事实上，泰勒模型很适合连续状态的计算，但是当进行离散跳转时需要和转换卫式做相交操作，而这个操作的复杂度很高。与 Flow* 不同，SpaceEx[506] 允许系统中的不变式、迁移卫式等元素可以为凸函数。同样在给定一个基本时间步之后，可以利用 support function 来计算在此时间步之后的系统状态域。相关工具目前已经在部分非线性系统上进行了验证，但如何扩展可验证系统种类与规模，仍然是一项值得关注的问题。近年来出现的 CORA[503]、C2E2[504]、HyPro/HyDRA[510]、XSpeed[509] 等工具多采用了类似方法。

除了上述连续时间验证工具，Hylaa[507]、JuliaReach[508] 等离散时间非线性混成系统验证工具也于近年来开始出现并得到关注。而机器人、汽车设计等领域的工程设计人员也在 CPSWEEK、ESWEEK、SPIN 等多个重要场合提到了基于离散时间混成自动机进行验证与测试生成对实际项目开发与分析的重要意义。

上述工具主要侧重于非线性系统验证，在线性系统上，近年来的 HyComp[511]、Lyse[512] 等工具则是基于约束编码与求解手段的典型代表。由于当前约束求解主要集中于解决线性约束，非线性约束难以求解，HySAT/iSAT[513] 及 dReach[514] 提出了基于区间运算进行非线性行为 SMT 编码与求解，并在一些案例上取得较好结果。由于篇幅所限，其他以 KeYmaera[49] 为代表的混成系统定理证明工具等在此不再赘述。

2.7 量子程序分析与验证

传统计算机科学研究和发展的经验充分说明，要发挥量子计算机的超强计算能力，量子软件是必不可少的。同时，由于量子计算机和传统计算机在信息处理方面有着本质的区别（比如量子信息的不可克隆性和纠缠的非局域作用等特征），量子程序设计将是一个非常困难而容易犯错的过程，甚至需要和经典程序设计完全不同的思维方式。因此，一套行之有效的分析和验证技术对于保证量子程序的正确性显得尤为重要。另一方面，量子硬件设计与制造技术的飞速发展使得人们已经开始乐观预测多于 100 个量子比特的特定用途的量子计算机有望在 5～10 年内实现（最近谷歌展示了 72 个量子比特的计算机芯片）。到那时，量子程序设计和程序验证将成为真正发挥量子计算机作用的关键。

程序分析技术在编译器设计及程序优化等方面有着重要的应用。普林斯顿大学、加州大学圣巴拉拉分校等单位合作设计的量子程序语言 Scaffold[527] 的编译器 ScaffCC 中已经采用了数据流分析的一些方法，分析量子程序中的纠缠并检查程序是否违反量子不可克隆原理。应明生等研究了有限维希尔伯特空间中循环体为酉算子的量子循环程序的终止问题[528]，这一结果被进一步推广到循环体为一般超算子的情形，并提供了计算平均执行时间的一般性方法[529]。注意到微软新近推出的量子程序语言 Q#[530] 已经支持循环，这一工作可以用来分析 Q#程序的正确性。Perdrix 与 Jorrand[531-532] 将抽象解释技术引入量子程序分析，特别是程序中的纠缠及其演化的分析。Honda[533] 进一步分析了可用 stablizer formalism 描述的一类特殊量子程序中的纠缠。

量子程序验证研究的一个重要方面是发展适用于量子计算的程序逻辑。Brunet 与

Jorrand[534]提出了将 Birkhoff-von Neumann 量子逻辑应用于量子程序推理的方法；Baltag 与 Smets[535]发展了一种可用于量子系统中信息流形式化的动态逻辑；冯元等[536]发现了关于量子程序的一些有用的推理规则。Chadha、Mateus 与 Sernadas[537]给出了量子程序在只允许有界迭代的限制下的一个 Floyd-Hoare 型证明系统；Kakutani[538]试图将 den Hartog 的概率 Hoare 逻辑推广到量子情形，但这些尝试都不成功。应明生建立了一个真正完整的量子 Floyd-Hoare 逻辑，证明了其（相对）完备性[539]。值得指出的是，量子 Floyd-Hoare 逻辑完备性的证明用到了与经典 Floyd-Hoare 逻辑完备性的证明很不一样的方法，需要采用一些分析数学的技术。

模型检测是计算机系统，包括硬件和软件系统形式化验证的一种重要技术，由于其完全机械化和可以提供错误诊断信息，在工业界得到了广泛的应用。有趣的是，针对量子程序和协议的验证，我们有两种不同的量子马尔可夫链模型以及适用于各自模型的形式化验证方法。Ⅰ型量子马尔可夫链的状态空间为系统的所有可能量子状态的集合，模型的变迁由描述量子系统演化的超算子给出。对这种类型的量子马尔可夫链，其状态空间的 BSCC 分解、可达空间概率、重复可达概率、persistence 概率等问题都存在多项式时间算法[542-543]。与之对比，Ⅱ型量子马尔可夫链的状态空间和传统马尔可夫模型一样是完全经典的，因此在一般情形下也是有限的，但连接经典状态的转移概率被替换成描述量子系统变迁的（不保迹）超算子。这一模型可以非常方便地描述带经典控制流的量子软件系统。文献［272-273，544］中研究了Ⅱ型量子马尔可夫链模型下的各种可达性问题，并详细讨论了其对于 QCTL 分枝时序逻辑公式和 ω - 正则语言描述的性质的模型检测。通过将概率测度推广到以超算子为值的测度，Anticoli 等开发了一款工具[546]，可将量子程序语言 Quipper 的程序翻译到 II 型量子马尔可夫链，然后用 QPMC 进行验证。

3 国内研究进展

3.1 定理证明

国内在交互式定理证明方面的研究主要集中于证明辅助工具的应用研究上。

在操作系统和编译器方面，文献［40］在 Coq 中形式化了一个抢占式操作系统内核验证框架，并基于该框架首次验证了一个商业化抢占式实时操作系统内核 μC/OS-Ⅱ的关键功能的正确性；文献［41］在 Coq 中形式化了同步数据流语言 Lustre 的编译器，文献［48］在 Coq 中实现了同步语言 SIGNAL 的编译器；文献［43］对安全增强操作系统中一种常用的访问控制模型在 Isabelle/HOL 中进行了形式化，并验证了其安全性和完备性。

在混成系统验证方面，文献［73］在 Isabelle/HOL 中形式化了基于 HCSP 的混成系统建模和验证框架，实现了混成系统的定理证明器，并将其应用到高速列车和航天控制软件的安全性验证中。

在算法验证方面，文献［42］在 Isabelle/HOL 证明了判定正则语言的 Myhill-Nerode 定理的正确性；文献［44］对可计算性理论进行了形式化，证明了图灵机、算盘机和递归函数 3 种计算模型之间的等价性。

在形式化数学方面，文献［45-46］基于 HOL4 系统完成了工程数学中的很多理论的形式化证明，其中复数和 gauge 积分形式化理论库已被英国剑桥大学 HOL4 官方接收，除此之外，他们还研究了空间总线 SpaceWire 的形式化验证。

国内对于自动推理的研究起步较晚，但在某些方面取得了突出成果。

清华大学的周旻博士等人提出了一个名为“惰性分解与调解（lazy decomposition and conciliation）”的 SMT 并行求解框架，并基于 Z3 实现了工具 PZ3。PZ3 在具有稀疏结构的实例上通常可取得超线性的加速比，与国际上先进的 SMT 并行求解器 PCVC4 相比更加高效[62]。

SMT 公式的可满足性是判定问题，即判断一个给定的 SMT 公式是否有解。在很多情况下，不仅要知道解是否存在，还关心解空间的大小，因此有必要将 SMT 由判定问题扩展为计数问题。中科院软件所的研究人员首先研究了 SMT 公式的解计数/解空间体积计算问题（#SMT），并提出了一个基于 DPLL(T) 的精确计算方法——逐簇法[63]。由于 #SMT 问题具有#P 的复杂度，对于实际应用中的问题，估算是更为可行的方法。清华大学的周旻博士提出了一种 SMT(LRA) 解空间大小估算算法，结合了 BDD 的数据结构以及一种采用射线式采样的蒙特卡罗算法[64]。中科院软件所的葛存菁等人针对凸多面体的体积估算问题，改进了经典的多相位蒙特卡罗算法，可以快速估算出几十维的高维凸多面体体积，并基于此设计了 SMT(LRA) 的解空间大小估算方法[65]。对于位向量理论上的#SMT 问题和一般的#SAT 问题，葛存菁等人也提出了一个新的基于哈希函数的近似方法，可得到有理论保证的估算结果，并且其求解效率比国际上同类工具提升了 1 ~ 2 个数量级[66]。

此外，国防科技大学张建民、李思昆教授等研究了 SMT 的极小不可满足核问题，通过对极小不可满足核问题的检测，可为 SMT 公式的不可满足性提供精确的解释[67]。

3.2　形式模型

3.2.1　概率自动机模型

概率自动机及概率模型检测国内近五六年来也涌现了相当多的优秀的结果。概率模型检测方面，文献［107，121］给出了概率混成系统安全性质验证的算法和验证的框架，文献［108-109］给出了连续时间马尔科夫过程上连续随机逻辑验证的算法和工具，文献［110］给出了马尔科夫人口模型上连续随机逻辑验证的算法，文献［111］将概率模型检测中的模型、逻辑、算法进行整合，实现了中国范围内第一个大规模的模型检测工具 iscasMc，该工具由中国科学院软件研究所的张立军研究员主持开发和维护，在概率模型检测功能上做到了广而精。在概率自动机理论和概率模型上的互模拟理论方面，文献

［80，111-118］研究了各种概率模型中强互模拟、弱互模拟、模拟以及基于分布的互模拟等概念，并在逻辑刻画、度量刻画、算法优化等角度进行了深入研究，而文献［119-120］也在概率自动机理论和概率程序分析方面做出了突破。除此以外，在有关概率模型检测的逻辑理论中，文献［126］将线性时序逻辑中安全性（safety）和活性（liveness）的概念推广到概率分支时序逻辑上，在拓扑刻画、公式分解及安全性活性判定方面取得了突破性的成果，其他在连续时间概率逻辑中的突破工作可参见文献［127，129］，这些理论工作对概率模型检测的理论和方法研究都有着长足的作用。

3.2.2 混成自动机模型

近5年来，国内学者在混成自动机模型形式化验证方面取得了丰富的成果。文献［172］和文献［169，171］中分别提出了基于SAT-LP-IIS结合的混成自动机有界验证剪枝与编码技术，以及基于不可行路径的有界验证结论全局推导等技术，有效提高了可验证系统的规模与验证效率。针对半代数混成系统的概率验证问题，文献［170］基于栅栏函数的思想，提出了新型方法求取安全概率精确下界。文献［547］中提出了利用凸优化计算具有指数变化条件栅栏函数的方法对多项式混成模型进行了形式化验证。文献［175］通过释放栅栏函数的约束条件对文献［547］做了进一步的推广；文献［173］中提出了计算多项式混成模型不变量的方法。文献［174］研究了一类解析解可得的非线性模型的可达集计算问题。文献［144］将刻画非线性系统可达集的Hamilton-Jacobi偏微分方程的求解问题转化为凸优化问题，在此方法中，仅需求解一个凸优化就可同时得到可达集的高估计和低估计。

特别需要强调的是，国内学者近5年推动了时滞混成模型形式化验证的发展。文献［178］中提出了一种基于区间泰勒模型近似的方法，通过计算可达集的过高估计来验证一类简单时滞微分方程的安全性和稳定性。文献［152］提出了基于欧拉方法计算时滞微分方程可达集过高估计的数值方法，同时能够给出严格的误差上界。文献［153］中利用敏感性分析给出了一类其解具有同胚性质的时滞微分方程，将常微分方程中利用边界计算可达集的方法推广到了此类时滞微分系统，提出了计算其可达集的过高和过低估计的方法。文献［153］中记载了第一个计算时滞系统可达集低估计的工作。

3.2.3 无穷状态并发系统模型

国内上海交通大学BASICS实验室在此领域有突出的研究贡献，主要贡献包括证明了nBPA上的分支互模拟是可判定的，解决了一个长期公开的问题[178]，给出了进程重写系统中的9类模型上的分支互模拟的可判定与不可判定的分界线[179]。最近，又证明了nBPA上的分支互模拟是EXPTIME-完备的[180]，并且给出了一个nBPA上分支互模拟判定问题的指数时间下界。该归约构造了指数个冗余集以达到目标[181]。另外，研究了PDA及其扩展系统的等价和可达性问题的判定性结论。在PDA的等价性验证方面，证明了将内部动作限制在popping动作或pushing动作的PDA的分支互模拟是可判定的。这一结果比已知的PDA的互模拟等价的可判定性和语言等价的可判定性都强。再进一步，证明了这一结果在一定意义下是最强的。

3.3 形式语义与形式建模

形式语义方面最近5年的国内研究主要包括如下工作：文献［197］中定义了PTSC语言的指称语义，其中PTSC语言是一种具备概率、时间、共享变量通信并发的语言；文献［197，199］分别对PTSC和Verilog的操作语义和指称语义的连接理论进行了研究；文献［200］中定义了一种模式图建模语言MDM的操作语义；文献［201］中定义了一种概率中断建模语言pIML的指称语义；［202］定义SystemC的指称语义；［204］定义了一种Orc语言的指称语义；文献［203］中对Rust编程语言操作语义进行了研究，将大部分Rust语义成分的语义在K框架中实现，语义用Rust编译器压力测试程序进行了语义正确性的测试，并利用K框架的工具生成了Rust程序解释器、调试器和验证工具。

形式建模方面，最近5年国内的研究主要包括以下工作：

西安电子科技大学段振华教授的团队建立了一个并行时序逻辑的建模语言MSVL（Modeling，Simulation and Verification Language），它是PTL（Projection Temporal Logic）的一个可执行子集。他们定义了MSVL的模型语义、操作语义和公理语义，开发了相应的编译器。模型检测的一大难题就是系统模型的提取，使用MSVL可以直接对系统建模，得到MSVL程序，也可以将原有的系统，如C/Verilog/VHDL程序转换为MSVL程序[221-222]。为方便转换，开发了相应的转换工具C2M和V2M，从而形成了一个基于PPTL和MSVL的统一模型检测方法UMC（Unified Model Checkiing）。另外，为了缓解状态空间爆炸问题，该团队提出了几个很有创新性的理论和方法：①大幅度改进了CEGAR方法，将精化时间复杂度从指数阶降为多项式（平方阶），同时将反例路径检测算法时间复杂度从多项式降为线性的；②提出了基于Craig插值（interpolation）的安全插值和检错插值，大大缩减了静态分析的状态空间，提高了效率；③提出了一个基于MSVL的运行时程序验证方法：一个待验证的系统模型用MSVL程序M表达，无论直接编码或者转化得到，一个正则性质则用PPTL公式表达，该性质的非又可以通过自动转化工具P2M转为MSVL程序M'。对新程序M和M'进行编译，生成可执行代码。执行程序，如果得到false，则程序满足性质；如果得到true，一条执行路径便是一个反例。程序的输入怎么产生呢？可以通过两种方法：①应用DSE方法对M和M'进行符号约束，再应用Z3求解得到输入变量的初值集；②应用静态分析CEGAR方法得到反例路径，再执行程序检测反例的真假。为方便使用，开发了一个工具集MSV，包括建模工具、执行工具、验证工具、转换工具等。

中国科学技术大学提出了一种基于依赖/保证的模拟关系RGSim和一种霍尔风格的程序逻辑实现并发程序精化验证[223]。

文献［224］等提出利用模糊规则和Petri Net刻画自适应系统。文献［225］等提出基于控制理论的自适应软件建模方法。文献［226，227］等提出对安全威胁相关性质进行描述的时序逻辑DT-MTL，利用交错自动机生成运行时监控器来实现对无人飞行系统及无人机系统的运行时验证。

3.4 形式规约

南京大学赵建华、李宣东教授的研究团队，提出了分离逻辑的一种变种，称之为 Scope Logic[265]。该种逻辑中引入了 3 类基本算子，分别对应于基本内存访问、结构字段访问、数组访问操作。这种扩展可以较好地与命令式语言之间产生对应。

中科院软件所的吴志林等人对带归纳谓词的分离逻辑的判定算法进行了深入的研究。他们首先考虑了对 Cook 等人的单链表片段上的多项式时间的结果的扩展，证明了含有双链表片段的归纳谓词的分离逻辑的可满足性问题是 NP-完全的，而且他们将 Cook 等人的多项式时间的结果扩展到树结构上[266]。他们对于含有线性算术约束和可组合归纳谓词的分离逻辑子集的判定算法进行了深入的探讨[267-268]。

西安电子科技大学段振华教授的团队提出了投影时序逻辑 PTL，研究了命题 PTL（PPTL）的判定性、复杂性和表达性，建立了 PPTL 的一个合理的（soundness）且完备的（completeness）公理系统。由于 PPTL 是可判定的，而且它的表达能力是完全正则的，不但能够表达常用的 LTL 和 CTL 可以表达的时序性质，而且能够表达更重要的程序性质，例如区间相关的性质以及周期性重复的闭包性质。

华东师范大学的蒲戈光等人在 LTL 的可满足性问题方面展开了深入的探讨，提出了高效的判定算法，并开发了求解器[122-125,128]。

上海科技大学的宋富等人在文献［269］中证明了 ATL、AMC 和 ATL* 在下推多智能体系统上的模型检测问题分别是 EXPTIME-、2EXPTIME- 和 3EXPTIME- 完全的。在文献［270］中对策略逻辑 SL 的子类模型检测问题进行了研究，证明 SL[1G] 和 BSIL 子类模型检测问题分别是 3EXPTIME- 和 2EXPTIME- 完全的。

在时序逻辑的概率扩展方面，刘万伟、宋磊、王戟、张立军等人在文献［271］中提出了 PμTL，同 μ^P - 演算相比，它仅包含 $X \geq P$ 这样的概率时序算子。然而，它同样可以规约“区域安全”性质，并与 PCTL 的表达能力不兼容。他们证明了 PμTL 的可满足性问题属于 2EXPTIME-。

在时序逻辑的量子扩展方面，冯元等人给出了量子马尔可夫链上关于 QCTL 的模型检测算法[272]。随后，在文献［273］中，又给出了关于 ω - 正规性质的模型检测算法。

3.5 形式验证技术与方法

3.5.1 演绎推理

在针对实际系统的基于演绎推理的验证方面，清华大学团队在 Coq 中开展了 Lustre 语言编译器 L2C 的验证[311-313]。L2C 以扩展的 Lustre 语言为源语言，以 Clight（CompCert 中的 C 语言子集）为目标语言。编译器在 Coq 中实现并验证。L2C 可以和 CompCert 无缝衔接，最终实现从 Lustre 到汇编的可信编译。中科大团队对商业化嵌入式实时系统 μC/

OS-II 在 Coq 中开展验证[40]，完成了近 70% 常用 API 的验证。验证忠于系统原有代码的实现，对源代码改动极少。这是国际上首次对产品级抢占式并发内核的验证。北京航空航天大学团队和合作者对航空航天操作系统的事实标准 ARINC 653 进行了完整的形式化建模，并在模型层面进行了隔离性的验证[314]。解放军理工大学团队在 Isabelle/HOL 中对优先级继承协议、访问控制协议和文件比较算法等开展了验证[315,69,316]。

在程序逻辑方面，中科大团队先后提出一系列关系型并发程序精化验证逻辑，包括验证线性一致性的逻辑[294]、对无锁并发程序进展性验证的程序逻辑[317]、对阻塞算法进展性验证的程序逻辑 LiLi[318]及其扩展[319]等。中科院软件所团队对混成 CSP 提出了一系列程序逻辑[320-321]，并使用程序逻辑开展了对高铁控制系统的验证[176]。

3.5.2 抽象解释

国防科技大学的陈立前等人在数值抽象域的设计与实现、基于抽象解释的静态分析等方面开展了深入的研究：针对已有数值抽象域在表达能力方面存在的凸性局限性，提出了绝对值八边形抽象域[329]、区间线性模板约束抽象域[368]等非凸数值抽象域，以更精确地分析程序中的析取行为，减少误报；为了提高抽象解释分析的精度，提出了一种结合抽象域与 SMT 的块级抽象解释技术[375]；基于抽象解释提出了一种面向中断驱动型程序的可靠数值性质分析方法，以检查中断驱动型嵌入式软件中的数值相关运行时错误，并在实际航天工业代码上开展了示范应用[365-366]；在抽象解释框架下提出了一种通过结合形态抽象和数值抽象来分析链表操作程序的方法，以发现同时包含形态和数值信息的复杂程序性质[330]；提出了一种深度融合数组抽象和数组上动态数据结构（如数组内部维护的链表）抽象的方法，自动验证了 Minix、TinyOS 等操作系统内存管理、驱动程序等相关模块的正确性性质[344]。

北京邮电大学的金大海、宫云战等人基于抽象解释围绕区间抽象域扩展、抽象内存建模、缺陷关联等及其在缺陷检测中的应用开展了研究[336]，并把相关技术应用于缺陷检测工具 DTS 中。南京大学的潘建东等人针对显式和隐式含有析取语义的循环结构，提出了基于循环分解和归纳推理的不变式生成改进方法，缓解了抽象解释分析中出现的语义损失问题[369]。苏州大学陈冬火等人提出了一种基于凸多面体抽象域的自适应状态空间离散化方法，实现了自适应的基于凸多面体域的强化学习算法[367]。

3.5.3 模型检测

清华大学形式化验证团队研究基于屏障证书的混成系统验证，提出了一种更一般的、基于指数条件的屏障证书，可以为系统的可达集形成一个更加精确的上近似[547]。

清华大学贺飞等人研究基于假设/保证（assume/guarantee）规则的组合推理方法。将组合验证和符号化这两种应对模型检测状态空间爆炸的有效技术结合起来，提出了一个完全符号化的、基于假设 - 保证推理的组合验证方法[548]。他们还提出回归组合验证的思想，给出了基于假设 - 保证推理的回归组合验证框架，并证明了该框架的正确性和完备性[549]。该团队还研究了概率系统的组合验证问题，通过引入权重自动机，首次实

现了一个正确且完备的概率系统自动组合验证框架[550,276]。

清华大学贺飞等人开发了一个面向构件化系统的模型检测工具 Beagle。针对构件化系统的特点实现了专属的高效验证算法[277]，并成功应用于高铁控制设备、航空发动机控制软件的分析与验证中。

南京大学卜磊等提出了在路径求解中不可行子路径定位及状态空间剪枝的方法[377]，并将其应用到线性数值计算软件代码 BMC 中。南京大学团队在 LLVM 框架下，开发了相关验证工具 BRICK，在数值计算代码的有界模型检测上取得良好效果。

由国防科技大学研发的并行程序验证工具 Yogar-CBMC 在 SV-COMP 2017 和 2018 中连续两年获并行程序验证冠军。该工具结合限界模型检测、抽象精化和图分析等方法对并行程序的并行行为进行抽象，避免了限界模型检测过程中生成复杂的约束表达式。而西安电子科技大学田聪等提出了一个高效的反例路径检测算法和抽象模型精化方法，并将其在软件模型检测工具 CPAChecker 中使用，大幅度提高了程序验证的效率和可验证程序的规模[375-376]。

在时序性质验证方面，上海科技大学宋富等提出了顺序递归程序 CTL 性质模型检查算法，基于该算法设计实现的模型检查工具 PuMoC 可用于验证布尔程序、C/C ++ 和 Java 程序[373]，同时提出了两种时态逻辑 SCTPL 和 SLTPL 用于更加简洁地描述程序的性质及其程序栈的行为，设计实现了 SCTPL 和 SLTPL 模型检查工具 Pommade，并应用于二进制恶意软件的检测和 C 程序 API 约束检查[374]。

国防科技大学刘万伟等人针对采用线性规约的软件验证问题，提出了一种轻量级的“保持反例的规约化简”方法[372]。与传统的规约化简方法相比，它不要求保持规约的等价性，因此扩展了规约化简的范围。

Craig interpolant 生成在软件验证，特别是基于抽象的 CEGAR 验证框架中发挥着重要作用。中科院软件所詹乃军等提出首个非线性 Craig Interpolant 生成算法，将任何两个非线性公式的插值问题归结为两个半代数系统的插值问题，并进一步利用 Positivstellensatz 定理归结为半定规划问题而进行有效求解[370-371]。相关工作在非线性代码验证、模型检测、定理证明等方面具有重要意义。

3.5.4 符号执行

国内在符号执行方面开展研究的单位主要集中在中国科学院、南京大学、国防科技大学、武汉大学、华东师范大学、西安交通大学等一些单位。下面从缓解路径空间爆炸、约束求解、工具支撑等几个方面阐述国内的进展。

在缓解路径空间爆炸方面，南京大学的李游等人提出了基于子路径频谱覆盖频率的符号执行制导策略，指导符号执行优先探索程序中的罕至部分，以更好地覆盖程序并发现错误[460]。此外，他们还提出了基于目标的符号执行制导策略，用于确认警报的正确性，降低人工审查的开销。国防科技大学的张羽丰、陈振邦、王戟等人提出了正规性质（所有可用有限状态机表达的性质）制导的动态符号执行方法[463]，以引导符号执行过程更快地找到被分析程序中满足给定正规性质的路径；在此基础上，提出了基于符号执行

的正规性质符号化验证方法[407]，能有效削减程序的路径空间，提高验证效率。国防科技大学的陈立前等人基于符号执行技术提出了一种针对浮点程序鲁棒性的自动分析方法[456]。华东师范大学的苏亭、蒲戈光等人把软件模型检测技术与动态符号执行相结合，用于提高数据流测试的效率[447]。西安交通大学王海军等人提出了一种程序依赖引导的符号执行技术[421]，利用程序依赖关系引导符号执行选择性地探索程序路径，在保证和传统符号执行相同的错检测能力的情况下，约简执行重复程序行为的测试案例。

在约束求解方面，南京大学卜磊等提出了将基于机器学习的优化分析技术纳入符号执行框架进行约束求解的方法，通过后台猜值、前台执行确认的迭代方式对难解路径约束进行求解，在含复杂非线性约束、三方库函数调用等的复杂代码案例上取得良好实验效果[429]。国防科技大学的张羽丰、陈振邦、王戟等人提出了猜测符号执行方法[424]，以减少符号执行过程中求解器的调用次数。武汉大学贾向阳等提出了一种基于约束之间的逻辑子集/超集关系，可在符号执行中重用约束求解结果，以提高符号执行效率的方法[425]。

在工具支撑方面，中国科学院软件研究所许振波、张健等人开发了面向真实 C 语言程序的符号执行分析工具 Canalyze[457]，支持多种程序错误检测的跨过程分析框架。到目前为止，Canalyze 已经在一些重要的开源程序（例如 bftpd-3.8、httpd-2.4.4）中找到上百个程序错误。国防科技大学的张羽丰、陈振邦、王戟等人基于 JPF 开发了针对 Java 的动态符号执行工具，在开源 Java 程序上发现了多个缺陷；同时，开发了 C 程序的单元符号执行工具，支持包含结构体、动态数据结构、指针、浮点 C 程序的符号化分析，并在国防安全关键嵌入式控制程序上进行了应用。华东师范大学的苏亭、蒲戈光等人开发了 C 程序动态符号执行工具 CAUT，并在多个实际嵌入式系统上开展了应用。

3.6　形式验证工具以及应用

3.6.1　软件形式验证工具

在软件形式化验证工具的研发方面，清华大学形式化验证团队开发了一个面向 C 语言的程序验证工具 Ceagle，并基于 Ceagle 参加了国际软件验证大赛（SV-COMP），取得了 2017 年 3 个二级分支（ReachSafety：Array，ReachSafety：ECA，MemSafety：Array）第 1 名的成绩。由国防科技大学研发的并行程序验证工具 Yogar-CBMC 在 SV-COMP 2017 和 SV-COMP 2018 中连续两年获得并行程序验证冠军。该工具结合限界模型检测、抽象精化和图分析等方法对并行程序的并行行为进行抽象，避免了在限界模型检测过程中生成复杂的约束表达式。国防科技大学的陈立前等人基于抽象解释框架，采用开源编译器前端 CIL、开源数值抽象域库 APRON 实现了一个面向 C 程序的数值静态分析工具 CAI，能够分析 C 程序中变量的取值范围并检测除零错、数组越界、算术上溢、空指针解引用等运行时错误，并支持中断驱动型嵌入式软件的分析。

在软件形式化验证应用方面，国内取得了系列成果。中科院软件所研究并行程序的数据竞争和死锁检测[515]；大连理工大学研究多核限界模型检测[516]；清华大学研究面向

多核处理器的低级并行程序验证[517]；国防科技大学研究基于共享内存的并行程序验证[518]。清华大学研究面向车辆总线网络的运行时验证[519]。北京邮电大学基于抽象解释开发了缺陷检测工具[373]。

3.6.2 混成系统验证工具

国内在混成自动机验证工具上也取得系列成果。中科院软件所相关课题组在混成系统形式化方法研究中开发了 MARS 和 HHL 证明器等工具。MARS 主要用于混成系统的建模分析和验证[520]。HHL 证明器[521]是面向混成系统的交互式定理证明器。它以 Isabelle/HOL 为基础，形式化了以 HCSP(Hybrid CSP) 为建模语言，以时态演算为性质描述语言的混成 Hoare 逻辑，在此基础上实现了混成系统的验证框架。HHL 证明器主要用于验证基于 HCSP 的混成系统模型的正确性。这些工具应用到了高铁列控系统[176]和月球车导航与控制系统[177]等典型混成系统形式化验证上。

南京大学课题组所开发的混成自动机验证工具集 BACH[522]对单自动机、组合自动机等线性混成系统提供全面支持，可以对路径验证、有界验证、全局验证、乃至在线验证等问题进行快速判定[169]。BACH 在可处理问题规模与效率上较国际相关领域工具具有明显优势，在国际混成系统有界验证工具比赛有界验证 track 上连续两年获得第一，在国际相关领域引起关注并产生影响。基于工具集 BACH，南大课题组也开展了系列工业级实际系统应用探索。相关典型应用包括 CPS 系统在线验证工作[523]已经成功部署于列车控制国家工程中心硬件在环仿真平台。以 BACH 作为底层验证器的 IoT 验证修复工具原型“门神”系列，由于其在智能家居领域的重要前景，入选微软年度大会 TechFest，引起广泛关注[524]。

3.7 量子程序分析与验证

中国科学院软件研究所詹乃军研究组[540]基于 Isabelle/HOL 设计实现了一个量子 Floyd-Hoare 逻辑的定理证明器。为了提高其自动化程度，文献［541］中研究了量子程序不变式与 ranking 函数的生成算法，并将量子程序不变式生成问题转换为半正定规划问题。

中国科学院软件研究所张立军研究组和悉尼科技大学合作开发了一款模型检测工具 QPMC[545]。该工具使用了一种类似于概率模型检测工具 PRISM[88]的语法，可以对简单量子程序和协议的 QCTL 性质进行自动化验证。

4 国内外研究进展比较

4.1 定理证明

交互式定理证明在国内的起步较晚，一直到 2000 年以后，清华大学、华东师范大学

等相继举办 Coq 暑期班，它在国内形式化方法领域才逐渐得到推广应用。国外对于交互式定理证明的理论技术和应用两方面都进行了研究，而国内主要集中于交互式定理证明在形式化领域的应用研究。近年来，由于安全攸关软件的发展需求，国内关于交互式定理证明在系统验证等方向的应用研究明显增多，也取得了一些显著的成果，但距离国际领先水平仍存在些许差距。

国内外对 SMT 的研究，大致是因为国外起步较早，所以研究成体系，技术比较成熟。国内的研究起步较晚，但是在某些特定的研究方向上有一定优势。SMT 在国外的发展有很长的历史，主流的求解算法均产生在国外的研究机构。欧美的一些大学和科研机构成立有专门的 SMT 研究组，并开发出一系列成熟的求解工具。最具影响力的 SMT 求解器有 Z3、CVC4，其中微软在很长一段时间内处于领先地位。

相对来说，国内专门研究 SMT 的研究组较少，只有中科院、清华等少数科研院所和高校的科研人员从事这一研究。就研究内容而言，国内的研究工作重点放在一些新兴方向上面，所研发的求解器支持的理论较少。

4.2　形式模型

关于概率自动机模型的研究，绝大多数活跃在国外，国内对于概率自动机模型的研究尚还不够重视，活跃在这一领域的学者还是太少。一方面，我国概率模型检测的起步相对落后于西方，从 2010 年以后才陆续出现我国学者在这一领域发表成果；另一方面，概率自动机模型的研究往往是跨学科、跨背景的，需要一些背景知识和研究经验。不过令人欣喜的是，近些年来越来越多的中国学者在这一方向有所成果、我们也能时常在这一领域看到中国学者的名字。我们在概率自动机以及概率模型检测方面的脚步，已经有追赶上国外的趋势。纵观这一领域，某个重要的定义或者重要领域的发现和开发往往源于西方，而我国学者在之后能做出相当多的成果，这表明现在国内已经具有研究概率模型检测的能力和实力，但在创新性方面表现略显不足。

总体来讲，国内在混成自动机模型的研究方面处于国际前沿，与美国和欧洲等处于同一水平。近年来，中国科学院软件研究所、华东师范大学、南京大学、北京航空航天大学等团队在混成系统建模、验证、分析和设计等方面取得多项具有国际影响的成果，例如，中科院软件所詹乃军等解决了半代数集成为多项式混成系统不变式的充要条件问题，首次提出了时滞混成系统形式验证问题等；北京航空航天大学佘志坤等提出基于数值求解的混成系统可达集高效计算方法等。

在无穷状态并发系统模型的研究中，国内的研究进展与国外几个顶级实验室进展速度相当，多方之间在友好竞争当中，获得了越来越多的成果。

4.3　形式语义与形式建模

目前国内程序语言形式语义研究人员相比国外程序语言研究者偏少，研究力量不足，

归其原因是这方面的工作比较基础和冷门，入门门槛高且就业难。

在操作语义和指称语义方面，国外大部分对工业级程序语言的语义进行研究，对语义通常会在定理证明器或 K 框架中实现，实现的语义会进行测试并用于程序的分析和验证；对语义的研究不仅仅为了更好地理解程序语言，而是为了更好地对程序进行分析和验证。与之相反，国内程序语言的语义研究，主要（除 Rust 外）是针对一些学术性语言，还停留在形式语义研究初期的目的，即为了更好地理解程序语言，而非对这些语义进行应用。

尽管国内形式建模方面的研究在近 5 年有较为快速的发展，也有大量的论文发表，然而与国外研究相比，国内的研究多是在国际上提出的形式模型的基础上，做一些扩展性研究工作，缺少原创性工作。另外，与国外许多相对成熟的建模与验证工具相比，如 SPIN[228]、PAT[229]、K 框架[190]、UPPAAL[211]等，国内开发的形式建模工具多是已有工具的扩展，缺少独立开发的原创性工具，导致在国际上的影响较小。

另一方面，形式化方法作为解决工业应用领域系统安全可靠性问题的重要技术，在国内缺少比较有影响力的成功案例。国际上，有很多有影响力的案例使得产业界对形式化建模与验证有足够的重视，如 Event-B 曾成功用于巴黎地铁的设计与开发[205]，亚马逊也将 TLA + 用于 Web Service 的设计与开发[230]。然而，国内产业界由于形式化方法过于抽象、扩展性等问题，对形式化的重视力度不够，在系统的实际设计与开发过程中未充分发挥形式化方法的作用，缺少具有影响力的成功案例。

4.4 形式规约

国内从事形式规约的研究人员不多，目前流行的形式规约语言，比如分离逻辑、时序逻辑等，基本是国外的研究人员提出来的。国内研究人员在形式规约语言的研究工作集中在对已有规约语言的理论性质（比如表达能力和判定算法）的探讨上。国内研究人员最近 5 年在分离逻辑和时序逻辑的各种扩展上开展了较深入的工作，总体来讲较以前有所进步。

4.5 形式验证技术与方法

与国际研究现状相比，国内在基于演绎推理的验证技术方面工作相对较少，特别是在新型验证理论和程序逻辑方面的工作相对较少。关注的程序性质主要集中在功能正确性方面，对于其他性质（如信息流控制等安全性）的验证工作较少。在并发程序精化验证方面，特别是并发程序的活性验证方面的工作处于国际领先的地位。

我国在实际系统方面的验证工作较为活跃，但很多工作集中在模型和算法层面，对于代码的验证工作相对较少。与国际同行相比，验证的系统的规模和影响力上面有所欠缺。

近年来，国外在抽象解释的理论、方法、工具等方面形成了许多有影响力的成果，也形成了较多有影响力的开源和商业化的有效工具平台，并且有些工具在工业界取得了

成功应用。国内在新型抽象域的设计与实现、特定领域应用（如中断驱动型程序的分析）方面研究形成特色。然而，国内在基于抽象解释的程序分析工具研发、产业影响等方面还与国外存在一定差距，相信随着国内软件可信需求的不断提升，差距会不断缩小。

在模型检测方面，近年来，国外在软件形式化验证的理论、方法和技术等方面产生了大量有影响力的成果，形成了许多高效的软件形式化验证工具。国内在特定领域的研究形成特色，已经达到国际领先的水平，如国防科技大学研发的并行程序验证工具 Yogar-CBMC 等。

在符号执行方面，总体而言，国内在符号执行各个方面的研究正逐渐与国外接轨，包括理论、方法与工具等，国内相关研究学者的工作也能发表在主流的符号执行相关的国际会议或期刊上，这些工作也引起了国外学者的注意，同时与国外相关学者的交流与合作也越来越多。然而，国内在符号执行的工具和平台的开发、产业影响方面还与国外存在一定差距，相信随着国内软件行业自主可控需求的不断提升，这部分的差距会不断缩小。

4.6　形式验证工具以及应用

近 10 年来，国外在软件形式化验证的理论、方法和技术等方面产生了大量有影响力的成果，形成了许多高效的软件形式化验证工具。国内在软件形式化验证工具开发方面起步较晚，通用工具的研发与国外仍存在较大差距。然而，国内在某些特定领域的研究形成特色，已经达到国际领先的水平，如国防科技大学研发的并行程序验证工具 Yogar-CBMC 等。

在混成自动机验证工具方面，国内外研究各有侧重与特色。国内工作在混成系统定理证明、线性混成自动机有界验证等领域处于国际领先地位。而国外近年来则在非线性混成自动机有界可达性检验方面取得系列进展，开发了以 Flow* 、SpaceEx 等为代表的系列工具。

4.7　量子程序分析与验证

从量子程序的分析与验证来讲，国内与国外基本处于同一水平，国内中科院软件所应明生教授团队在量子程序的分析与验证方面处于国际引领地位。

5　发展趋势与展望

5.1　定理证明

定理证明本质上可以看作一个证明搜索问题。由于一般使用的逻辑的不可判定性，一个能够高效率地解决所有证明问题的算法是不存在的。因此启发法（heuristic）的使用

尤其重要。目前交互式定理证明里的自动化在一般情况下还远远不及人类的能力，其中主要问题是人在证明中使用的有目的的搜索还无法在计算机上实现。机器学习，尤其是深度学习的方法为实现这种有目的的搜索提供了新的希望。文献［34］中首次把递归神经网络运用到定理证明问题上。在这项工作中，机器学习被用于使用自动定理证明器之前筛选已有定理（premise selection），但没有用于指导证明搜索本身。对使用各种机器学习方法指导证明搜索本身也开始有一些研究，比如文献［36-38］。使用机器学习的一个主要问题是如何获取大量的可供学习的数据。现在使用的一些主要的数据库来自 Mizar 和 HOL Light，各包括几万个定理。更大的数据库还有待建立。除了定理证明本身，机器学习也开始应用于非形式化数学到形式化数学的自动转换[35]。

当前基于 DPLL(T) 框架的主流 SMT 求解技术已经趋于成熟，但尚不能完全满足现实应用的需要，在一些特定的发展方向上仍有很大的研究空间，包括 SMT 的扩展问题、非完备求解算法等。

SMT 本身为判定问题，但科学研究和现实世界中的很多问题不仅关心解的存在性，而且要进一步探寻最优解，乃至解空间的大小，因此对 SMT 问题类型的扩展近年来引起了国际学界的重视。上文所提到的 SMT 解计数问题即是一个重要的 SMT 扩展问题，在程序分析与验证、近似推理等领域有广泛的应用前景。另一个热门的扩展方向是将 SMT 由判定问题扩展到优化问题，即优化模理论（optimization modulo theory）。优化模理论问题旨在求出满足 SMT 约束的最优解，在实际应用中很有意义。目前代表性的研究来自于意大利 Trento 大学[68-69]。

作为一个 SMT 求解的完备算法框架，DPLL(T) 虽然已取得了巨大的成功，但本质上是基于回溯框架的指数级复杂度的算法，其求解效率难以持续突破，在求解规模上很受限制。近年来，在 SAT 领域，基于局部搜索的 SAT 算法在工业化的例子上取得了较大成功，与基于 CDCL 的完备算法形成了并驾齐驱的局面。在 SMT 领域，当前已有非完备的 SMT 求解算法出现，主要用于 BV 理论。非完备算法在设计上具有灵活性，并对问题的规模不敏感，会是将来 SMT 求解算法的一个发展趋势。

5.2 形式模型

概率自动机模型不仅在概率模型检测上有重要的理论价值和应用场景，在概率程序语义和验证［91，99］、随机算法［115，72，105］甚至机器学习领域[74,102]也都有着广泛的应用。这表明，概率自动机模型不仅在形式化验证领域是一个有研究价值的热门话题，在计算机的其他领域，甚至数学、物理、生物、医学等其他自然科学领域，也都有着相当广泛的研究前景，包括概率自动机模型及其相关理论、验证问题和应用都有广泛的研究空间和很高的研究价值。经过了近 30 年对概率模型检测的研究，其理论基础已十分坚实，很多重要问题已经得到解决，但同时不乏一些重要的公开问题（如 PCTL 公式可满足性的判定问题）仍然存在。这一领域仍有很多虽然看起来不大却依然重要的问题有待解决和开发，同时该领域与计算机科学其他领域及其他自然科学的结合也必将成为

新的研究方向。

混成自动机模型的研究取得了一定的进展，但仍有许多问题亟待解决。未来 5 年，混成自动机模型的研究将集中在以下几个方面：随机和概率混成系统的形式验证，时滞混成系统的形式验证，大规模非线性混成动态模型的形式验证，开放动态模型的行为预测与验证。

无穷状态并发系统模型研究方面取得了很大的进展，有关强互模拟的判定问题大部分已解决，而当允许内部迁移时，互模拟的可判定性结果很少。所以着重解决 2.2.3 节表 1 中列出的悬而未决的公开问题，是本领域的趋势。因为这些问题长期公开，所以用已有的证明方法很难得到结果，如何开发新的证明方法，是这一领域的关键所在。

5.3　形式语义与形式建模

从目前国内外程序语言语义的研究来看，未来程序语言语义的研究将针对工业级的程序语言，而非学术性无实际应用的语言；语义研究的目的也不在仅仅为了更好地理解程序语言，而是为了更好地对程序进行分析与验证。

另一方面，多年前就寄望于程序语言设计者在设计语言时对程序的语义进行形式化定义的问题尚未解决。虽然在学术性程序语言提出时，设计者会给出其形式语义，但是对于工业级程序语言，比如 C、Rust 和 Go，设计者都没有定义形式语义，甚至没有完整的形式化语法定义。主要原因是工业级程序语言相对复杂，语义的定义困难，具备这方面能力的工业界人员少，也没有易用的语义开发框架。

新型计算系统如大数据系统、人工智能系统、无人驾驶系统等的出现为形式建模技术的发展带来新的机遇和挑战。系统的智能性、实时性、空间离散性等特征对系统的安全性和可靠性提出更高的要求，迫切需针对这些系统的特性发展新的建模理论、方法和工具，这将成为形式化方法领域研究的热点问题。然而随着系统的复杂性越来越高、特性越来越多，很难定义一种统一的建模方法来完整地描述系统的所有特性。如何在模型的表达能力与验证问题的可判定性及复杂度等方面做出取舍，都是非常具有挑战性的工作。

5.4　形式规约

新的计算模式比如大数据、机器学习算法等不断涌现。这些计算模式下的程序的行为与经典的串行和并发程序非常不同，需要设计新的规约语言以及分析验证方法与技术。针对这些新的计算模式的形式规约语言将是今后 5 年左右的研究热点。

5.5　形式验证技术与方法

近年来，伴随着云计算、人工智能等各种研究热点，在相关领域中开展的验证也越

来越得到人们的重视。在云计算和分布式系统验证方面，逐步出现了一些对分布式算法和系统的验证工作[303-304]。但其中对分布式数据一致性协议和算法的验证尚处于起步阶段，国内外相关工作较少。

人工智能算法和演绎推理技术的结合是另外一个很有发展前景的方向。国内外已经有了初步的采用模型检测或者其他技术来对人工智能算法进行验证的工作，但基于演绎推理的验证工作较少。这里的主要困难在于，基于演绎推理的验证技术需要对正确性有形式化定义（什么是正确），同时验证过程本身要求明确验证对象工作的原理（为什么正确）。而以上这两点在人工智能研究领域尚未有明确答案。

未来，抽象解释技术将进一步在新的架构、语言、应用等实际需求驱动下不断发展，值得关注的方向包括对弱内存模型的分析验证[335,322]、神经网络的分析与验证[339,364]、大数据处理相关错误的分析[353]、Python 程序的自动分析[338]等。与约束求解、自动推理、人工智能等基础支撑技术的紧密结合[358,350,328,341,357]，将是抽象解释后续研究趋势之一[333]。同时，降低误报率将依然是基于抽象解释的程序分析技术拓展实际应用的研究挑战和重点。

在模型检测部分，我们主要侧重于一般软件代码近年来相关验证工作的介绍。在此之外，近年来相关领域关注热点进一步拓展到针对多线程代码验证，对递归等特定类型程序验证，对领域相关代码验证等。除了上述安全性（safety）/可达性验证（reachability）之外，软件代码的活性（liveness）/可终止性（termination）验证也是近年来关注热点所在。

在符号执行方面，符号执行技术将进一步在软件工程、安全、系统、网络等相关领域的实际需求驱动下不断发展；面向大规模软件的高效符号执行方法、技术和工具将是下一步的研究挑战和重点；同时，符号执行搜索策略的更加智能化也将是下一步的研究重点；此外，与其他技术在不同层面的密切结合，以进一步提高软件分析效果，也将是符号执行后续的研究趋势之一。

5.6 形式验证工具以及应用

软件形式化验证面临严重的状态空间爆炸问题。软件形式化验证工具研发的一个重要趋势是多技术融合。利用形式化方法的严格性和其他技术的可伸缩性，力求在验证规模和验证精度上达成协调和统一。

混成系统由于其行为离散、连续交织非常复杂，难以掌握与控制，现阶段可进行有效分析的系统种类和规模仍有一定限制。当前主要关注问题包括大规模组合非线性混成系统验证、开放动态系统行为预测与验证、概率随机行为建模与验证、混成系统控制生成以及混成系统测试生成等。相关领域科研人员力图在上述问题上取得突破，力图对实际系统规模问题具备分析能力，从而保障相关系统运行安全。

5.7 量子程序分析与验证

总的来说，虽然量子程序的分析与形式化验证领域已经取得了一些可喜的进展，但

目前的研究还非常零散，很多问题甚至还不清楚如何准确定义。这大体上有两方面的原因：

1）尽管最近几年量子硬件设备的物理实现取得了长足的进展，但距离能够运行真正实用的量子程序的通用量子计算机仍然非常遥远。因此绝大部分传统计算机科学家以及程序设计和验证方面的专家还持观望态度，并没有对这一领域给予足够多的重视。

2）由于量子程序和传统计算机程序相比具有很大的不同，特别是由于量子叠加和纠缠的存在，量子程序的验证往往非常困难。但是，我们有理由相信，量子程序理论和验证的研究将会吸引越来越多计算机科学家的关注，从而带动这一领域蓬勃发展。

6 结束语

形式化方法是计算机科学的传统研究方向，其涵盖很多不同的研究方向。本报告对形式化方法各个方面的近 5 年的研究进展进行了相对全面的总结，对发展趋势进行了展望。希望本报告能够让计算机科学与技术的其他领域的研究人员和 IT 产业界的从业人员更好地了解形式化方法这个学科的内涵、外延以及国内外研究现状，促进形式化方法跟其他领域的交叉融合，促使形式化方法在 IT 产业界得到更多应用。

参考文献

[1] J C B, Cezary Kaliszyk, Lawrence C Paulson, Josef Urban. Hammering towards QED[J]. Journal of Formalized Reasoning, 2016, 9(1): 101-148.

[2] J C B, Sascha Böhme, Lawrence C Paulson. Extending Sledgehammer with SMT solvers[J]. Journal of Automated Reasoning, 2013, 51(1): 109-128.

[3] L C Paulson. A mechanised proof of Gödel' s incompleteness theorems using Nominal Isabelle[J]. Journal of Automated Reasoning, 2015, 55(1): 1-37.

[4] J Avigad, J Hölzl, Luke Serafin. A formally verified proof of the Central Limit Theorem[J]. Journal of Automated Reasoning, 2017, 59(4): 389-423.

[5] A Mahboubi, G Melquiond, T Sibut-Pinote. Formally verified approximation of definite integrals[C]. ITP 2016, LNCS 9807, 2016: 274-289.

[6] F Immler, C Traut. The Flow of ODEs[C]. ITP 2016, LNCS 9807, 2016: 184-199.

[7] F Immler. A Verified ODE Solver and the Lorenz Attractor[J]. Journal of Automated Reasoning, 2018, 61(1-4): 73-111.

[8] J Holzl, A Heller, Three Chapters of Measure Theory[C]. ITP 2011, LNCS 6898, 2011: 135-151.

[9] J Holzl. Markov chains and Markov decision processes in Isabelle/HOL[J]. Journal of Automated Reasoning, 2017, 59(3): 345-387.

[10] M Abdulaziz, L C Paulson. An Isabelle/HOL Formalization of Green' s Theorem[C]. ITP 2016, LNCS

9807, 2016: 3-19.

[11] J Blanchette, N Peltier, S Robillard. Superposition for datatypes and codatatypes[C]. IJCAR 2018, LNCS 10900, 2018: 370-387.

[12] A Bentkamp, J Blanchette, S Cruanes, U Waldmann. Superposition for lambda-free higher-order logic [C]. IJCAR 2018, LNCS 10900, 2018: 28-46.

[13] B Bohrer, V Rahli, I Vukotic, M Volp, A Platzer. Formally verified differential dynamic logic[C]. CPP 2017, 2017: 208-221.

[14] W Li, L Paulson. A formal proof of Cauchy's residue theorem[C]. ITP 2016, LNCS 9807, 2016: 235-251.

[15] W Li, G O Passmore, L C Paulson. Deciding Univariate Polynomial Problems Using Untrusted Certificates in Isabelle/HOL[J/OL]. Journal of Automated Reasoning. http://doi.org/10.1007/s10817-017-9424-6, 2017.

[16] D Matichuk, T C Murray, M Wenzel. Eisbach: A Proof Method Language for Isabelle[J]. Journal of Automated Reasoning, 2016, 56(3): 261-282.

[17] L de Moura, S Kong, J Avigad, F van Doorn, J von Raumer. The Lean Theorem Prover[C]. CADE 2015. LNAI 9195, 2015: 378-388.

[18] G Ebner, S Ullrich, J Roesch, J Avigad, L de Moura. A Metaprogramming Framework for Formal Verification[C]. ICFP 2017, 2017: 34: 1-34: 29.

[19] J C Blanchette, S Bohme, A Popescu, N Smallbone. Encoding Monomorphic and Polymorphic types[C]. TACAS 2013. LNCS 7795. 2013: 493-507.

[20] J C Blanchette, D Greenaway, C Kaliszyk, D Kuhlwein, J Urban. A Learning-Based Fact Selector for Isabelle/HOL[J]. Journal of Automated Reasoning, 2016, 57(3): 219-244.

[21] C Kaliszyk, J Urban. Learning-assisted Automated Reasoning with Flyspeck[J]. Journal of Automated Reasoning, 2014, 53(2): 173-213.

[22] L Czajka, C Kaliszyk. Hammer for Coq: Automation for Dependent Type Theory[J]. Journal of Automated Reasoning, 2018: 61(1-4): 423-453.

[23] Alexandria[OL]. http://www.cl.cam.ac.uk/~lp15/Grants/Alexandria/.

[24] R Thiemann, A Yamada. Formalizing Jordan Normal Forms in Isabelle/HOL[C]. CPP 2016, 2016: 88-99.

[25] Divasón, S Joosten, O Kunčar, R Thiemann, A Yamada. Efficient Certification of Complexity Proofs - Formalizing the Perron-Frobenius Theorem[C]. CPP 2018, 2018: 2-13.

[26] A Bentkamp, J C Blanchette, D Klakow. A formal proof of the expressiveness of deep learning[C]. ITP 2017, LNCS 10499, 2017: 46-64.

[27] J Biendarra, J C B, A Bouzy, M Desharnais, M Fleury, J Hölzl, O Kunčar, A Lochbihler, F Meier, L Panny, A Popescu, C Sternagel, R Thiemann, D Traytel. Foundational (co)datatypes and (co)recursion for higher-order logic[C]. FroCoS 2017, LNCS 10483, 2017: 3-21.

[28] J C B, Fabian Meier, A Popescu, D Traytel. Foundational nonuniform (co)datatypes for higher-order logic[C]. LICS 2017, 2017: 1-12.

[29] S Awodey, Á Pelayo, M A Warren. Voevodsky's Univalence Axiom in homotopy type theory[J]. Notices of the American Mathematical Society, 2013, 60(9): 1164-1167.

[30] D Licata, M Shulman, Calculating the fundamental group of the circle in homotopy type theory[C]. LICS 2013, 2013: 223-232.

[31] B Zhan. Formalization of the fundamental group in untyped set theory using auto2[C]. ITP 2017, LNCS 10499, 2017: 514-530.

[32] Gonthier, G, Asperti, A, Avigad, J, Bertot, Y, Cohen, C, Garillot, F, Le Roux, S, Mahboubi, A, OConnor, R, Biha, SO, et al.: A machine-checked proof of the odd order theorem[C]. In ITP 2013, LNCS 7998, 2013: 163-179.

[33] T Hales, M Adams, G Bauer, T Dang, J Harrison, L T Hoang, C Kaliszyk, V Magron, S Mclaughlin, T T Nguyen, Q T Nguyen, T Nipkow, S Obua, J Pleso, J Rute, A Solovyev, T Ta, N T Tran, T D Trieu, J Urban, K Vu, R Zumkeller. A formal proof of the Kepler conjecture[J]. Forum of Mathematics. 2017.

[34] G Irving, C Szegedy, A A Alemi, N Eén, F Chollet, J Urban. DeepMath - Deep Sequence Models for Premise Selection[C]. NIPS 2016, 2016: 2243-2251.

[35] C Kaliszyk, J Urban, J Vyskocil. Automating Formalization by Statistical and Semantic Parsing of Mathematics[C]. ITP 2017, LNCS 10499, 2017: 12-27.

[36] S Loos, G Irving, C Szegedy, C Kaliszyk. Deep Network Guided Proof Search[C]. LPAR 2017, EPiC 46, 2017: 85-105.

[37] T Gauthier, C Kaliszyk, J Urban TacticToe: Learning to Reason with HOL4 Tactics[C]. LPAR 2017, EPiC 46, 2017: 85-105.

[38] M Farber, C Kaliszyk, J Urban. Monte Carlo Tableau Proof Search[C]. CADE 2017. LNCS 10395. 2017: 563-579.

[39] T V H. Prathamesh. Formalizing Knot Theory in Isabelle/HOL[C]. ITP 2015. LNCS 9236, 2015: 438-452.

[40] F Xu, M Fu, X Feng, X Zhang, H Zhang, Z Li. A Practical Verification Framework for Preemptive OS Kernels[C]. In CAV 2016, part II, 2016: 59-79.

[41] G Shi, Y Gan, S Shang, S Wang, Y Dong, P Yew. A formally verified sequentializer for lustre-like concurrent synchronous data-flow programs[C]. ICSE (Companion Volume) 2017, 2017: 109-111.

[42] C Wu, X Zhang, C Urban. A Formalisation of the Myhill-Nerode Theorem Based on Regular Expressions [J]. Journal of Automated Reasoning, 2014, 52(4): 451-480.

[43] C Wu, X Zhang, C Urban. A Formal Model and Correctness Proof for an Access Control Policy Framework[C]. CPP 2013, 2013: 292-307.

[44] J Xu, X Zhang, C Urban. Mechanising Turing Machines and Computability Theory in Isabelle/HOL[C]. ITP 2013, LNCS 7998, 2013: 147-162.

[45] Shi Z, Li L, Guan Y, et al. Formalization of the Complex Number Theory in HOL4[J]. Applied Mathematics & Information Sciences, 2013, 7(1): 279-286.

[46] Shi Z, Gu W, Li X, Guan Y, Ye S, Zhang J. The gauge integral theory in HOL4[J]. Journal of Applied Mathematics, 2013: 1-7.

[47] N Zhan, S Wang, H Zhao. Formal Verification of Simulink/Stateflow Diagrams: A Deductive Approach [M]. Berlin: Springer, 2017.

[48] Yang, Z, Bodeveix, JP, Filali, M et al. Towards a verified compiler prototype for the synchronous language SIGNAL[J]. Front. Comput. Sci., 2016, 10(1): 37-53.

[49] A Platzer, J Quesel. KeYmaera: A hybrid theorem prover for hybrid systems[C]. IJCAR 2008, LNCS

5195, 2008: 171-178.

[50] D Kroening, O Strichman, Decision Procedures—An Algorithmic Point of View[M]. Berlin: Springer, 2008.

[51] 金继伟, 马菲菲, 张健, SMT 求解技术简述 [J]. 计算机科学与探索, 2015, 9(7): 769-780.

[52] SMT-COMP[OL]. http://www.smtcomp.org/.

[53] C Barrett, C L Conway, M Deters, L Hadarean, D Jovanovic, T King, A Reynolds, C Tinelli. CVC4 [C]. CAV 2011, LNCS 6806, 2011: 171-177.

[54] B Dutertre. Yices 2.2[C]. CAV 2014, LNCS 8559, 2014: 737-744.

[55] Lde Moura, N Bjørner. Z3: An Efficient SMT Solver[C]. TACAS 2008, LNCS 4963, 2008: 337-340.

[56] R Nieuwenhuis, A Oliveras, C Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T)[J]. Journal of ACM 53(6), 2006: 937-977.

[57] A Fröhlich, A Biere, C M Wintersteiger, Y Hamadi. Stochastic Local Search for Satisfiability Modulo Theories[C]. AAAI 2015, 2015: 1136-1143.

[58] A Niemetz, M Preiner, A Biere. Precise and Complete Propagation Based Local Search for Satisfiability Modulo Theories[C]. CAV 2016, LNCS 9779, 2016: 199-217.

[59] A Niemetz, M Preiner, A Biere. Propagation based local search for bit-precise reasoning[J]. Formal Methods in System Design, 2017, 51(3): 608-636.

[60] M Bromberger, C Weidenbach. Fast Cube Tests for LIA Constraint Solving[C]. IJCAR 2016, LNCS 9706, 2016: 116-132.

[61] Z Fu, Z Su. XSat: A Fast Floating-Point Satisfiability Solver[C]. CAV 2016, LNCS 9780, 2016: 187-209.

[62] Xi Cheng, Min Zhou, Xiaoyu Song, Ming Gu, Jiaguang Sun, Parallelizing SMT solving: Lazy decomposition and conciliation[J]. Artificial Intelligence 257, 2018: 127-157.

[63] F Ma, S Liu, J Zhang, Volume Computation for Boolean Combination of Linear Arithmetic Constraints [C]. CADE 2009, LNCS 5663, 2009: 453-468.

[64] M Zhou, F He, X Song, S He, G Chen, M Gu. Estimating the Volume of Solution Space for Satisfiability Modulo Linear Real Arithmetic[J]. Theory of Computing Systems, 2015, 56(2): 347-371.

[65] Cunjing Ge, Feifei Ma, Peng Zhang, Jian Zhang. Computing and estimating the volume of the solution space of SMT(LA) constraints[J/OL]. Theoretical Computer Science. http://doi.org/10.1016/j.tcs.2016.10.019.

[66] Cunjing Ge, Feifei Ma, Tian Liu, Jian Zhang, Xutong Ma, A New Probabilistic Algorithm for Approximate Model Counting[C]. IJCAR 2018, LNCS 10900, 2018: 312-328.

[67] Jianmin Zhang, ShengYu Shen, Jun Zhang, Weixia Xu, Sikun Li. Extracting minimal unsatisfiable subformulas in satisfiability modulo theories[J]. Comput. Sci. Inf. Syst, 2011, 8(3): 693-710.

[68] Roberto Sebastiani, Patrick Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories[C]. CAV 2015, LNCS 9206, 2015: 447-454.

[69] Roberto Sebastiani, Silvia Tomasi. Optimization Modulo Theories with Linear Rational Costs[J]. ACM Transactions on Computational Logic, 2015, 16(2): 12: 1-12: 43.

[70] Michael O Rabin. Probabilistic Automata[J]. Inf4ormation and Control, 1963, 6 (3): 230-245.

[71] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, Robert Brayton. Model checkingcontinuous time markov chains[J]. ACM Transactions on Computational Logic, 2000, 1(1): 162-170.

[72] Christel Baier. Quantitative Analysis of Randomized Distributed Systems and Probabilistic Automata[M]. Springer, 2013.

[73] Christel Baier, Boudewijn R Haverkort, Holger Hermanns, Joost Pieter Katoen. Model checking continuous-time markov chains by transient analysis[C]. CAV 2000, LNCS 1855, 2000: 358-372.

[74] Borja Balle, Jorge Castro. Adaptively learning probabilistic deterministic automatafrom data streams[M]. Berlin: Kluwer Academic Publishers, 2014.

[75] Richard Bellman. A markovian decision process[J]. Indiana University Mathematical Journal, 1957, 6(4): 679-684.

[76] Andrea Bianco, Luca De Alfaro. Model checking of probabalistic and nondeterministicsystems[C]. FSTTCS 1995, LNCS 1026, 1995: 499-513.

[77] R E Bryant. Graph-based algorithms for boolean function manipulation[J]. IEEE transactionon computers, 1986, 35(8): 677-691.

[78] Costas Courcoubetis. The complexity of probabilistic verification[J]. Journal of the ACM, 1995, 42(4): 857-907.

[79] Josee Desharnais, Abbas Edalat, Prakash Panangaden. A logical characterizationof bisimulation for labeled markov processes[C]. LICS 1998, 1998: 478-487.

[80] Christian Eisentraut, Jens Chr. Godskesen, Holger Hermanns, Lei Song, LijunZhang. Probabilistic bisimulation for realistic schedulers[C]. FM 2015, LNCS 9109, 2015: 248-264.

[81] Yuan Feng, Lijun Zhang. When equivalence and bisimulation join forces in probabilisticautomata[C]. FM 2014, LNCS 8442, 214: 247-262.

[82] M Fruth. Probabilistic model checking of contention resolution in the IEEE 802.15.4low-rate wireless personal area network protocol[C]. ISOLA 2006, 2006: 290-297.

[83] M Fruth. Formal Methods for the Analysis of Wireless Network Protocols. PhD thesis[D]. Oxford: Oxford University, 2011.

[84] M Fujita, P C Mcgeer, C Y Yang. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation[M]. Berlin: Kluwer Academic Publishers, 1997.

[85] Jan Friso Groote, David N Jansen, Jeroen J A Keiren, Anton Wijs. An O(mlogn)algorithm for computing stuttering equivalence and branching bisimulation[J]. ACMTransaction on Computational Logic, 2017, 18(2): 13: 1-13: 34.

[86] Hans Hansson, Bengt Jonsson. A logic for reasoning about time and reliability[J]. Formal Aspectsof Computing, 1994, 6(5): 512-535.

[87] Holger Hermanns, Jan Krcal, Jan Kretinsky. Probabilistic Bisimulation: Naturallyon Distributions[M]. Berlin: Springer, 2014.

[88] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, David Parker. PRISM: A tool for automatic verification of probabilistic systems[C]. TACAS 2006, LNCS 3920, 2006: 441-444.

[89] Ronald A Howard. Dynamic programming and markov processes[J]. MathematicalGazette, 1960, 3(358): 136.

[90] Ron Koymans. Specifying real-time properties with metric temporal logic[J]. Real-TimeSystems, 1990, 2(4): 255-299.

[91] Dexter Kozen. Semantics of probabilistic programs[C]. FOCS 1979, 1979: 101-114.

[92] M Kwiatkowska, G Norman, D Parker, J Sproston. Performance analysis ofprobabilistic timed automata

using digital clocks[J]. Formal Methods in System Design, 29, 2006: 33-78.

[93] M Kwiatkowska, G Norman, J Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol [C]. PAPM/PROBMIV 2002, LNCS 2399, 2002: 169-187.

[94] M Kwiatkowska, G Norman, J Sproston. Probabilistic model checking of deadlineproperties in the IEEE 1394 FireWire root contention protocol[J]. Formal Aspects of Computing, 2003, 14(3): 294-318.

[95] M Kwiatkowska, G Norman, J Sproston, F Wang. Symbolic model checking forprobabilistic timed automata[C]. FORMATS/FTRTFT 2004, LNCS 3253, 2004: 293-308.

[96] M Kwiatkowska, G Norman, J Sproston, F Wang. Symbolic model checking for probabilistic timed automata[J]. Information and Computation, 2007, 205(7): 1027-1077.

[97] M Kwiatkowska, G Norman, J Sproston, F Wang. Symbolic Model Checking for Probabilistic Timed Automata[J]. Information and Computation, 2007, 205(7): 1027-1077.

[98] K G Larsen, A Skou. Bisimulation through probabilistic testing[J]. Information and Computation, 1989, 94(1): 344-352.

[99] A Legay, A S Murawski, J Ouaknine, J Worrell. On automatedverification of probabilistic programs[C]. TACAS 2008, LNCS 4963, 2008: 173-187.

[100] J Ouaknine, J Worrell. Some recent results in metric temporal logic[C]. FORMATS 2008, LNCS 5215, 2008: 1-13.

[101] P Panangaden. Metrics for labelled markov processes[J]. Theoretical ComputerScience, 2004, 318(3): 323-354.

[102] D Ron, Y Singer, N Tishby. The power of amnesia: learning probabilisticautomata with variable memory length[J]. Machine Learning, 1996, 25(2-3): 117-149.

[103] V Sassone, M Nielsen, G Winskel. Models for concurrency: Towards a classification[J]. Theoretical Computer Science, 1996, 170(1-2): 297-348.

[104] R Segala. Modeling, Verification of Randomized Distributed Realtime Systems[D]. Cambridge: MIT, 1995.

[105] M Swaminathan, J Katoen, E-R Olderog. Layered reasoningfor randomized distributed algorithms[J]. Formal Aspects of Computing, 2012, 24(4-6): 477-496.

[106] N Urabe, I Hasuo. Generic forward and backward simulations III: quantitativesimulations by matrices [C]. CONCUR 2014, LNCS 8704, 2014: 451-466.

[107] L Zhang, Z She, S Ratschan, H Hermanns, E Hahn. Safety verification for probabilistic hybrid systems [J]. European Journal of Control, 2012, 18(6): 305-587.

[108] Y Gao, E Hahn, N Zhan andL Zhang. CCMC : A Conditional CSL Model Checker for Continuous-Time Markov Chains[C]. ATVA 2013, LNCS 8172, 2013: 464-468.

[109] Y Gao, M Xu, N Zhan, L Zhang. Model checking conditional CSL for continuous-time markov chains [J]. Information Processing Letters, 2013, 113(1-2): 44-50.

[110] D Spieler, E Hahn, L Zhang. Model Checking CSL for Markov Population Models[C]. QAPL 2014, EPTCS 154, 2014: 93-107.

[111] E Hahn, Y Li, S Schewe, A. Turrini, L. Zhang. iscasMc: A Web-Based Probabilistic Model Checker [C]. FM 2014, LNCS 8442, 2014: 312-317.

[112] D N Jansen, L Song, Zhang. Revisiting weak simulation for substochastic markov chains[C]. QEST 2013, LNCS 8054, 2013: 209-224.

[113] C E isentraut, H Hermanns, A Turrini, L Zhang. Deciding bisimilarities on distributions[C]. QEST

2013, LNCS 8054, 2013: 72-88.

[114] L Song, L Zhang, J C Godskesen. Bisimulations Meet PCTL Equivalences for Probabilistic Automata [C]. CONCUR 2011, LNCS 6901, 2011: 108-123.

[115] L Song, L Zhang, J C Godskesen. Bisimulations and Logical Characterizations on Continuous-Time Markov Decision Processes[C]. VMCAI 2014, LNCS 8318, 2014: 98-117.

[116] L Song, L Zhang, H Hermanns, J C Godskesen. Incremental bisimulation abstraction refinement[J]. ACM Transactions on Embedded Computing Systems, 2014, 13(4s): 1-23.

[117] L Zhang, D N Jansen. A space-efficient simulation algorithm on probabilistic automata[J]. Information and Computation249, 2016: 138-159.

[118] P Yang, D N Jansen, L Zhang. Distribution-Based Bisimulation for Labelled Markov Processes[C]. FORMATS 2017, LNCS 10419, 2017: 170-186.

[119] C Eisentraut, H Hermanns, J Schuster, A Turrini, L Zhang. The Quest for Minimal Quotients for Probabilistic Automata[C]. TACAS 2013, LNCS 7795, 2013: 16-31.

[120] Y Feng, L Zhang, D N Jansen, N Zhan, B Xia. Finding Polynomial Loop Invariants for Probabilistic Programs[C]. ATVA 2017, LNCS 10482, 2017: 400-416.

[121] S Wang, N Zhan, L Zhang. A Compositional Modelling and Verification Framework for Stochastic Hybrid Systems[J]. Formal Aspects of Computing, 2017, 29(4): 751-775.

[122] J Li, L Zhang, G Pu, M Y Vardi, J He. LTL Satisfiability Checking Revisited[C]. TIME 2014, 2014: 91-98.

[123] J Li, G Pu, L Zhang, Z Wang, J He, K G Larsen. On the relationship between LTL normal forms and Büchi automata[C]. Theories of Programming and Formal Methods, LNCS 8051, 2013: 256-270.

[124] J Li, Y Yao, G Pu, L Zhang, J He. Aalta: an LTL satisfiability checker over Infinite/Finite traces[C]. FSE 2014, 2014: 731-734.

[125] J Li, L Zhang, G Pu, M Y Vardi, J He. Ltlf satisfiability checking[J]. Computer Science, 2014, 12(2): 123-137.

[126] J P Katoen, L Song, L Zhang. Probably safe or live[C]. CSL-LICS 2014, 2014: 55: 1-55: 10.

[127] Y Peng, S Wang, N Zhan, L Zhang. Extending Hybrid CSP with Probability and Stochasticity[C]. SETTA 2015, LNCS 9409, 2015: 87-102.

[128] J Li, L Zhang, S Zhu, G Pu, M Y Vardi, J He. An explicit transition system construction approach to LTL satisfiability checking[J]. Formal Aspects of Computing, 2017, 30(2): 1-25.

[129] L Zhang, D N Jansen, F Nielson, H Hermanns. Automata-Based CSL Model Checking[C]. ICALP 2011, LNCS 6756, 2011: 271-282.

[130] T A Henzinger. The Theory of Hybrid Automata[C]. LICS 1996, 1996: 278-292.

[131] R David. Modeling of hybrid systems using continuous and hybrid Petri nets[C]. PNPM 1997, 1997: 47-58.

[132] A Platzer. Differential dynamic logic for hybrid systems[J]. Journal of Automated Reasoning, 2008, 41(2): 143-189.

[133] T A Henzinger, P H Ho, H Wong-Toi. HYTECH: A model checker for hybrid systems[C]. CAV 1997, LNCS 1254, 1997: 460-463.

[134] G Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech[C]. HSCC 2005, LNCS 3414, 2005: 258-273.

[135] E Asarin, T Dang, O Maler. d/dt: A tool for reachability analysis of continuous and hybrid systems [C]. IFAC Nonlinear Control Systems, IFAC Proceedings, 34(6): 741-746, 2001.

[136] G Frehse G, C Le Guernic, A Donzé, S Cotton, R Ray, O Lebeltel, R Ripado, A Girard, T Dang, O Maler. SpaceEx: Scalable verification of hybrid systems[C]. CAV 2011, LNCS 6806, 2011: 379-395.

[137] X Chen, E Abraham, S Sankaranarayanan. Flow *: An analyzer for non-linear hybrid systems[C]. CAV 2013, LNCS 8044, 2013: 258-263.

[138] B Xue, A Easwaran, N Cho, M Fränzle. Reach-Avoid Verification for Nonlinear Systems Based on Boundary Analysis[J]. IEEE Transactions on Automatic Control, 2016, 62(7): 3518-3523.

[139] B Xue, M Fränzle, P N Mosaad. Just scratching the surface: Partial exploration of initial values in reach-set computation[C]. CDC 2017, 2017: 1769-1775.

[140] B Xue, Z She, A Easwaran. Under-Approximating Backward Reachable Sets by Polytopes[C]. CAV 2016, 2016: 457-476.

[141] H Didier, K Milan. Convex computation of the region of attraction of polynomial control systems[J]. IEEE Transactions on Automatic Control, 2014, 59(2): 297-312.

[142] M Shankar, V Ram. Convex computation of the reachable set for hybrid systems with parametric uncertainty[C]. ACC 2016, 2016: 5141-5147.

[143] B Xue, M Fränzle, N Zhan. Under-Approximating Reach Sets for Polynomial Continuous Systems[C]. HSCC 2018, 2018: 51-60.

[144] H Kong, EBartocci, T A Henzinger. Reachable Set Over-Approximation for Nonlinear Systems Using Piecewise Barrier Tubes[C]. CAV 2018, 2018: 449-467.

[145] O Bokanowski, N Forcadel, H Zidani. Reachability and minimal times for state constrained nonlinear problems without any controllability assumption[J]. SIAM Journal on Control and Optimization, 2010, 48(7): 4292-4316.

[146] J F Fisac, M Chen, C J Tomlin andS S Sastry. Reach-avoid problems with time-varying dynamics, targets and constraints[C]. HSCC 2015, 2015: 11-20.

[147] S Prajna, A Jadbabaie. Methods for safety verification of time-delay systems[C]. CDC 2005, 2005: 4348-4353.

[148] Z Huang, C Fan, S Mitra. Bounded invariant verification for time-delayed nonlinear networked dynamical systems[J]. Nonlinear Analysis: Hybrid Systems, 2017, 23: 211-229.

[149] L Zou, M Fränzle, N Zhan, P N Mosaad. Automatic verification of stability and safety for delay differential equations[C]. CAV 2015, 2015: 338-355.

[150] P N Mosaad, M Fränzle, B Xue. Temporal Logic Verification for Delay Differential Equations[C]. ICTAC 2016, 2016: 405-421.

[151] M Chen, M Fränzle, Y Li, P N Mosaad, N Zhan. Validated simulation-based verification of delayed differential dynamics[C]. FM 2016, 2016: 137-154.

[152] B Xue, P N Mosaad, M Fränzle, M Chen, Y Li, N Zhan. Safe Over-and Under-Approximation of Reachable Sets for Delay Differential Equations[C]. FORMATS 2017, 2017: 281-299.

[153] Goubault Eric, Putot Sylvie, Sahlmann Lorenz. Inner and Outer Approximating Flowpipes for Delay Differential Equations[C]. CAV 2018, 2018: 523-541.

[154] R Mayr. Process rewrite systems[J]. Information and Computation, 2000, 156(1-2): 264-286.

[155] J C M Baeten, J A Bergstra, J W Klop. Decidability of bisimulation equivalencefor processes generating

context-free languages[C]. PARLE Parallel Architectures andLanguages Europe, LNCS 259, 1987: 94-111.

[156] S Christensen, Y Hirshfeld, F Moller. Bisimulation equivalence isdecidable for basic parallel processes [C]. CONCUR 1993, LNCS 715, 1993: 143-157.

[157] Y Hirshfeld, M Jerrum. Bisimulation equivalence is decidable for normed process algebra[C]. ICALP 1999, LNCS 1644, 1999: 412-421.

[158] C Stirling. Decidability of bisimulation equivalence for normed pushdown processes[C]. CONCUR 1996, LNCS 1119, 1996: 217-232.

[159] P Jan car. Undecidability of bisimilarity for Petri nets and some related problems[J]. Theoretical Computer Science, 1995, 148(2): 281-301.

[160] J Balcúzar, J Gabarró, M Sántha. Deciding bisimilarity isP-complete[J]. Formal Aspects of Computing, 1992, 4(1): 638-648.

[161] Y Hirshfeld, M Jerrum, F Moller. A polynomial-time algorithm for decidingequivalence of normed context-free processes[C]. FOCS 1994, 1994: 623-631.

[162] Y Hirshfeld, M M Jerrum, F Moller. A polynomial-time algorithmfor deciding bisimulation equivalence of normed Basic Parallel Processes[J]. Theoretical Computer Science, 1996(259): 1-8.

[163] S Kiefer. BPA bisimilarity is EXPTIME-hard[J]. Information Processing Letters, 2013, 103(4): 101-106.

[164] P Jan car. Strong bisimilarity on basic parallel processes in PSPACE-complete[C]. LICS 2003, 2003: 218-227.

[165] M Benedikt, et al. Bisimilarity of Pushdown Automata is Nonelementary[C]. LICS 2013, 2013: 488-498.

[166] R van Gabbeek, W Weijland. Branching time and abstraction in bisimulationsemantics[J]. Journal of the ACM, 1996, 43(3): 555-600.

[167] W Czerwinski, P Hofman, S Lasota. Decidability of Branching Bisimulation on Normed Commutative Context-Free Processes[C]. CONCUR 2011, LNCS 6901, 2011: 528-542.

[168] W Czerwiński, P Jan car. Branching Bisimilarity of Normed BPA ProcessesIs in NEXPTIME[C]. LICS 2015, 2015: 168-179.

[169] D Xie, W Xiong, L Bu, X Li. Deriving Unbounded Reachability Proof of Linear Hybrid Automata during Bounded Checking Procedure[J]. IEEE Transactions on Computers, 2017, 66(3): 416-430.

[170] C Huang, X Chen, W Lin, Z Yang, X Li. Probabilistic Safety Verification of Stochastic Hybrid Systems Using Barrier Certificates[J]. ACM Transactions on Embedded Computing Systems, 2017, 16(5): 186: 1-186: 19.

[171] D Xie, L Bu, X Li. Deriving Unbounded Proof of Linear Hybrid Automata from Bounded Verification [C]. RTSS 2014, 2014: 128-137.

[172] D Xie, L Bu, J Zhao, X Li. SAT-LP-IIS Joint-directed Path-oriented Bounded Reachability Analysis of Linear Hybrid Automata. FMSD 2014, 2014: 45(1): 42-62.

[173] Q Wang, Y Li, B Xia, N Zhan. Generating semi-algebraic invariants for non-autonomous hybrid systems [J]. Journal of System Science and Complexity, 2017, 30(1): 234-253.

[174] T Gan, M Chen, Y Li, B Xia, N Zhan. Reachability analysis for solvable dynamical systems[J]. IEEE Transaction on Automatic Control, 2018, 63(7): 2003-2018.

[175] L Dai, T Gan, B Xia, N Zhan. Barrier certificates revisited[J]. Journal of Symbolic Computation, 2017, 80: 62-86.

[176] L Zou, J Lv, S Wang, N Zhan, T Tang, L Yuan, Y Liu. Verifying Chinese train control system under a combined scenario by theorem proving[C]. VSTTE 2013, LNCS 8164, 2013: 262-280.

[177] M Yang, N Zhan. Combining Formal and Informal Methods in the Design of Spacecrafts[C]. SETSS 2014, LNCS 9506, 2014: 290-323.

[178] Y Fu. Checking Equality and Regularity for Normed BPA with Silent Moves[C]. ICALP 2013, Part II, LNCS 7966, 2013: 244-255.

[179] Q Yin, Y Fu, C He, M Huang, X Tao. Branching BisimilarityChecking for PRS[C]. ICALP 2014, Part II, LNCS 8573, 2014: 363-374.

[180] C He, M Huang. Branching Bisimilarity on Normed BPA Is EXPTIME-Complete[C]. LICS 2015, 2015: 180-191.

[181] M Huang, Q Yin. Two Lower Bounds for BPA[C]. CONCUR 2017, LIPIcs 85, 2017: 20: 1-20: 16.

[182] C Ellison, G Rosu, An executable formal semantics of C with applications[C]. POPL 2012, 2012: 533-544.

[183] Y Gurevich, J K Huggins. The semantics of the C programming language[C]. In CSL 1993, LNCS 702, 1993: 274-308.

[184] J V Cook, E L Cohen, T S Redmond. A formal denotational semantics for C[R]. Technical Report 409D, Trusted Information Systems, 1994.

[185] J V Cook, S Subramanian. A formal semantics for C in Nqthm[R]. Technical Report 517D, Trusted Information Systems, 1994.

[186] M Norrish. C formalised in HOL[R]. Technical Report UCAM-CL-TR-453, University of Cambridge, 1998.

[187] M Norrish. A formal semantics for C++[R]. Technical report, NICTA, 2008.

[188] N S Papaspyrou. Denotational semantics of ANSI C[J]. Computer Standards and Interfaces, 2001, 23(3): 169-185.

[189] S Blazy, X Leroy. Mechanized semantics for the Clight subset of the C language[J]. Journal of Automated Reasoning, 2009, 43(3): 263-288.

[190] G Rosu, T-F Serbanuta. An overview of the K semantic framework[J]. The Journal of Logic and Algebraic Programming, 2010, 79(6): 397-434.

[191] D Bogdanas, G Rosu. K-Java: A Complete Semantics of Java[C]. POPL 2015, 2015: 445-456.

[192] D Park, A Stefanescu, G Rosu. KJS: a complete formal semantics of JavaScript[C]. PLDI 2015, 2015: 346-356.

[193] D Filaretti, S Maffeis. An Executable Formal Semantics of PHP[C]. ECOOP 2014, LNCS 8586, 2014: 567-592.

[194] D Guth. A formal semantics of Python 3.3[D]. Champaign: University of Illinois at Urbana-Champaign, 2013.

[195] S Kan, D Sanan, S Lin, Y Liu, K-Rust: An executable formal semantics for Rust[OL]. CoRR, vol. abs/1804.07608, 2018.

[196] P O'Hearn, J Reynolds, H Yang. Local reasoning about programs that alter data structures[C]. CSL 2001, LNCS 2142, 2001: 1-19.

[197] H Zhu, F Yang, J He, J Bowen, J Sanders, S Qin: Linking operational semantics and algebraic semantics for a probabilistic timed shared-variable language[J]. The Journal of Logic and Algebraic Programming, 2012, 81(1): 2-25.

[198] H Zhu, J Sanders, J He, S Qin, Denotational Semantics for a Probabilistic Timed Shared-Variable Language[C]. UTP 2012, LNCS 7681, 2012: 224-247.

[199] H Zhu, P Liu, J He, S Qin. Mechanical Approach to Linking Operational Semantics and Algebraic Semantics for Verilog Using Maude[C]. UTP 2012, LNCS 7681, 2012: 164-185.

[200] M Yang, Z Wang, G Pu, S Qin, B Gu, J He. The stochastic semantics and verification for periodic control systems[J]. SCIENCE CHINA Information Sciences, 2012, 55(12): 2675-2693.

[201] Y Huang, Y Zhao, S Qin, J He. Probabilistic Denotational Semantics for an Interrupt Modelling Language[C]. ICECCS 2015, 2015: 160-169.

[202] H Zhu, J He, S Qin, P Brooke. Denotational semantics and its algebraic derivation for an event-driven system-level language[J]. Formal Aspects of Computing, 2015, 27(1): 133-166.

[203] F Wang, F Song, M Zhang, X Zhu, J Zhang. KRust: A Formal Executable Semantics of Rust [C]. TASE, 2018.

[204] Q Li, Y Zhao, H Zhu, J He. A UTP semantic model for Orc language with execution status and fault handling[J]. Frontiers of Computer Science, 2014, 8(5): 709-725.

[205] J Abrial. Modeling in Event-B: System and Software Engineering[M]. Cambridge: Cambridge University Press, 2013.

[206] J Peterson. Petri Net Theory and the Modeling of Systems[M]. New Jersey: Prentice Hall, 1981.

[207] H Comon, M Dauchet, R Gilleron, C Loding, F Jacquemard, D Lugiez, S Tison, M Tommasi. Tree Automata Techniques and Applications[OL]. http://www.grappa.univ-lille3.fr/tata.

[208] T Cachat. Higher Order Pushdown Automata, the Caucal Hierarchy of Graphs and Parity Games[C]. ICALP 2003, LNCS 2719, 2003: 556-569.

[209] L Ong. On Model-Checking Trees Generated by Higher-Order Recursion Schemes[C]LICS 2006, 2006: 81-90.

[210] N Kobayashi. Higher-Order Model Checking: From Theory to Practice[C]. LICS 2011, 2011: 219-224.

[211] K G Larsen, P Pettersson, W Yi. Uppaal in a nutshell[J]. International Journal on Software Tools for Technology Transfer, 1997, 1(1-2): 134-152.

[212] M Faugere, T Bourbeau, S Gerard, et al. MARTE: Also an UML Profile for Modeling AADL Applications[C]. ICECCS 2007, 2007: 359-364.

[213] G Booch, J Rumbaugh, I Jacobson. The Unified Modeling Language user guide[M]. New Jersey: Addison-Wesley, 2001.

[214] S Kuno andK Fujino. An Application of formal language Z in Software engineering[J]. Technical Report of Ieice Kbse, 1994, 94: 25-32.

[215] M Beal. The infinite hidden Markov model[J]. NIPS 2001, 2011: 577-584.

[216] P Arcaini, E Riccobene, P Scandurra. Formal Design and Verification of Self-Adaptive Systems with Decentralized Control[J]. ACM Transactions on Autonomous & Adaptive Systems, 2017, 11(4): 1-35.

[217] M Camilli, A Gargantini, P Scandurra. Zone-based formal specification and timing analysis of real-time self-adaptive systems[J]. Science of Computer Programming, 2018(159): 28-57.

[218] R Lemos, H Giese, H Müller, et al. Software Engineering for Self-Adaptive Systems. A Second Research Roadmap[C]. Software Engineering for Self-Adaptive Systems II. LNCS 7475, 2013: 1-32.

[219] X Huang, M Kwiatkowska, S Wang, et al. Safety Verification of Deep Neural Networks[C]. CAV 2017, LNCS 10426, 2017: 3-29.

[220] G Katz, C Barrett, D Dill, et al. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks [C]. CAV 2017, LNCS 10426, 2017: 97-117.

[221] Y Yu, Z Duan, C Tian, et al. Model Checking C Programs with MSVL[C]. SOFL 2012, LNCS 7787, 2012: 87-103.

[222] 段振华, 张康, 田聪, 等. 一种硬件语言 VHDL 到 MSVL 的自动转换系统 [P]. CN 104503816 A, 2015.

[223] H Liang, X Feng, M Fu. Rely-Guarantee-Based Simulation for Compositional Verification of Concurrent Program Transformations[J]. ACM Transactions on Programming Languages & Systems, 2014, 36(1): 1-55.

[224] Z Ding, Y Zhou, M Zhou. Modeling Self-Adaptive Software Systems By Fuzzy Rules and Petri Nets[J]. IEEE Transactions on Fuzzy Systems, 2018, 26(2): 967-984.

[225] W Yang, C Xu, M Pan, et al. Efficient validation of self-adaptive applications by counterexample probability maximization[J]. Journal of Systems and Software, 2018(138): 82-99.

[226] G Zhou, W Dong, W Liu, et al. Optimizing Monitor Code Based on Patterns in Runtime Verification [C]. QRS 2017 Companion, 2017: 348-354.

[227] 刘斌斌, 刘万伟, 毛晓光, 等. 无人驾驶汽车决策系统的规则正确性验证 [J]. 计算机科学, 2017, 44(4): 72-74.

[228] G Holzmann. The model checker SPIN[J]. IEEE Transactions on Software Engineering1997, 23(5): 279-294.

[229] J Sun, Y Liu, J Dong. Model Checking CSP Revisited: Introducing a Process Analysis Toolkit[C]. ISOLA 2008, CCIS 17, 2008: 307-322.

[230] C Newcombe, T Rath, F Zhang, et al. How Amazon web services uses formal methods [J]. Communications of the ACM, 2015, 58(4): 66-73.

[231] C A R Hoare. An Axiomatic Basis for Computer Programming [J]. Communications of the ACM, 1969, 12(10): 576-583.

[232] J C Reynolds. Speparation Logic: A Logic for Shared Mutable Data Structures[C]. LICS 2012, 2012: 55-74.

[233] B Cook, C Haase, J Ouaknine, M Parkinson, J Worrell. Tractable Reasoning in a Fragment of Separation Logic[C]. CONCUR 2011, LNCS 6901, 2011: 235-249.

[234] T Antonopoulos, N Gorogiannis, C Haase, M Kanovich, J Ouaknine. Foundations for Decision Problems in Separation Logic with General Inductive Predicates [C]. FoSSaCS 2014, LNCS 8412, 2014: 411-425.

[235] R Iosif, A Rogalewicz, J Simacek. The tree width of separation logic with recursive definitions[C]. CADE 2013, LNCS 7898, 2013: 21-38.

[236] J Brotherston, C Fuhs, J A N Perez, N Gorogiannis. A decision procedure for satisfiability in separation logic with inductive predicates[C]. LICS 2014, 2014: 25: 1-25: 1.

[237] R Iosif, A Rogalewicz, T Vojnar. Deciding entailments in inductive separation logic with tree automata

[C]. ATVA 2014, LNCS 8837, 2014: 201-218.

[238] W N Chin, C David, H H Nguyen, S Qin. Automated verification of shape, size and bag properties via user-defined predicates in separation logic[J]. Science of Computer Programming, 2012, 77(9).

[239] P Madhusudan, X Qiu, A Stefanescu. Recursive proofs for inductive tree data-structures[C]. POPL 2012, 2012: 123-136.

[240] Q L Le, J Sun, W-N Chin. Satisfiability modulo heap-based programs[C]. CAV 2016, LNCS 9779, 2016: 382-404.

[241] M Tatsuta, Q L Le, W Chin. Decision procedure for separation logic with inductive definitions and Presburger arithmetic[C]. APLAS 2016, LNCS 10017, 2016: 423-443.

[242] R Piskac, T Wies, D Zufferey. Automating separation logic using SMT[C]. CAV 2013, LNCS 8044, 2013: 773-789.

[243] R Piskac, T Wies, D Zufferey. Automating separation logic with trees and data[C]. CAV 2014, LNCS 8559, 2014: 711-728.

[244] C Enea, M Sighireanu, Z Wu. On automated lemma generation for separation logic with inductive definitions[C]. ATVA 2015, LNCS 9364, 2015: 80-96.

[245] A Pnueli. The Temporal Logic of Programs[C]. FOCS 1977, 1977: 46-57.

[246] A Prasad Sistla, Edmund M Clarke. The Complexity of Propositional Linear Temporal Logics[J]. Journal of ACM, 1985, 32(3): 733-749.

[247] E A Emerson, E M Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints[C]. ICALP 1980, LNCS 85, 1980: 169-181.

[248] E M Clarke, E A Emerson, A P Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1986, 8(2): 244-263.

[249] Vardi, M Y, L Stockmeyer. Improved upper and lower bounds for modal logics of programs[C]. STOC 1985, 1985: 240-251.

[250] M J Fischer, R E Ladner. Propositional modal logic of programs[J]. Journal of Computer and Systems Sciences, 1979(18): 194-211.

[251] E A Emerson, Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back[J]. Science of Computer Programming, 1987, 8(3): 275-306.

[252] R Alur, TA Henzinger, O Kupferman. Alternating-Time Temporal Logic[J]. Journal of ACM, 2002, 49(5): 672-713.

[253] K Chatterjee, TA Henzinger, N Piterman. Strategy Logic[J]. Information and Computation, 2007, 208(6): 677-693.

[254] Dirk Walther, Carsten Lutz, Frank Wolter, Michael Wooldridge, ATL Satisfiability is Indeed ExpTime-complete[J]. Journal of Logic and Computation, 2006, 16(6): 765-787.

[255] Sven Schewe. ATL* Satisfiability Is 2EXPTIME-Complete[C]. ICALP 2008, 2008: 373-385.

[256] F Mogavero, A Murano, G Perelli, M Y Vardi. Reasoning About Strategies: On the Model-Checking Problem[J]. ACM Transactions on Computational Logic, 2014, 15(4): 34: 1-34: 47.

[257] M Y Vardi. Automatic verification of probabilistic concurrent finite-state programs[J]. FOCS 1985, 1985: 327-338.

[258] M Y Vardi, P Wolper. Reasoning about infinite computations[J]. Information and Computation, 1994,

115(1): 1-37.

[259] M Y Vardi. Probabilistic linear-time model checking: An overview of theautomata-theoretic approach [C]. ARTS 1999, LNCS 1601, 1999: 265-276.

[260] R Alur, C Courcoubetis, D Dill, Model-checking in dense real-time[J]. Information and Computation, 1993(104): 2-34.

[261] T A Henzinger. The temporal specification and verification of real-time systems [D]. Palo Alto: Technical Report STAN-CS-91-1380, Stanford University, 1991.

[262] R Alur, T A Henzinger. Real-time logics: complexity and expressiveness [J]. Information and Computation, 1993(104): 35-77.

[263] Clarkson, M R, Finkbeiner, B, Koleini, M, Micinski, K K, Rabe, M N, Sánchez, C. Temporal logics for hyperproperties[C]. POST 2014. LNCS 8414, 2014: 265-284.

[264] B Finkbeiner, M N Rabe, C Sánchez. Algorithms for Model Checking HyperLTL and HyperCTL * [C]. CAV 2015, Part I, LNCS 9206, 2015: 30-48.

[265] J Zhao, X Li. Scope Logic with Local Reasoning and Pre/Post-State Properties[OL]. http://arxiv.org/abs/1012.2553. 2010.

[266] Taolue Chen, Fu Song, Zhilin Wu, Tractability of Separation Logic with Inductive Definitions: Beyond Lists[C]. CONCUR 2017, LIPIcs 85, 2017(37): 1-17.

[267] X Gu, T Chen, Z Wu. A complete decision procedure for linearly compositional separation logic with data constraints[C]. IJCAR 2016, LNCS 9706, 2016: 532-549.

[268] Z Xu, T Chen, Z Wu. Satisfiability of Compositional Separation Logic with Tree Predicates and Data Constraints[C]. CADE 2017, LNCS 10395, 2017: 509-527.

[269] T Chen, F Song, Z Wu. Global Model Checking on Pushdown Multi-Agent Systems[C]. AAAI 2016, 2016: 2459-2465.

[270] T Chen, F Song, Z Wu. Verifying Pushdown Multi-Agent Systems against Strategy Logics[C]. IJCAI 2016, 2016: 180-186.

[271] W Liu, L Song, J Wang, L Zhang. A Simple Probabilistic Extension of Modal Mu-Calculus[C]. IJCAI 2015, 2015: 882-888.

[272] Y Feng, N Yu, M Ying. Model checking quantum Markov chains[J]. Journal of Computer and System Sciences, 2013, 79(7): 1181-1198.

[273] Y Feng, E M Hahn, A Turrini, S Ying. Model Checking ω-regular Properties for Quantum Markov Chains[C]. CONCUR 2017, LIPIcs 85, 2017: 35: 1-35: 16.

[274] J Reynolds. Intuitionistic Reasoning about Shared Mutable Data Structure[C]. Millennial Perspectives in Computer Science, Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare, 1999.

[275] S Ishtiaq, P O' Hearn. BI as an Assertion Language for Mutable Data Structures[C]. POPL 2001, 2001: 14-26.

[276] Fei He, Xiaowei Gao, Miaofei Wang, Bow-Yaw Wang, Lijun Zhang. Learning Weighted Assumptions for Compositional Verification of Markov Decision Processes (journal version of the POPL' 15 paper) [J]. ACM Transactions on Software Engineering and Methodology (TOSEM), Article 21, 2016, 25(3): 39.

[277] Fei He, Liangze Yin, Bow-Yaw Wang, Lianyi Zhang, Guanyu Mu, Wenrui Meng. VCS: A Verifier for

Component-Based Systems[C]. ATVA 2013, 2013: 478-481.

[278] S Brookes. A Semantics for Concurrent Separation Logic[C]. CONCUR 2004, LNCS 3170, 2004: 16-34.

[279] P O' Hearn. Resources, Concurrency and Local Reasoning[C]. CONCUR 2004, LNCS 3170, 2004: 49-67.

[280] S Brookes and P O' Hearn. Concurrent separation logic[EB/OL]. SIGLOG News 3(3): 47-65, 2016.

[281] V Vafeiadis, C Narayan. Relaxed separation logic: a program logic for C11 concurrency[C]. OOPSLA 2013, 2013: 867-884.

[282] A Turon, V Vafeiadis, D Dreyer. GPS: navigating weak memory with ghosts, protocols, and separation [C]. OOPSLA 2014, 2014: 691-707.

[283] O Lahav, V Vafeiadis. Owicki-Gries Reasoning for Weak Memory Models[C]. ICALP (2) 2015, LNCS 9135, 2015: 311-323.

[284] M Doko, V Vafeiadis. A Program Logic for C11 Memory Fences[C]. VMCAI 2016, LNCS 9583, 2016: 413-430.

[285] M Doko, V Vafeiadis. Tackling Real-Life Relaxed Concurrency with FSL++[C]. ESOP 2017, LNCS 10201, 2017: 448-475.

[286] J Kaiser, H Dang, D Dreyer, O Lahav, V Vafeiadis. Strong Logic for Weak Memory: Reasoning About Release-Acquire Consistency in Iris[C]. ECOOP 2017, LIPIcs 74, 2017: 17: 1-17: 29.

[287] K Svendsen, J Pichon-Pharabod, M Doko, O Lahav, V Vafeiadis. A Separation Logic for a Promising Semantics[C]. ESOP 2018, LNCS 10801, 2018: 357-384.

[288] N Benton. Simple relational correctness proofs for static analyses and program transformations[C]. POPL 2004, 2004: 14-25.

[289] H Yang. Relational separation logic[J]. Theoretical Computer Science, 2007, 375(1-3): 308-332.

[290] L Beringer, M Hofmann. Secure information flow and program logics[C]. CSF 2007, 2007: 233-248.

[291] G Barthe, J Manuel Crespo, C Kunz. Product programs and relational program logics[J]. The Journal of Logic and Algebraic Programming, 2016, 85(5): 847-859.

[292] A Aguirre, G Barthe, M Gaboardi, D Garg, P-Y Strub. A relational logic for higher-order programs [C]. ICFP 2017, 2017: 21: 1-21: 29.

[293] A Turon, D Dreyer, L Birkedal. Unifying refinement and hoare-style reasoning in a logic for higher-order concurrency[C]. ICFP 2013, 2013: 377-390.

[294] H Liang, X Feng. Modular verification of linearizability with non-fixed linearization points[C]. PLDI 2013, 2013: 459-470.

[295] M Sousa, I Dillig. Cartesian hoare logic for verifying k-safety properties[C]. PLDI 2016, 2016: 57-69.

[296] G Klein, J Andronick, K Elphinstone, T Murray, T Sewell, R Kolanski, G Heiser. Comprehensive formal verification of an OS microkernel[J]. ACM Transaction on Computer Systems, 2014, 32(1): 2: 1-2: 70.

[297] R Gu, Z Shao, H Chen, X Wu, J Kim, V Sjöberg, D Costanzo. CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels[C]. OSDI 2016, 2016: 653-669.

[298] X Leroy. Formal verification of a realistic compiler[J]. Communication of ACM, 2009, 52(7): 107-115, 2009.

[299] X Leroy. A Formally Verified Compiler Back-end[J]. Journal of Automated Reasoning, 2009, 43(4):

363-446.

[300] J Ševčík, V Vafeiadis, F Zappa Nardelli, S Jagannathan, P Sewell. CompCertTSO: A Verified Compiler for Relaxed-Memory Concurrency[J]. Journal of ACM, 2013, 60(3): 22: 1-22: 50.

[301] G Stewart, L Beringer, S Cuellar, A Appel. Compositional CompCert [C]. POPL 2015, 2015: 275-287.

[302] J Kang, Y Kim, C-K Hur, D Dreyer, V Vafeiadis. Lightweight Verification of Separate Compilation [C]. POPL 2016, 2016: 178-190.

[303] C Hawblitzel, J Howell, M Kapritsos, J Lorch, B Parno, M Roberts, S Setty, B Zill. IronFleet: proving practical distributed systems correct[C]. SOSP 2015, 2015: 1-17.

[304] M Lesani, C Bell, A Chlipala. Chapar: Certified causally consistent distributed key-value stores[C]. POPL 2016, 2016: 357-370.

[305] C Hawblitzel, J Howell, J Lorch, A Narayan, B Parno, D Zhang, B Zill. Ironclad Apps: End-to-End Security via Automated Full-System Verification[C]. OSDI 2014, 2014: 165-181.

[306] L Beringer, A Petcher, K Ye, A Appel. Verified Correctness and Security of OpenSSL HMAC[C]. USENIX Security 2015, 2015: 207-221.

[307] A Appel. Verification of a Cryptographic Primitive: SHA-256[J]. ACM Transaction on Programming Language and Systems 2015, 37(2): 7: 1-7: 31.

[308] K Ye, M Green, N Sanguansin, L Beringer, A Petcher, A Appel. Verified Correctness and Security of mbedTLS HMAC-DRBG[C]. CCS 2017, 2017: 2007-2020.

[309] H Chen, D Ziegler, T Chajed, A Chlipala, M Frans Kaashoek, N Zeldovich. Using Crash Hoare logic for certifying the FSCQ file system[C]. SOSP 2015, 2015: 18-37.

[310] H Chen, T Chajed, Alex Konradi, S Wang, A Ileri, A Chlipala, M Kaashoek, N Zeldovich. Verifying a high-performance crash-safe file system using a tree specification[C]. SOSP 2017, 2017: 270-286.

[311] 尚书, 甘元科, 石刚, 王生原, 董渊. 可信编译器 L2C 的核心翻译步骤及其设计与实现 [J]. 软件学报, 2017, 28(5): 1233-1246.

[312] G Shi, Y Gan, S Shang, S Wang, Y Dong, P-C Yew. A formally verified sequentializer for lustre-like concurrent synchronous data-flow programs[C]. ICSE (Companion Volume) 2017, 2017: 109-111.

[313] G Shi, Y Zhang, S Shang, et al. A formally verified transformation to unify multiple nested clocks for a Lustre-like language[J]. Science China Information Science, 2019, 62(1).

[314] Y Zhao, D Sanán, F Zhang, Y Liu. Reasoning About Information Flow Security of Separation Kernels with Channel-Based Communication[C]. TACAS 2016, LNCS 9636, 2016: 791-810.

[315] Xingyuan Zhang, Christian Urban, Chunhan Wu. Priority Inheritance Protocol Proved Correct[C]. ITP 2012, LNCS 7406, 2012: 217-232.

[316] 宋丽华, 王海涛, 季晓君, 张兴元. 文件比较算法 fcomp 在 Isabelle/HOL 中的验证 [J]. 软件学报, 2017, 28(2): 203-215.

[317] H Liang, X Feng, Z Shao. Compositional verification of termination-preserving refinement of concurrent programs[C]. CSL-LICS 2014, 2014: 65: 1-65: 10.

[318] H Liang, X Feng. A program logic for concurrent objects under fair scheduling[C]. POPL 2016, 2016: 385-399.

[319] H Liang, X Feng. Progress of concurrent objects with partial methods[C]. POPL 2018, 2018: 20: 1-20: 31.

[320] S Wang, N Zhan, D Guelev. An Assume/Guarantee Based Compositional Calculus for Hybrid CSP[C]. TAMC 2012, LNCS 7287, 2012: 72-83.

[321] D Guelev, S Wang, N Zhan. Compositional Hoare-Style Reasoning About Hybrid CSP in the Duration Calculus[C]. SETTA 2017, LNCS 10606, 2017: 110-127.

[322] J Alglave, P Cousot. Ogre and Pythia: an invariance proof method for weak consistency models[C]. POPL 2017, 2017: 3-18.

[323] X Allamigeon, S Gaubert, E Goubault, S Putot, N Stott. A fast method to compute disjunctive quadratic invariants of numerical programs[J]. EMSOFT 2017, ACM Transactions on Embedded Computing Systems, TECS, 2017, 16(5).

[324] A Becchi, E Zaffanella. A Direct Encoding for NNC Polyhedra[C]. CAV 2018, LNCS 10981, 2018: 230-248.

[325] A Becchi, E Zaffanella. An Efficient Abstract Domain for Not Necessarily Closed Polyhedra[C]. SAS 2018.

[326] S Bygde, B Lisper, N Holsti. Improved precision in polyhedral analysis with wrapping[J]. Science of Computer Programming, 2017, 133(1): 74-87.

[327] A Chakarov, S Sankaranarayanan. Expectation Invariants as Fixed Points of Probabilistic Programs[J]. SAS 2014, LNCS 8723, 2014: 85-100.

[328] K Chae, H Oh, K Heo, H Yang. Automatically Generating Features for Learning Program Analysis Heuristics for C-like Languages[C]. OOPSLA 2017, 2017: 101: 1-101: 25.

[329] L Chen, J Liu, A Miné, D Kapur, J Wang. An Abstract Domain to Infer Octagonal Constraints with Absolute Value[C]. SAS 2014, LNCS 8723, 2014: 156-175.

[330] L Chen, R Li, X Wu, J Wang. Static analysis of lists by combining shape and numerical abstractions [J]. Science Computer Programming 95, 2014: 320-342.

[331] P Cousot, R Cousot. Abstract Interpretation: a unified lattice mode for static analysis of programs by construction or approximation of fixpoints[C]. POPL 1977, 1977: 238-252.

[332] P Cousot, R Cousot. Systematic design of program analysis frameworks[C]. In POPL 1979, 1979: 269-282.

[333] P Cousot, R Cousot. Abstract interpretation: past, present and future[C]. CSL-LICS 2014, 2014: 2: 1-2: 10.

[334] P Cousot. Abstracting Induction by Extrapolation and Interpolation[C]. VMCAI 2015, LNCS 8931, 2015: 19-42.

[335] A Dan, Y Meshman, M Vechev, E Yahav. Effective Abstractions for Verification under Relaxed Memory Models[C]. VMCAI 2015, LNCS 8931, 2015: 449-466.

[336] Y Dong, D Jin, Y Gong. Symbolic Procedure Summary Using Region-based Symbolic Three-valued Logic[J]. Journal of Computers, 2014, 9(3): 774-780.

[337] Z Fu. Modularly Combining Numeric Abstract Domains with Points-to Analysis, and a Scalable Static Numeric Analyzer for Java[C]. VMCAI 2014, LNCS 8318, 2014: 282-301.

[338] A Fromherz, A Ouadjaout, A Miné. Static value analysis of Python programs by abstract interpretation [C]. NFM 2018, LNCS 10811, 2018: 185-202.

[339] T Gehr, M Mirman, D Drachsler-Cohen, P Tsankov, S Chaudhuri, M Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation[C]. IEEE S&P, 2018.

[340] H Illous, M Lemerre, X Rival. ARelational Shape Abstract Domain[C]. NFM 2017, LNCS 10227, 2017: 212-229.

[341] S Jeong, M Jeon, S Cha, H Oh. Data-Driven Context-Sensitivity for Points-to Analysis[C]. OOPSLA 2017, 2017: 100: 1-100: 28.

[342] J Jiang, L Chen, X Wu, J Wang. Block-wise abstract interpretation by combining abstract domains with SMT[C]. VMCAI 2017, LNCS 10145, 2017: 310-329.

[343] H Li, F Berenger, B-Y Chang, X Rival. Semantic-Directed Clumping of Disjunctive Abstract States [C]. POPL 2017, 2017: 32-45.

[344] J Liu, X Rival, L Chen. Automatic Verification of Embedded Manipulating Dynamic Structures Stored in System Code Contiguous Regions[C]. EMSOFT, 2018.

[345] Z Kincaid, J Breck, A Forouhi Boroujeni, T Reps. Compositional recurrence analysis revisited[C]. PLDI 2017, 2017: 48-262.

[346] Z Kincaid, J Cyphert, J Breck, T Reps. Non-linear Reasoning for Invariant Synthesis[C]. POPL, 2018: 54: 1-54: 33.

[347] Y Li, A Albarghouthi, Z Kincaid, A Gurfinkel, M Chechik. Symbolic optimization with SMT solvers [C]. POPL 2014, 2014: 607-618.

[348] J Liu, X Rival. Abstraction of Arrays Based on Non Contiguous Partitions[C]. VMCAI 2015, LNCS 8931, 2015: 282-299.

[349] A Miné, D Delmas. Towards an industrial use of sound static analysis for the verification of concurrent embedded avionics software[C]. EMSOFT 2015, 2015: 65-74.

[350] Antoine Miné, Jason Breck, Thomas W Reps, An Algorithm Inspired by Constraint Solvers to Infer Inductive Invariants in Numeric Programs[C]. ESOP 2016, LNCS 9632, 2016: 560-588.

[351] Caterina Urban, Antoine Miné, Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation[C]. VMCAI 2015, LNCS 8931, 2015: 190-208.

[352] Caterina Urban: FuncTion: An Abstract Domain Functor for Termination -(Competition Contribution) [C]. TACAS 2015, LNCS 9035, 2015: 464-466.

[353] Caterina Urban, Peter Müller. An Abstract Interpretation Framework for Input Data Usage[C]. ESOP 2018, LNCS 10801, 2018: 683-710.

[354] Hakjoo Oh, Kihong Heo, Wonchan Lee, Woosuk Lee, Daejun Park, Jeehoon Kang, Kwangkeun Yi: Global Sparse Analysis Framework[J]. ACM Transactionon Programming Languageand Systems, 2014, 36(3): 8: 1-8: 44.

[355] Hakjoo Oh, Wonchan Lee, Kihong Heo, Hongseok Yang, Kwangkeun Yi: Selective X-Sensitive Analysis Guided by Impact Pre-Analysis[J]. ACM Transactionon Programming Languageand Systems, 2016, 38(2): 6: 1-6: 45.

[356] Abdelraouf Ouadjaout, Antoine Miné, Noureddine Lasla, Nadjib Badache. Static analysis by abstract interpretation of functional properties of device drivers in TinyOS[J]. Journal of Systems and Software 120, 2016: 114-132.

[357] Yassamine Seladji. Finding Relevant Templates via the Principal Component Analysis[C]. VMCAI 2017, LNCS 10145, 2017: 483-499.

[358] Vijay D' Silva, Caterina Urban: Abstract Interpretation as Automated Deduction[C]. CADE 2015, LNCS 9195, 2015: 450-464.

[359] Gagandeep Singh, Markus Püschel, Martin T. Vechev: Making numerical program analysis fast[C]. PLDI 2015, 2015: 303-313.

[360] Gagandeep Singh, Markus Püschel, Martin Vechev. Fast Polyhedra Abstract Domain[C]. POPL 2017, 2017: 46-59.

[361] Gagandeep Singh, Markus Püschel, Martin Vechev. A Practical Construction for Decomposing Numerical Abstract Domains[C]. POPL 2018, 2018: 55: 1-55: 28.

[362] Gagandeep Singh, Markus Püschel, Martin Vechev. Fast Numerical Program Analysis with Reinforcement Learning[C]. CAV 2018, LNCS 10981, 2018: 211-229.

[363] Chungha Sung, Markus Kusano, Chao Wang. Modular verification of interrupt-driven software[C]. ASE 2017, 2017: 206-216.

[364] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, Suman Jana. Formal Security Analysis of Neural Networks using Symbolic Intervals[C]. USENIX Security, 2018.

[365] Xueguang Wu, Liqian Chen, Antoine Miné, Wei Dong, Ji Wang. Numerical Static Analysis of Interrupt-driven Programs via Sequentialization[C]. EMSOFT 2015, 2015: 55-64.

[366] Xueguang Wu, Liqian Chen, Antoine Miné, Wei Dong, Ji Wang. Static Analysis of Run-Time Errors in Interrupt-Driven Programs via Sequentialization[J]. ACM Transactions on Embedded Computing Systems (TECS)15(4), 2016: 70: 1-70: 26.

[367] 陈冬火, 刘全, 朱斐, 金海东. 基于凸多面体抽象域的自适应强化学习技术研究 [J]. 计算机学报, 2018, 41(1): 112-131.

[368] 姜加红, 尹帮虎, 陈立前. 基于区间线性模版约束的程序分析 [J]. 计算机学 2018, 41(3): 545-557.

[369] 潘建东, 陈立前, 黄达明, 孙浩, 曾庆凯. 含有析取语义循环的不变式生成改进方法 [J]. 软件学, 2016, 27(7): 1741-1756.

[370] Liyun Dai, Bican Xia, Naijun Zhan. Generating Non-linear Interpolants by Semidefinite Programming [C]. CAV 2013, LNCS 8044, 2013: 364-380.

[371] Ting Gan, Liyun Dai, Bican Xia, Naijun Zhan, Deepak Kapur, Mingshuai Chen. Interpolant Synthesis for Quadratic Polynomial Inequalities and Combination with EUF[C]. IJCAR 2016, LNCS 9706, 2016: 195-212.

[372] W Liu, RWang, X Fu, J Wang, W Dong, X Mao. Counterexample Preserving Reduction for Symbolic Model Checking[C]. ICTAC 2013, LNCS 8049, 2013: 249-266.

[373] Fu Song, Tayssir Touili. POMMADE: PushdOwn Model-checking for Malware Detection[C]. FSE 2013, 2013: 607-610.

[374] Fu Song, Tayssir Touili. Efficient CTL Model-Checking for Pushdown Systems [C]. Theoretical Computer Science, 2014, 549(11): 127-145.

[375] Cong Tian, Zhenhua Duan. Detecting spurious counterexamples efficiently in abstract model checking [C]. ICSE 2013, 2013: 202-211.

[376] Cong Tian, Zhenhua Duan, Zhao Duan. Making CEGAR More Efficient in Software Model Checking [J]. IEEE Transaction on Software Engineering, 2014, 40(12): 1206-1223.

[377] Dingbao Xie, Lei Bu, Jianhua Zhao, Xuandong Li. SAT-LP-IIS joint-directed path-oriented bounded reachability analysis of linear hybrid automata[J]. Formal Methods in System Design, 2014, 45(1): 42-62.

[378] Aws Albarghouthi, Yi Li, Arie Gurfinkel, Marsha Chechik. Ufo: A Framework for Abstraction and Interpolation-Based Software Verification[J]. CAV 2012, LNCS 7358, 2012: 672-678.

[379] De Angelis, Emanuele, Fabio Fioravanti, Alberto Pettorossi, Maurizio Proietti. Verimap: A tool for verifying programs through transformations[C]. TACAS 2014, LNCS 8413, 2014: 568-574.

[380] Dirk Beyer, M Erkan Keremoglu. CPAchecker: A Tool for Configurable Software Verification[C]. CAV 2011, LNCS 6806, 2011: 184-190.

[381] A Biere, A Cimatti, E M Clarke, O Strichman, Y Zhu. Bounded Model Checking[C]. Advance in Computers, 2003(58): 118-149.

[382] Aaron Bradley. SAT-based model checking without unrolling[C]. VMCAI 2011, LNCS 6538, 2011: 70-87.

[383] Martin Brain, Vijay D' Silva, Alberto Griggio, Leopold Haller, Daniel Kroening. Interpolation-based verification of floating-point programs with abstract CDCL[C]. SAS 2013, LNCS 7935, 2013: 412-432.

[384] Alessandro Cimatti, Alberto Griggio. Software Model Checking via IC3[C]. CAV 2012, LNCS 7358, 2012: 277-293.

[385] E M Clarke, O Grumberg, S Jha, Y Lu, H Veith. Counterexample-guided abstraction refinement[C]. CAV 2000, LNCS 1855, 2000: 154-169.

[386] E M Clarke, D Kroening, F Lerda. A tool for checking ANSI-C programs[C]. TACAS 2004, LNCS 2988, 2004: 168-176.

[387] L Cordeiro, B Fischer, J Marques-Silva, SMT Based Bounded Model Checking for Embedded ANSI-C Software[J]. IEEE Transaction on Software Engineering, 2012, 38(4): 957-974.

[388] Stephan Falke, Florian Merz, Carsten Sinz. The Bounded Model Checker LLBMC (Tool Demonstration)[C]. ASE 2013, 2013: 706-709.

[389] Arie Gurfinkel, Temesghen Kahsai, Anvesh Komuravelli, Jorge A Navas. The seahorn verification framework[C]. CAV 2015, LNCS 9206, 2015: 343-361.

[390] F Ivancic, I Shlyakhter, A Gupta, M K Ganai. Model checking C programs using F-SOFT[C]. ICCD 2005, 2005: 297-308.

[391] Joxan Jaffar, Vijayaraghavan Murali, Jorge A. Navas, Andrew E. Santosa. TRACER: A symbolic execution tool for verification [C]. CAV 2012, LNCS 7358, 2012: 758-766.

[392] D Kroening, G Weissenbacher. Interpolation-based software verification with Wolverine[C]. CAV 2011, LNCS 6806, 2011: 573-578.

[393] Robert S. Boyer, Bernard Elspas, Karl N. Levitt. SELECT-a formal system for testing and debugging programs by symbolic execution[J]. ACM SigPlan Notices, 1975, 10(6): 234-245.

[394] Lori A Clarke. A program testing system[C]. Proceedings of the ACM Annual Conference, 1976: 488-491.

[395] James C King. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394.

[396] 张健. 精确的程序静态分析 [J]. 计算机学报, 2008, 31(9): 1549-1553.

[397] Vijay Ganesh, David L Dill. A decision procedure for bit-vectors and arrays[C]. CAV 2007, LNCS 4590, 2007: 519-531.

[398] Patrice Godefroid, Nils Klarlund, Koushik Sen. DART: Directed Automated Random Testing[C]. PLDI

2005, 2005: 213-223.

[399] Koushik Sen, Darko Marinov, Gul Agha. CUTE: A Concolic Unit Testing Engine for C[C]. FSE 2005, 2005: 263-272.

[400] Patrice Godefroid, Michael Y Levin, David A Molnar. Automated Whitebox Fuzz Testing[C]. NDSS 2008, 2008: 151-166.

[401] Cristian Cadar, Daniel Dunbar, Dawson Engler. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs[C]. OSDI 2008, 2008: 209-224.

[402] Vitaly Chipounov, Volodymyr Kuznetsov, George Candea. S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems[C]. ASPLOS 2011, 2011: 265-278.

[403] Patrice Godefroid, Adam Kiezun, Michael Y. Levin. Grammar-Based Whitebox Fuzzing[C]. PLDI 2008, 2008: 206-215.

[404] Junaid Haroon Siddiqui, Sarfraz Khurshid. Scaling symbolic execution using ranged analysis[C]. OOPSLA 2012, 2012: 523-536.

[405] Heming Cui, Gang Hu, Jingyue Wu, Junfeng Yang. Verifying systems rules using rule-directed symbolic execution[C]. ASPLOS 2013, 2013: 329-342.

[406] David Trabish, Andrea Mattavelli, Noam Rinetzky, Cristian Cadar. Chopped Symbolic Execution[C]. ICSE 2018, 2018: 350-360.

[407] Hengbiao Yu, Zhenbang Chen, Ji Wang, Zhendong Su, Wei Dong. Symbolic Verification of Regular Properties[C]. ICSE 2018, 2018: 871-881.

[408] Peter Boonstoppel, Cristian Cadar, Dawson R Engler. RWset: Attacking Path Explosion in Constraint-Based Test Generation[C]. TACAS 2008, LNCS 4963, 2008: 351-366.

[409] Joxan Jaffar, Vijayaraghavan Murali, Jorge A Navas. Boosting concolic testing via interpolation[C]. FSE 2013, 2013: 48-58.

[410] Patrice Godefroid. Compositional dynamic test generation[C]. POPL 2007, 2007: 47-54.

[411] Prateek Saxena, Pongsin Poosankam, Stephen McCamant, Dawn Song. Loop-extended symbolic execution on binary programs[C]. ISSTA 2009, 2009: 225-236.

[412] Patrice Godefroid, Daniel Luchaup. Automatic partial loop summarization in dynamic test generation [C]. ISSTA 2011, 2011: 23-33.

[413] Jan Strejček, Marek Trtík. Abstracting path conditions[C]. ISSTA, 2012: 155-165.

[414] Rui Qiu, Guowei Yang, Corina S Pasareanu, Sarfraz Khurshid. Compositional Symbolic Execution with Memoized Replay[C]. ICSE 2015, 2015: 632-642.

[415] Qiuping Yi, Zijiang Yang, Shengjian Guo, Chao Wang, Jian Liu, Chen Zhao. Eliminating Path Redundancy via Postconditioned Symbolic Execution[J]. IEEE Transactionon Software Engineering, 2018, 44(1): 25-43.

[416] Koushik Sen. Scalable automated methods for dynamic program analysis[D]. Champaign: University of Illinois at Urbana-Champaign, 2006.

[417] Chao Wang, Zijiang Yang, Vineet Kahlon, Aarti Gupta. Peephole Partial Order Reduction[C]. TACAS 2008, LNCS 4963, 2008: 382-396.

[418] Volodymyr Kuznetsov, Johannes Kinder, Stefan Bucur, George Candea. Efficient State Merging in Symbolic Execution[C]. PLDI 2012, 2012: 193-204.

[419] Thanassis Avgerinos, Alexandre Rebert, Sang Kil Cha, David Brumley. Enhancing symbolic execution

with veritesting[C]. ICSE 2014, 2014: 1083-1094.

[420] Dawei Qi, Hoang D T Nguyen, Abhik Roychoudhury. Path exploration based on symbolic output[J]. ACM Transactions on Software Engineering and Methodology, 2013, 22(4): 32: 1-32: 41.

[421] Haijun Wang, Ting Liu, Xiaohong Guan, Chao Shen, Qinghua Zheng, Zijiang Yang. Dependence Guided Symbolic Execution[J]. IEEE Transaction on Software Engineering, 2017, 43(3): 252-271.

[422] Willem Visser, Jaco Geldenhuys, Matthew B Dwyer. Green: reducing, reusing and recycling constraints in program analysis[C]. FSE 2012, 2012: 58: 1-58: 11.

[423] Andrea Aquino, Francesco A Bianchi, Meixian Chen, Giovanni Denaro, Mauro Pezzè. Reusing constraint proofs in program analysis[C]. ISSTA 2015, 2015: 305-315.

[424] Yufeng Zhang, Zhenbang Chen, Ji Wang. Speculative Symbolic Execution[C]. ISSRE 2012, 2012: 101-110.

[425] Xiangyang Jia, Carlo Ghezzi, Shi Ying. Enhancing reuse of constraint solutions to improve symbolic execution[C]. ISSTA 2015, 2015: 177-187.

[426] Earl T Barr, Thanh Vo, Vu Le, Zhendong Su. Automatic detection of floating-point exceptions[C]. POPL 2013, 2013: 549-560.

[427] Kiran Lakhotia, Nikolai Tillmann, Mark Harman, and Jonathan de Halleux. Flopsy - search-based floating point constraint solving for symbolic execution[C]. ICTSS 2010, LNCS 6435, 2010: 142-157.

[428] Anthony Romano. Practical floating-point tests with integer code[C]. VMCAI 2014, LNCS 8318, 2014: 337-356.

[429] Xin Li, Yongjuan Liang, Hong Qian, Yi-Qi Hu, Lei Bu, Yang Yu, Xin Chen, Xuandong Li. Symbolic execution of complex program driven by machine learning based constraint solving [C]. ASE 2016, 2016: 554-559.

[430] Zhoulai Fu, Zhendong Su. XSat: A Fast Floating-Point Satisfiability Solver [C]. CAV 2016, LNCS 9779, 2016: 187-209.

[431] Nikolaj Bjørner, Nikolai Tillmann, Andrei Voronkov. Path Feasibility Analysis for String-Manipulating Programs[C]. TACAS 2009, LNCS 5505, 2009: 307-321.

[432] Sarfraz Khurshid, Corina S Pasareanu, Willem Visser. Generalized Symbolic Execution for Model Checking and Testing[C]. TACAS 2003, LNCS 2619, 2003: 553-568.

[433] David A Ramos, Dawson R Engler. Under-Constrained Symbolic Execution: Correctness Checking for Real Code[C]. USENIX Security 2015, 2015: 49-64.

[434] Nicolás Rosner, Jaco Geldenhuys, Nazareno Aguirre, Willem Visser, Marcelo F Frias. BLISS: Improved Symbolic Execution by Bounded Lazy Initialization with SAT Support [J]. IEEE Transaction on Software Engineering, 2015, 41(7): 639-660.

[435] Jinseong Jeon, Xiaokang Qiu, Jonathan Fetter-Degges, Jeffrey S Foster, Armando Solar-Lezama. Synthesizing framework models for symbolic execution[C]. ICSE 2016, 2016: 156-167.

[436] Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert, David Brumley. Unleashing Mayhem on Binary Code[C]. S&P 2012, 2012: 380-394.

[437] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Krügel, Giovanni Vigna. SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis[C]. S&P 2016, 2016: 138-157.

[438] Radu Banabic, George Candea, Rachid Guerraoui. Finding trojan message vulnerabilities in distributed

systems[J]. ASPLOS 2014, 2014: 113-126.

[439] Stefan Bucur, Johannes Kinder, George Candea. Prototyping symbolic execution engines for interpreted languages[J]. ASPLOS 2014, 2014: 239-254.

[440] Michael Emmi, Rupak Majumdar, Koushik Sen. Dynamic Test Input Generation for Database Applications[C]. ISSTA 2007, 2007: 151-162.

[441] Raimondas Sasnauskas, Olaf Landsiedel, Muhammad Hamad Alizai, Carsten Weise, Stefan Kowalewski, Klaus Wehrle. KleeNet: Discovering Insidious Interaction Bugs in Wireless Sensor Networks Before Deployment[C]. IPSN 2010, 2010: 189-196.

[442] Xianjin Fu, Zhenbang Chen, Hengbiao Yu, Chun Huang, Wei Dong, Ji Wang. Poster: Symbolic Execution of MPI Programs[C]. ICSE 2015, 2015: 809-810.

[443] Shengjian Guo, Markus Kusano, Chao Wang, Zijiang Yang, Aarti Gupta. Assertion guided symbolic execution of multithreaded programs[C]. FSE 2015, 2015: 854-865.

[444] Hengbiao Yu. Combining symbolic execution and model checking to verify MPI programs[C]. ICSE 2018, 2018: 527-530.

[445] Drew Davidson, Benjamin Moench, Thomas Ristenpart, Somesh Jha. FIE on Firmware: Finding Vulnerabilities in Embedded Systems Using Symbolic Execution[C]. USENIX Security 2013, 2013: 463-478.

[446] Shengjian Guo, Meng Wu, Chao Wang. Symbolic execution of programmable logic controller code [C]. FSE 2017, 2017: 326-335.

[447] Ting Su, Zhoulai Fu, Geguang Pu, Jifeng He, Zhendong Su. Combining Symbolic Execution and Model Checking for Data Flow Testing[C]. ICSE 2015, 2015: 654-665.

[448] Maria Christakis, Peter Müller, Valentin Wüstholz. Guiding dynamic symbolic execution toward unverified program executions[C]. ICSE 2016, 2016: 144-155.

[449] Sheng Liu, Jian Zhang. Program analysis: from qualitative analysis to quantitative analysis (NIER track)[C]. ICSE 2011, 2011: 956-959.

[450] Jaco Geldenhuys, Matthew B Dwyer, Willem Visser. Probabilistic symbolic execution[C]. ISSTA 2012, 2012: 166-176.

[451] Antonio Filieri, Corina S Pasareanu, Willem Visser. Reliability analysis in symbolic pathfinder[C]. ICSE 2013, 2012: 622-631.

[452] Bihuan Chen, Yang Liu, Wei Le. Generating performance distributions via probabilistic symbolic execution[C]. ICSE 2016, 2016: 49-60.

[453] Antonio Filieri, Corina S Pasareanu, Guowei Yang. Quantification of Software Changes through Probabilistic Symbolic Execution[C]. ASE 2015, 2015: 703-708.

[454] Xinyu Wang, Jun Sun, Zhenbang Chen, Peixin Zhang, Jingyi Wang, Yun Lin. Towards optimal concolic testing[C]. ICSE 2018, 2018: 291-302.

[455] Nick Stephens, John Grosen, Christopher Salls, Andrew Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, Giovanni Vigna. Driller: Augmenting Fuzzing Through Selective Symbolic Execution[C]. NDSS, 2016.

[456] Liqian Chen, Jiahong Jiang, Banghu Yin, Wei Dong, Ji Wang. Robustness Analysis of Floating-Point Programs by Self-Composition[J]. Journal of Applied Mathematics, 2014: 789213: 1-789213: 12.

[457] Zhenbo Xu, Jian Zhang, Zhongxing Xu, Jiteng Wang. Canalyze: a static bug-finding tool for C programs

[C]. ISSTA 2014, 2014: 425-428.

[458] A Cox, B Evan Chang, X Rival. Desynchronized Multi-State Abstractions for Open Programs in Dynamic Languages[C]. ESOP 2015, LNCS 9032, 2015: 483-509.

[459] Tao Xie, Nikolai Tillmann, Jonathan de Halleux, Wolfram Schulte. Fitness-guided path exploration in dynamic symbolic execution[C]. DSN 2009, 2009: 359-368.

[460] You Li, Zhendong Su, Linzhang Wang, Xuandong Li. Steering Symbolic Execution to Less Traveled Paths[C]. OOPSLA 2013, 2013: 19-32.

[461] Hyunmin Seo, Sunghun Kim. How we get there: a context-guided search strategy in concolic testing [C]. FSE 2014, 2014: 413-424.

[462] Kin-Keung Ma, Khoo Yit Phang, Jeffrey S Foster, Michael Hicks. Directed Symbolic Execution[C]. SAS 2011, LNCS 6887, 2011: 95-111.

[463] Yufeng Zhang, Zhenbang Chen, Ji Wang, Wei Dong and Zhiming Liu. Regular property guided dynamic symbolic execution[C]. ICSE 2015, 2015: 643-653.

[464] Suzette Person, Guowei Yang, Neha Rungta, Sarfraz Khurshid. Directed incremental symbolic execution [C]. PLDI 2011, 2011: 504-515.

[465] Paul Dan Marinescu, Cristian Cadar. make test-zesti: A symbolic execution solution for improving regression testing[C]. ICSE 2012, 2012: 716-726.

[466] S Bucur, V Ureche, C Zamfir, G Candea, Parallel symbolic execution for automated real-world software testing[C]. EuroSys 2011, 2011: 183-198.

[467] R Qiu, S Khurshid, C S Pasareanu, J Wen, G Yang: Using Test Ranges to Improve Symbolic Execution [C]. NFM 2018, LNCS 10811, 2018: 416-434.

[468] C S Pasareanu, D Giannakopoulou, M Gheorghiu Bobaru, J M Cobleigh, H Barringer: Learning to divide and conquer: applying the L * algorithm to automate assume-guarantee reasoning[J]. Formal Methods in System Design 2008, 32(3): 175-205.

[469] Guodong Li, Indradeep Ghosh, Sreeranga P Rajan: KLOVER: A Symbolic Execution and Automatic Test Generation Tool for C++ Programs[C]. CAV 2011, LNCS 6806, 2011: 609-615.

[470] Robin David, Sébastien Bardin, Josselin Feist, Laurent Mounier, Marie-Laure Potet, Thanh Dinh Ta, Jean-Yves Marion. Specification of concretization and symbolization policies in symbolic execution[C]. ISSTA 2016, 2016: 36-46.

[471] Lian Li, Yi Lu, Jingling Xue, Dynamic symbolic execution for polymorphism[C]. CC 2017, 2017: 120-130.

[472] Dirk Beyer, Thomas Henzinger, Ranjit Jhala, Repack Majumdar. The Software Model Checker Blast [J]. International Journal on Software Tools for Technology Transfer, 2007, 9 (5-6): 505-525.

[473] Peter Schrammel, Daniel Kroening. 2LS for Program Analysis[C]. TACAS 2016, LNCS 9636, 2016: 905-907.

[474] Kroening, Daniel, Michael Tautschnig. CBMC-C bounded model checker[C]. TACAS 2014, LNCS 8413, 2014: 389-391.

[475] Beyer, Dirk, M Erkan Keremoglu. CPAchecker: A tool for configurable software verification[C]. CAV 2011, LNCS 6806, 2011: 184-190.

[476] Ball, Thomas, Byron Cook, Vladimir Levin, Sriram K Rajamani. SLAM and Static Driver Verifier: Technology transfer of formal methods inside Microsoft[C]. IFM 2004, LNCS 2999, 2004: 1-20.

[477] Heizmann, Matthias, Jochen Hoenicke, Andreas Podelski. Software model checking for people who love automata[C]. CAV 2013, LNCS 8044, 2013: 36-52.

[478] Beyer, Dirk, Thomas A Henzinger, Grégory Théoduloz. Configurable software verification: Concretizing the convergence of model checking and program analysis[C]. CAV 2007, LNCS 4590, 2007: 504-518.

[479] L C Cordeiro, J Morse, D Nicole, B Fischer. Context bounded model checking with ESBMC 1.17-(competition contribution)[C]. TACAS 2012, LNCS 7214, 2012: 534-537.

[480] PolySpace[OL]. http://www.mathworks.com/products/polyspace.html.

[481] Astrée[OL]. http://www.astree.ens.fr/.

[482] aiT WCET Analyzer[OL]. http://www.absint.com/ait/.

[483] Code Hawk[OL]. http://kestreltechnology.com/technology.html.

[484] Sparrow[OL]. http://ropas.snu.ac.kr/sparrow/.

[485] Julia[OL]. http://juliasoft.com/abstract-interpretation/.

[486] Frama-C Value Analysis[OL]. http://frama-c.com/value.html.

[487] Manuel Fähndrich, Francesco Logozzo. Static Contract Checking with Abstract Interpretation[C]. FoVeOOS 2010, LNCS 6528, 2010.

[488] Interproc[OL]. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/index.html.

[489] Crab-llvm[OL]. http://github.com/seahorn/crab-llvm.

[490] IKOS[OL]. http://ti.arc.nasa.gov/opensource/ikos/.

[491] MemCAD[OL]. http://www.di.ens.fr/~rival/memcad.html.

[492] Fluctuat[OL]. http://www.lix.polytechnique.fr/~putot/fluctuat.html.

[493] Jandom[OL]. http://github.com/jandom-devel/Jandom.

[494] JsCFA[OL]. http://github.com/fiigii/JsCFA-prototype.

[495] MuJS[OL]. http://github.com/SPY-Lab/mu-js.

[496] O Inverso, E Tomasco, B Fischer, S La Torre, G Parlato. Bounded model checking of multi-threaded C programs via lazy sequentialization[C]. CAV 2014, LNCS 8559, 2014: 585-602.

[497] Carter M, He S, Whitaker J, et al. SMACK software verification toolchain[C]. ICSE Companion 2017, 2017: 589-592.

[498] Wendler P CPAchecker, with Sequential Combination of Explicit-State Analysis and Predicate Analysis [C]. TACAS 2013, LNCS 7795, 2013: 613-615.

[499] CertiKOS 操作系统[OL]. http://flint.cs.yale.edu/certikos/.

[500] Fisher K, Launchbury J, Richards R. The HACMS program: usingformal methods to eliminate exploitable bugs[J]. Philosophical Transactions of The Royal Society 2017, A375(2104).

[501] Project Everest -Verified Secure Implementations of the HTTPS Ecosystem[OL]. http://www.microsoft.com/en-us/research/project/project-everest-verified-secure-implementations-https-ecosystem/, http://project-everest.github.io/.

[502] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, Michael Deardeuff. How Amazon web services uses formal methods[J]. Communication of ACM, 2015, 58(4): 66-73.

[503] M Althoff, D Grebenyuk, N Kochdumper. Implementation of Taylor models in CORA 2018[C]. In Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems, 2018.

[504] C Fan, B Qi, S Mitra, M Viswanathan, P S Duggirala. Automatic reachability analysis for nonlinear hybrid models with c2e2[C]. CAV 2016, LNCS 9779, 2016: 531-538.

[505] X Chen, EÁbrahám, S Sankaranarayanan. Flow *: An analyzer for non-linear hybrid systems[C]. CAV 2013, LNCS 8044, 2013: 258-263.

[506] G Frehse. Reachability of hybrid systems in space-time[C]. In Alain Girault and Nan Guan, editors, Proc. Int. Conf. Embedded Software, EMSOFT 2015, 2015: 41-50.

[507] S Bak, P S Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for lin-ear systems [C]. HSCC 2017, 2017: 173-178.

[508] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Fr éd éric Viry, Andreas Podelski, Christian Schilling. Reach set approximation through decomposition with low-dimensional sets and high-dimensional matrices[C]. HSCC 2018, 2018: 41-50.

[509] Rajarshi Ray, Amit Gurung, Binayak Das, Ezio Bartocci, Sergiy Bogomolov, Radu Grosu. XSpeed: Accelerating reachability analysis on multi-core processors[C]. HVC 2015, LNCS 9434, 2015: 3-18.

[510] Stefan Schupp, Erika Abraham, Ibtissem Ben Makhlouf, Stefan Kowalewski[C]. HyPro: A C++ library for state set representations for hybrid systems reachability analysis. NFM 2017, LNCS 10227, 2017: 288-293.

[511] Alessandro Cimatti, Alberto Griggio, Sergio Mover, Stefano Tonetta. HYCOMP: an SMT-based Model Checker for Hybrid Systems[C]. TACAS 2015, LNCS 9035, 2015: 52-67.

[512] Sergiy Bogomolov, Goran Frehse, Mirco Giacobbe, Thomas A Henzinger. Counterexample-guided refinement of template polyhedral[C]. TACAS 2017, LNCS 10205, 2017: 589-606.

[513] Martin Franzle, Christian Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems[J]. Formal Methods in System Design, 2007, 30(3): 179-198.

[514] Sicun Gao, Soonho Kong, Edmund Clarke. dReach: Reachability Analysis for Nonlinear Hybrid Systems [C]. TACAS 2015, LNCS 9035, 2015: 200-205.

[515] Guo Y, Cai Y, Yang Z. AtexRace: across thread and execution sampling for in-house race detection [C]. FSE 2017, 2017: 315-325.

[516] Weiqiang Kong, Leyuan Liu, Takahiro Ando, Hirokazu Yatsu, Kenji Hisazumi, Akira Fukuda, Facilitating Multicore Bounded Model Checking with Stateless Explicit-State Exploration[J]. The Computer Journal, 2015, 58(11): 2824-2840.

[517] 朱允敏, 张丽伟, 王生原, 等. 面向多核处理器的低级并行程序验证[J]. 电子学报, 2009, 37(s1): 1-6.

[518] L Yin, W Dong, W Liu, Y Li, J Wang. YOGAR-CBMC: CBMC with scheduling constraint based abstraction refinement-(competition contribution)[C]. TACAS 2018, LNCS 10806, 2018: 422-426.

[519] Shuo Zhang, Fei He, Ming Gu. VeRV: A Temporal and Data-Concerned Verification Framework for the Vehicle Bus Systems[C]. INFOCOM 2015, 2015: 1167-1175.

[520] M Chen, X Han, T Tang, S Wang, M Yang, N Zhan, H Zhao, L Zou. MARS: A Toolchain for Modelling, Analysis and Verification of Hybrid Systems[C]. Provably Correct Systems. Springer International Publishing, 2017.

[521] S Wang, N Zhan, L Zou. An Improved HHL Prover: An Interactive Theorem Prover for Hybrid Systems [C]. ICFEM 2015, LNCS 9407, 2015: 382-399.

[522] Lei Bu, You Li, Linzhang Wang, Xuandong Li. BACH: Bounded reachability checker for linear hybrid automata[C]. FMCAD 2008, 2008: 1-4.

[523] Tao Li, Feng Tan, Qixin Wang, Lei Bu, Jiannong Cao, Xue Liu. From Offline toward Real Time: A Hybrid Systems Model Checking and CPS Codesign Approach for Medical Device Plug- and- Play Collaborations[C]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(3): 642-652.

[524] Lei Bu, Wen Xiong, Chieh-Jan Mike Liang, Shi Han, Dongmei Zhang, Shan Lin, Xuandong Li. Systematically Ensuring The Confidence of Real Time Home Automation IoT Systems[C]. ACM Transactions on Cyber-Physical Systems, 2018, 2(3): 22: 1-22: 23.

[525] Antoine Miné, David Delmas. Towards an industrial use of sound static analysis for the verification of concurrent embedded avionics software[C]. EMSOFT 2015, 2015: 65-74.

[526] Antoine Miné, Laurent Mauborgne, Xavier Rival, Jérôme Feret, Patrick Cousot, Daniel Kästner, Stephan Wilhelm, Christian Ferdinand. Taking Static Analysis to the Next Level: Proving the Absence of Run-Time Errors and Data Races with Astrée[C]. ERTS2 2016, 2016: 570-579.

[527] A JavadiAbhari, S Patil, D Kudrow, J Heckey, A Lvov, F T Chong, M Martonosi, ScaffCC: Scalable Compilation and Analysis of Quantum Programs[J]. arXiv. org, vol. quant- ph. pp. arXiv: 1507. 01902-17, 08-Jul-2015.

[528] M Ying, Y Feng. Quantum loop programs[J]. Acta Informatica, 2010: 47(4): 221-250.

[529] M Ying, N Yu, Y Feng, R Duan. Verification of quantum programs[J]. Science of Computer Programming, 2013: 78(9):1679-1700.

[530] The Q# Programming Language[OL]. http://docs. microsoft. com/en- us/quantum/quantum- qr- intro? view = qsharp-preview.

[531] S Perdrix. Quantum Entanglement Analysis Based on Abstract Interpretation[C]. SAS 2008, LNCS 5079, 2008: 270-282.

[532] P Jorrand, S Perdrix. Abstract Interpretation Techniques for Quantum Computation[C]. in Semantic Techniques in Quantum Computation, no. 6, S. Gay and I. Mackie, Eds. Cambridge: Cambridge University Press, 2009: 206-234.

[533] K Honda. Analysis of Quantum Entanglement in Quantum Programs using Stabilizer Formalism[J]. QPL 2015, EPTCS 195, 2015: 262-272.

[534] O Brunet, P Jorrand. Dynamic quantum logic for quantum programs[J]. International Journal of Quantum Information, 2011, 2(1): 45-54.

[535] A Baltag, S Smets. LQP: the dynamic logic of quantum information[J]. Mathemathical Structure in Computer Science, 2006, 16(3): 491-525.

[536] Y Feng, R Duan, Z Ji, M Ying. Proof rules for the correctness of quantum programs[J]. Theoretical Computer Science, 2007, 386(1): 151-166.

[537] R Chadha, P Mateus, A Sernadas. Reasoning About Imperative Quantum Programs[J]. Electronic Notes in Theoretical Computer Science, 2006(158): 19-39.

[538] Y Kakutani. A Logic for Formal Verification of Quantum Programs[C]. ASIAN 2009, LNCS 5913, 2009: 79-93.

[539] M Ying. Floyd-Hoare logic for quantum programs[J]. ACM Transactions on Programming Languages and Systems, 2011, 33(6): 1-49.

[540] T Liu, Y Li, S Wang, M Ying, N Zhan. A Theorem Prover for Quantum Hoare Logic and Its Applications[J]. arXiv.org, vol. cs. LO. p. arXiv: 1601. 03835, 15-Jan-2016.

[541] M Ying, S Ying, X Wu. Invariants of quantum programs-characterisations and generation[C]. POPL 2017, 2017: 818-832.

[542] S Ying, Y Feng, N Yu, M Ying. Reachability Probabilities of Quantum Markov Chains[C]. CONCUR 2013, LNCS 8052, 2013: 332-348.

[543] J Guan, Y Feng, M Ying. Decomposition of quantum Markov chains and its applications[J]. Journal of Computer and System Sciences, 2018(95): 55-68.

[544] Y Feng, N Yu, M Ying. Reachability Analysis of Recursive Quantum Markov Chains[C]. MFCS 2013, LNCS 8087, 2013: 385-396.

[545] Y Feng, E M Hahn, A Turrini, L Zhang. QPMC: A Model Checker for Quantum Programs and Protocols [C]. FM 2015, LNCS 9109, 2015: 265-272.

[546] L Anticoli, C Piazza, L Taglialegne, P Zuliani. Towards Quantum Programs Verification - From Quipper Circuits to QPMC[C]. RC 2016, LNCS 9720, 2016: 213-219.

[547] Hui Kong, Fei He, Xiaoyu Song, William N N. Hung, Ming Gu. Exponential-Condition-Based Barrier Certificate Generation for Safety Verification of Hybrid Systems[C]. CAV 2013, LNCS 8044, 2013: 242-257.

[548] Fei He, Bow-Yaw Wang, Liangze Yin, Lei Zhu. Symbolic Assume-Guarantee Reasoning through BDD Learning[C]. ICSE 2014, 2014: 1071-1082.

[549] Fei He, Shu Mao, Bow-Yaw Wang. Learning-based Assume-Guarantee Regression Verification[C]. CAV 2016, LNCS 9780, 2016: 310-328.

[550] Fei He, Xiaowei Gao, Bow-Yaw Wang, Lijun Zhang. Leveraging Weighted Automata in Composition Reasoning about Concurrent Probabilistic Systems[C]. POPL 2015, 2015: 503-514.

作者简介

卜 磊 南京大学，副教授，主要研究实时混成 CPS 系统、软件代码等复杂系统形式化验证与分析测试技术。CCF 形式化方法专委会委员、系统软件专委会委员。

陈立前 国防科技大学，副教授，主要研究程序分析与验证、抽象解释，CCF 形式化方法专委会委员。

陈　哲　南京航空航天大学，副教授，主要研究形式化方法、软件验证、航空电子系统的可靠性与安全性验证、软件工程，CCF 形式化方法专委会委员。

陈振邦　国防科技大学，副教授、硕导，主要研究形式化方法、程序分析及其应用，CCF 形式化方法专委会委员。

冯新宇　南京大学，教授，主要研究程序设计语言、形式化程序验证、软件安全，CCF 形式化方法专委会委员、系统软件专委会委员。

冯　元　悉尼科技大学量子软件与信息中心，教授，主要研究量子程序分析与验证。

贺　飞　清华大学，副教授、博导，主要研究形式化方法、程序分析与验证等，CCF 形式化方法专委会委员。

李国强　上海交通大学，副教授、博导，主要研究形式化验证、程序语言理论、知识表示与推理，CCF 形式化方法专委会委员。

刘万伟 国防科技大学，副教授，主要研究时序逻辑、模型检测、自动机理论，CCF 形式化方法专委会委员。

马菲菲 中国科学院软件研究所，副研究员，主要研究自动推理、约束求解。

宋 富 上海科技大学，助理教授、研究员、博导，主要研究形式化方法、程序分析与验证、计算机安全和软件工程，CCF 形式化方法专委会委员。

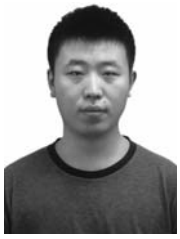
田 聪 西安电子科技大学，教授，主要研究可信软件理论与方法，大数据和机器学习软件开发方法，CCF 形式化方法专委会委员，CCF 青工委委员，女工委委员。

王淑灵 中国科学院软件研究所，副研究员，主要研究形式化方法、交互式定理证明以及混成系统建模与验证，CCF 形式化方法专委会委员。

吴志林 中国科学院软件研究所，副研究员，主要研究程序分析与验证、计算逻辑、自动机理论，CCF 形式化方法专委会委员。

薛　白　中国科学院软件研究所，副研究员，主要研究混成系统分析与验证。

杨鹏飞　中国科学院软件研究所，博士生，主要研究概率模型检测。

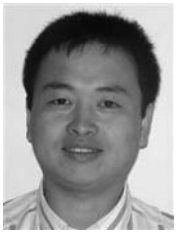


尹良泽　国防科技大学，讲师，主要研究形式化方法、程序分析与验证。

詹博华　慕尼黑工业大学，博士后，主要研究交互式定理证明、证明自动化、形式化数学和程序验证。

张　民　华东师范大学，副教授，主要研究形式化方法、程序分析与验证、计算机系统建模和软件工程，CCF 形式化方法专委会委员。

张立军　中国科学院软件研究所，研究员，主要研究概率模型检测，CCF 形式化方法专委会委员。



张兴元 中国人民解放军陆军工程大学，教授，主要研究计算机辅助定理证明，CCF 形式化方法专委会委员。

赵永望 北京航空航天大学，副教授，主要研究形式化方法、操作系统内核、程序验证等，CCF 形式化方法专委会委员、系统软件专委会委员。