软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(7):2367−2410 [doi: 10.13328/j.cnki.jos.006587]

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

可信系统性质的分类和形式化研究综述*

王淑灵[1,2], 詹博华[1,2], 盛欢欢[1,2], 吴 昊[1,2], 易士程[1,2], 王令泰[1,2], 金翔宇[1,2], 薛 白[1,2], 李静辉[3], 向霜晴[3], 向 展[3], 毛碧飞[3]

[1](计算机科学国家重点实验室(中国科学院 软件研究所), 北京 100190)

[2](中国科学院大学, 北京 100049)

[3](华为技术有限公司 可信理论研究室, 广东 深圳 518129)

通信作者: 詹博华, E-mail: bzhan@ios.ac.cn

摘 要: 计算机系统被应用于各种重要领域, 这些系统的失效可能会带来重大灾难. 不同应用领域的系统对于可信性具有不同的要求, 如何建立高质量的可信计算机系统, 是这些领域共同面临的巨大挑战. 近年来, 具有严格数学基础的形式化方法已经被公认为开发高可靠软硬件系统的有效方法. 目标是对形式化方法在不同系统的应用进行不同维度的分类, 以更好地支撑可信软硬件系统的设计. 首先从系统的特征出发, 考虑6种系统特征: 顺序系统、反应式系统、并发与通信系统、实时系统、概率随机系统以及混成系统. 同时, 这些系统又运行在众多应用场景, 分别具有各自的需求. 考虑4种应用场景: 硬件系统、通信协议、信息流以及人工智能系统. 对于以上的每个类别, 介绍和总结其形式建模、性质描述以及验证方法与工具. 这将允许形式化方法的使用者对不同的系统和应用场景, 能够更准确地选择恰当的建模、验证技术与工具, 帮助设计人员开发更加可靠的系统.

关键词: 可信系统; 形式化方法; 性质分类; 验证方法和工具

中图法分类号: TP311

中文引用格式: 王淑灵, 詹博华, 盛欢欢, 吴昊, 易士程, 王令泰, 金翔宇, 薛白, 李静辉, 向霜晴, 向展, 毛碧飞. 可信系统性质的分类和形式化研究综述. 软件学报, 2022, 33(7): 2367–2410. http://www.jos.org.cn/1000-9825/6587.htm

英文引用格式: Wang SL, Zhan BH, Sheng HH, Wu H, Yi SC, Wang LT, Jin XY, Xue B, Li JH, Xiang SQ, Xiang Z, Mao BF. Survey on Requirements Classification and Formalization of Trustworthy Systems. Ruan Jian Xue Bao/Journal of Software, 2022, 33(7): 2367–2410 (in Chinese). http://www.jos.org.cn/1000-9825/6587.htm

Survey on Requirements Classification and Formalization of Trustworthy Systems

WANG Shu-Ling[1,2], ZHAN Bo-Hua[1,2], SHENG Huan-Huan[1,2], WU Hao[1,2], YI Shi-Cheng[1,2], WANG Ling-Tai[1,2], JIN Xiang-Yu[1,2], XUE Bai[1,2], LI Jing-Hui[3], XIANG Shuang-Qing[3], XIANG Zhan[3], MAO Bi-Fei[3]

[1](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](Trustworthiness Theory Research Center, Huawei Technologies Co. Ltd., Shenzhen 518129, China)

Abstract: Computer systems have been applied in many different areas, and the failure of these systems may bring catastrophic results. Systems in different areas have different requirements, and how to build trustworthy computer systems with high quality is the challenge faced by all these areas. Recently, formal methods with rigorous mathematical foundation have been widely recognized as effective methods for developing trustworthy software and hardware systems. Based on formal methods, this paper presents a classification of requirements of systems and their formalization, to support the design of trustworthy systems. First of all, six types of system characteristics are considered, namely, sequential systems, reactive systems, parallel and communicating systems, real-time systems, probabilistic and stochastic systems, and continuous and hybrid systems. All these systems may run in different application scenarios, with

* 基金项目: 国家自然科学基金(61972385, 62032024, 61836005)
本文由“智能系统的分析和验证”专题特约编辑明仲教授、张立军教授和秦胜潮教授推荐.
收稿时间: 2021-09-05; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

their respective requirements. Four classes of scenarios are considered, i.e., hardware systems, security protocols, information flow, and AI systems. For each class of systems and application scenarios above, the related formal methods are introduced and summarized, including formal modeling, property specification, verification methods and tools. This will allow users of formal methods to choose, based on different system characteristics and application scenarios, the appropriate formal models, verification methods and tools, which ultimately helps the design of more trustworthy systems.

Key words: trustworthy system; formal method; requirement classification; verification method and tool

随着计算机科学和技术的飞速发展, 大量计算机系统被应用于人们生活的各个方面, 计算机系统的可信保障[1]也越来越受到关注. 系统可信性的研究传统上关注安全攸关领域, 例如轨道交通、航空航天、核电、医疗、军工等. 在这些领域, 系统的任何故障都可能给人的生命和财产带来重大损失. 此外, 计算机系统越来越多地应用于人们之间的通信, 并需要存储和处理大量用户信息. 如何保障通信内容和用户信息的完整性和隐私性, 成为这些系统设计和实现时的重点关注问题. 近期, 人工智能系统在数据挖掘与推荐、人脸识别、语音识别、文本处理等应用场景取得了显著的成功和广泛的应用, 但同时也带来了新的问题, 例如智能系统的安全性、公平性、训练数据的隐私性等. 总而言之, 建立高质量的可信计算机系统, 是计算机科学各个领域共同面临的巨大挑战. 不同领域和应用场景对系统可信性提出了不同的需求, 梳理这些可信性需求并对其进行分类, 有助于对系统可信性形成完整的理解.

近年来, 形式化建模与验证广泛应用于软硬件系统的正确性保障, 并被认为是开发高可信软硬件系统的有效方法[2,3]. 形式化建模使用数学语言描述系统的行为及其需要满足的性质, 而相应的验证技术则通过遍历系统状态或逻辑推理来证明系统是否满足这些性质. 与计算机系统的不同类型和应用场景相对应, 形式化方法也包含各种各样的系统模型、性质描述语言以及验证技术. 例如, 对于普通的顺序程序, 我们或许只关注程序的输入输出关系; 但对于需要与环境交互的反应式系统, 我们关注系统的整个运行过程; 对于具有实时性要求的系统, 我们关注系统能否在一定时间内做出响应, 等等. 基于深度学习的智能系统同样带来了新的关注点和研究方向.

针对不同类型的系统和性质, 采用什么样的数学语言去描述, 使用什么样的方法和工具去验证, 存在共通之处, 但同时也存在较大差异. 对各类系统和性质所采用的形式化方法进行整理, 一方面能够更准确地为不同的系统和应用场景选择恰当的验证技术与工具, 另一方面也有助于更完整地描述不同系统的可信性需求. 这对于可信系统的设计与开发具有重要的理论和实际指导意义. 本文的目标是对可信系统的需求和形式化方法在不同系统中的应用进行不同维度的分类, 以更好地支撑基于形式化方法的可信软硬件系统的分析与验证.

我们首先从不同的系统特征出发, 考虑 6 种系统特征: 顺序系统、反应式系统、并发与通信系统、实时系统、概率随机系统以及混成系统. 同时, 我们考虑 4 种应用场景: 硬件系统、通信协议、信息流以及人工智能系统. 这些系统特征和应用场景并非互不相容, 而是有很多交叉之处. 针对每种系统特征和应用场景, 我们对相关的形式化方法研究进行概述, 主要关注以下 3 个问题.

(1) 系统描述语言, 即可以使用哪些形式化模型对系统的行为进行建模. 常见的建模语言包括迁移系统、自动机、进程代数、重写系统等. 根据不同的系统特征, 需要对这些基本建模语言进行相应的拓展. 例如, 实时系统需要建模语言包含时钟的描述, 混成系统需要建模语言使用微分方程描述连续行为, 通信协议的建模语言需要刻画攻击者模型, 等等.

(2) 性质规约语言, 即可以使用哪些形式语言描述系统需要满足的性质. 常见的性质规约语言包括命题逻辑和谓词逻辑、时序逻辑、动态逻辑、霍尔逻辑、区间演算等. 性质规约语言的选择需要考虑性质是否只关注输入输出关系还是整个运行过程, 以及是否需要考虑实时和概率因素等.

(3) 验证方法与工具, 即在给定系统描述和性质规约后, 采用哪些方法验证系统满足规约, 以及现有哪些验证工具. 验证方法可以分为两大类: 模型检验[4−6]和定理证明[7,8]. 其中, 模型检验通过遍历系统的状态空间验证系统, 自动化程度较高, 但面临状态空间爆炸的问题. 如何控制状态空间的大小,

是模型检验研究长期关注的问题, 解决方案包括符号模型检验[9]、抽象精化[10]、限界模型检验[11]等. 定理证明则是通过逻辑推理的方式验证系统, 理论框架以霍尔逻辑及其扩展为代表, 并通过 SAT、SMT 等自动定理证明算法或交互式定理证明完成验证. 定理证明的优点在于不受状态空间大小的限制, 但自动化程度较低, 一般需要用户提供大量证明细节. 本文将主要关注自动化程度更高的模型检验方法与工具, 仅在相关小节简要介绍定理证明的方法.

形式化方法领域已有多篇综述文章[12,13]和相关书籍[14,15]. 与已有的综述相比, 本文的一个特点是从应用的角度出发, 关注系统的建模和性质规约语言, 以及相关的验证方法与工具. 我们希望这篇文章能够帮助工业界的系统开发人员和新进入形式化方法领域的研究人员了解形式化方法的技术体系, 并根据自己的需求更快地找到相关的文献和验证工具. 限于篇幅, 我们不会深入讨论验证算法的细节以及计算复杂性的分析. 读者可以从每节的引用更深入地了解相关的理论. 此外, 我们对程序分析(静态分析、动态分析、抽象解释等)技术的介绍相对较少, 这个领域也已有相关的综述[16]和书籍[17].

本文第 1–6 节分别介绍 6 种系统特征. 第 7–10 节分别介绍 4 种应用场景. 第 11 节对本文进行总结.

1 顺序系统

在本文中, 我们将串行执行的程序称为顺序系统. 这种程序在给定输入后, 经过计算返回输出结果. 我们关心的性质包括: 程序的功能正确性(functional correctness), 即给定合法的输入, 返回的输出结果应该满足一定的性质; 终止性, 即程序针对合法的输入是终止的. 相对于后面章节描述的系统, 顺序系统的形式和性质都比较单一. 然而, 顺序系统的正确性属于形式化方法最初研究的范畴, 仍然有非常丰富的理论, 是研究更复杂系统的基础. 本节将先后介绍顺序系统的形式语义、验证逻辑、终止性和不变式生成、软件模型检验以及验证工具.

1.1 形式语义

顺序系统一般使用程序语言来表示. 最简单的程序语言包含赋值、顺序语句、条件语句和循环语句. 更复杂的程序语言添加函数调用(包括递归调用)、指针以及类、对象、方法调用、动态绑定等面向对象语言的元素. 程序语言的形式语义为程序的行为给出数学的定义, 可以分为 3 种类型: 操作语义、指称语义和公理语义. 操作语义由 Plotkin 提出[18], 它把程序的执行描述为迁移系统, 其中, 状态为程序执行期间的变量取值和执行所在的位置. 迁移系统的每条迁移规则对应一个语句的执行, 根据规则中语句的粒度, 分为小步操作语义和大步操作语义. 指称语义在 Scott 的工作中最先提出[19], 将每个程序片段赋予一个数学对象, 其特点是该赋值通过程序结构上的递归定义, 因此每个程序的语义可以从子程序的语义计算得出. 公理语义通过定义关于程序的推理规则描述程序的行为, 其代表为霍尔逻辑, 将在下一节详细介绍.

1.2 验证逻辑

最初研究的程序验证方法使用逻辑推导的方式, 其主要思想是在程序中插入断言, 表达到达每个程序语句时系统状态应当满足的性质. Floyd 首先提出使用断言描述程序正确性的思想[20]. 在此基础上, Hoare 提出了霍尔逻辑[21], 使用霍尔三元组定义程序的部分正确性和完全正确性. 给定程序 c 和两个断言 P, Q (称为前置条件和后置条件), 部分正确性的霍尔三元组, 记为 $\{P\}\ c\ \{Q\}$, 表示如果初始状态满足 P, 并且 c 的执行终止, 那么终止状态满足 Q. 完全正确性的霍尔三元组, 记为 $[P]\ c\ [Q]$, 表示如果初始状态满足 P, 则 c 的执行肯定终止, 并且终止状态满足 Q. 因此, 完全正确性和部分正确性的区别在于: 将终止性作为一个需要满足的性质, 而不是一个假设. 同一时期, Dijkstra 提出了最弱前条件演算[22]: 给定程序 c 和后置条件 Q, 将 $wp(c,Q)$定义为最弱的可以保证 c 的执行终止并且终止状态满足 Q 的前条件. 最弱前条件演算的一个主要贡献是允许通过后向推导的方式, 在给定程序的前置条件、后置条件以及每个循环的不变式之后, 自动生成正确性验证需要的验证条件, 是众多程序验证工具的算法基础. 通过添加 Adaptation 和其他规则, 可以将霍尔逻辑推广到包含函数调用和递归调用的程序以及面向对象程序[23]. 为了处理带指针的程序, Reynolds 等人在 Hoare 逻辑的基础

上提出了分离逻辑(separation logic)[24]. 在分离逻辑中, 断言不仅描述状态满足的性质, 还可以描述当前涉及的状态在内存中的范围. 霍尔三元组$\{P\}\ c\ \{Q\}$表示程序 c 将由 P 描述的状态转换到由 Q 描述的状态, 并且不会修改任何其他状态. 分离逻辑可以有效应用于验证链表和树等数据结构的正确性, 以及程序中不存在内存错误等安全性性质. 分离逻辑也用于对面向对象程序的验证[25,26].

1.3 终止性和不变式生成

以上介绍的霍尔逻辑为程序验证提供了一个理论框架, 而不是完全自动的方法, 其主要原因是, 基于霍尔逻辑的正确性验证需要提供每个循环和递归调用的不变式. 如果还要证明程序的终止性, 则需要提供循环和递归调用的秩函数(ranking function). 不变式和秩函数的自动生成存在相当多的研究[27,28], 存在各种各样的自动生成方法. 例如, 符号计算和优化理论被广泛应用于线性和非线性程序的不变式和秩函数生成[29]. 基于插值的不变式生成最初用于硬件系统的验证[30], 但也可以扩展到软件验证中. 基于学习的不变式生成方法包括 ICE[31]等. 近期, 以数据驱动为基础的神经网络和深度学习方法也被用于不变量生成中[32,33]. 对于一些线性情况, 存在完备的秩函数合成算法[34]. 针对一般的循环程序, Terminator[35]使用反例制导的精化方法(CEGAR), 对于每个可能的程序路径, 判断是否有对应的秩函数, 否则将生成反例, 并从反例生成新的秩函数. Ultimate Automizer 使用 Büchi 自动机来表示程序的执行路径, 通过计算 Büchi 自动机的集合来覆盖程序的所有路径, 从而判断程序终止性[36].

1.4 软件模型检验

模型检验技术也可用于顺序系统的验证. 针对一般应用程序进行模型检验的研究称为软件模型检验[37]. 软件模型检验的主要难点在于需要处理庞大的, 甚至无穷大的状态空间, 以及来自整数等数据类型和可能递归的函数调用. 在软件模型检验中, 程序通常使用两种模型表达: 递归状态机[38]和下推系统[39]. 结合程序分析中使用的摘要(summarization)和饱和(saturation)方法, 可以在这些模型上进行模型检验[39,40]. 软件模型检验的另一个问题是如何设计具有足够表达能力, 又存在高效的模型检验算法的性质描述语言. 基于嵌套词(nested word)的描述语言在表达能力和验证效率之间提供了一个平衡[41]. 这个描述语言可以表达类似于霍尔逻辑的性质, 即如果在进入某个函数调用时状态满足 P, 则从函数调用返回后状态满足 Q. 该语言也可以描述更一般的性质, 例如函数 P 的调用只能出现在函数 P'的调用中等等.

1.5 程序验证工具

目前有许多程序验证工具以霍尔逻辑及其扩展为基础, 这里仅列出几个代表. VeriFast[42]支持验证带有分离逻辑断言标注的 C 程序或 Java 程序, 它支持检查数组越界、内存访问错误等; 并且支持使用归纳数据类型、不动点等来定义谓词, 描述复杂的数据结构性质, 从而验证程序是否满足分离逻辑断言描述的性质. 其他基于霍尔逻辑的程序验证工具包括 Dafny[43]、Why3[44]等. Frama-C[45]用于工业级 C 代码的验证, 支持以 ACSL 规范语言描述的功能性质, 包括以 requires、ensures、assigns、loop invariant 等标注的性质, 以及基于逻辑的推理验证.

软件模型检验最初的一个成功案例是 SLAM[46], 在微软公司用于验证驱动器满足 API 的使用规则. 该工具使用抽象精化的方法, 将待验证程序转化为布尔程序, 并采用上述的基于下推系统的模型检验方法. 基于分离逻辑的 Infer 工具在 Facebook 公司应用于代码库中寻找内存错误, 通过迭代分析代码的修改, 提高分析效率, 使其能够应用于上百万行的代码库[47].

2 反应式系统

顺序系统的正确性只关注系统的输入和输出, 然而对于一些系统, 我们不仅关注输入和输出, 还关注系统的整个运行过程. 我们称这类系统为反应式系统(reactive system), 它在运行过程中持续与环境进行交互, 而不是生成某个计算结果. 反应式系统的运行过程可以表示为由状态(state)和事件(event)构成的序列, 时序性质则是描述这些序列的性质. 本节将介绍反应式系统的行为模型、性质规约语言、时序性质的分类, 以及时

序性质的验证工具.

2.1 行为模型

反应式系统可以使用迁移系统建模. 迁移系统[48,49]经常被用于描述系统行为, 使用有向图来表示: 有向图的点对应状态, 有向图的边对应状态迁移. 状态可描述系统在某一时刻所有变量的值, 状态迁移表示从一个状态可以转换到另一个状态. 状态迁移有时还与执行的动作相关, 即在某一状态下执行某一动作可以转换到另一个状态. 另外, 状态往往有其对应的原子命题. 下面我们给出迁移系统的形式化表示. 迁移系统由元组$\langle S,Act,\rightarrow,I,AP,L\rangle$表示, 其中, S 是状态集, Act 是动作集, $\rightarrow\subseteq S\times Act\times S$ 是迁移关系, $I\subseteq S$ 是初始状态集, AP 是原子命题的集合, $L:S\rightarrow 2^{AP}$ 将状态映射到对应的原子命题的集合. 迁移系统的路径定义为系统状态的序列. 对于一个路径片段$\pi=s_0s_1s_2\ldots$, 定义其迹为 $trace(\pi)=L(s_0)L(s_1)L(s_2)\ldots$ 将状态标记为原子命题集合的迁移系统也称作 Kripke 结构.

2.2 性质规约语言

反应式系统所关注的性质是时序性质, 即对迁移系统的执行路径的要求. 我们可以用自动机[50−53]或时序逻辑[54]对时序性质进行精确的数学表达. 自动机是一个可以接受输入序列的系统, 接受的输入序列集合可以对应迁移系统路径的迹的集合, 故可用自动机来表示迁移系统需要满足的时序性质. 时序逻辑是对谓词逻辑的扩展, 在谓词逻辑的基础上加入了模态词, 用以描述执行路径上的状态关系. 例如, 在一条路径上, 某一状态出现在另一个状态之后. 在时序逻辑中, 可用线性或者分支的方式对其进行刻画, 分别对应线性时序逻辑(linear temporal logic, LTL)[54]和计算树逻辑(computation tree logic, CTL)[55].

2.2.1 正则性质和自动机

如果一个时序性质可以被有穷状态自动机表示, 则这个性质是正则性质. 正则安全性质可以用非确定性有穷自动机(NFA)来表示其“坏前缀”; 其他正则性质可以用非确定性 Büchi 自动机(NBA)[50]来表示.

非确定性有穷自动机是用来接受有穷输入序列的自动机, 可被形式化地表示为元组$\langle Q,\Sigma,\delta,Q_0,F\rangle$, 其中, Q 是有穷状态的集合, $\Sigma=2^{AP}$, $\delta:Q\times\Sigma\rightarrow 2^Q$ 表示迁移函数, Q_0 是初始状态的集合, F 是接受状态的集合. 对于自动机 A 和输入序列 $w=A_1A_2\ldots A_n$, 如果存在一条路径$\pi=q_0q_1q_2\ldots q_n$, 满足 $q_0\in Q_0$, $q_i\xrightarrow{A_{i+1}}q_{i+1}$, $q_n\in F$, 则 w 是可被自动机 A 接受的, 所有可被自动机 A 接受的输入序列的集合被称为 A 的接受语言. 在验证迁移系统 TS 是否满足某一正则安全性质时[56], 可构造一个非确定性有穷自动机 A, 使其接受语言是该正则安全性质的所有“坏前缀”的集合, 将 TS 与 A 相乘得到一个新的迁移系统, 对该迁移系统进行不变式检查, 检查其是否满足从不到达 A 的接受状态: 若满足, 则满足该正则安全性质.

非确定性 Büchi 自动机可用于接受无穷输入序列, 其语法与 NFA 相同, 不同之处在于其接受语言的定义. 对于非确定性 Büchi 自动机 A 和输入序列 $w=A_1A_2\ldots$, 如果存在一条对应路径, 使其可以无穷次访问接受状态, 那么 w 是可被自动机 A 接受的, 所有可被自动机 A 接受的输入序列的集合被称为 A 的接受语言. 在验证迁移系统 TS 是否满足某一正则性质 P 时[57], 可构造一个非确定性 Büchi 自动机 A, 使其接受语言是$\neg P$, 将 TS 与 A 相乘, 得到一个新的迁移系统, 对该迁移系统进行持续性检查, 检查其是否满足从某一时刻起不再到达 A 的接受状态: 若满足, 则满足该正则性质 P.

2.2.2 线性时序逻辑

线性时序逻辑从线性的角度考虑性质, 即每一时刻只考虑一个后继的状态. 因此, LTL 公式定义在系统的单条路径上. LTL 的语法如下:

$$\varphi::=\text{true}|a|\varphi_1\wedge\varphi_2|\neg\varphi|\circ\varphi|\varphi_1\text{U}\varphi_2,$$

其中, $a\in AP$ 是原子命题. LTL 在命题逻辑的基础上增加了○和 U 两个运算符, 读为“下一个”和“直到”. 性质$\circ\varphi$表示从下一个时刻开始的路径满足φ. 性质$\varphi_1\text{U}\varphi_2$表示存在一个时刻 i, 在 i 之前路径满足φ_1, 而在时刻 i, 路径满足φ_2. 在此基础上, 衍生出◊和□两个算符, 分别表示“最终满足”和“一直满足”. 将上述运算符组合, 可以得到更复杂的性质. 例如, 性质$\Box\Diamond\varphi$表示“无穷次满足φ”; 性质$\Diamond\Box\varphi$表示“最终一直满足”; 性质$\circ\Box\varphi$表示“从下一

时刻开始, 总是满足φ", 等等. 对于交通信号灯系统, 令 AP={red,yellow,green}, 则性质"无穷次出现绿灯"可以表示为□◊green.

LTL 可以使用基于自动机的方法进行模型检验[53]. 从任何一个 LTL 公式φ, 可以构造出一个非确定性 Büchi 自动机 A, 使其接受语言正好是满足¬φ的路径的集合. 然后, 将 TS 与 A 相乘得到一个新的迁移系统, 若这个新的迁移系统中存在一条路径满足自动机 A 的接受状态, 则在原系统中找到了一条满足¬φ的路径, 即原系统不满足性质φ; 若找不到这样的路径, 则原系统满足φ. LTL 模型检验的复杂度与原迁移系统的大小成正比, 与性质的公式长度成指数关系.

2.2.3 计算树逻辑

计算树逻辑从分支的角度考虑性质. 在迁移系统中, 一个状态可以对应多个后继状态, 因此, 从一个状态出发可以有多条路径, 形成一个树状结构. 在这种树状结构上定义性质, 即分支时序逻辑[58]. CTL 是一种典型的分支时序逻辑, 引入了存在量词(∃)和全称量词(∀), 其语法定义包括状态公式ϕ和路径公式φ两部分:

$$\phi:=\text{true}|a|\phi_1\wedge\phi_2|\neg\phi|\exists\varphi|\forall\varphi,$$

$$\varphi:=\circ\phi|\phi_1\text{U}\phi_2,$$

其中, ∃φ表示从该状态出发存在一条路径满足φ, ∀φ表示从该状态出发所有的路径都满足φ.

验证迁移系统 T 是否满足 CTL 公式ϕ, 可以通过递归计算满足公式ϕ的每个子公式的状态集合, 然后判断 T 的所有初始状态是否包含在该集合中[55,59−61]. CTL 模型检验的复杂度与原迁移系统和公式的长度都成线性关系. 然而, 这并不是说 CTL 性质相比于 LTL"更容易检验": LTL 在某些性质的表达上更简练, 对于一些 LTL 性质, 可能需要指数长的 CTL 公式才能表达相同的性质.

2.2.4 CTL*

CTL*[62]是对 CTL 的扩展. 在 CTL 中, 每个时序运算符○和 U 前面必须紧跟路径量词(∃, ∀), 但在 CTL*中, 时序运算符和路径量词可以任意嵌套. 例如, ∀○○a 在 CTL 中不合法, 但在 CTL*中合法.

LTL 与 CTL 的表达能力无法比较, 即有些性质可以用 LTL 表达但不能用 CTL 表达, 而有些性质可以用 CTL 表达但无法用 LTL 表达. CTL*的表达能力强于 CTL 和 LTL, 且 CTL*还能表达 LTL 和 CTL 无法表达的性质.

2.2.5 μ-演算

μ-演算是一种重要的模态逻辑, 用于迁移系统性质的刻画与验证. μ-演算的表达能力比 CTL*强, 与一元二阶逻辑(monadic second order logic)在迁移系统上互模拟不变子集相同[63]. 从博弈论的角度看, μ-演算可以归约成双人完全信息下的奇偶性游戏(parity game), 可以借助这一归约得到良好的算法性质[64].

μ-演算的主要运算符是最小不动点算子μ与最大不动点算子ν. 例如, $\mu X.\alpha$解释为最小的状态集合子集 S 满足$\alpha(S)=S$. 根据μ-演算中逻辑运算符的单调性以及完全格上的不动点存在定理, 可知这两个算子是良定义的[65].

μ-演算公式的验证归结为公式的语义解释, 难点在于不动点算子的计算. 对于有穷迁移系统, 一种不动点算子计算方法是迭代法, 如计算$\mu X.\alpha$, 记空集为 S_0, 并由 $S_{i+1}=\alpha(S_i)$得到一个收敛到最小不动点 S'的序列, 复杂度较高. 且对于包含无穷状态数的迁移系统, 会涉及无穷次迭代, 需要借助序数(ordinal number)的概念[66]. μ-演算公式的表达能力与公式中μ算子和ν算子交错出现的次数有关, 这一参数称为交错深度(alternating depth), 可以仿照交错有穷自动机根据交错深度以及第 1 个出现的不动点算子得到表达能力的严格层次结构[67].

关于μ-演算的模型检测算法复杂度一直是相关研究中的重要问题, 给定一个包含 n 个状态的迁移系统和交错深度为 d 的公式, 基于迭代的模型检测算法复杂度是 $O(n^{d+1})$. 但从博弈的角度看, μ-演算公式的模型检测问题可以归结为奇偶性游戏必胜策略的存在性问题, 可以显著降低算法的复杂度, 对算法复杂度下界的探索, 一直是μ-演算相关研究中的重要问题[68−70].

2.3 性质的分类

2.3.1 安全性与活性

时序性质中有两类特殊的性质特别受到关注: 安全性(safety)[71]与活性(liveness)[72]. 安全性是指“坏的事情不会发生”, 比如“不会有两个进程同时进入互斥区”; 活性是指“好的事情一定会发生”, 比如“交通信号灯会出现绿灯”. 对于线性时间性质, 安全性和活性可以定义如下.

- 一个性质是安全性质, 如果性质的违反总能在有限步内发现. 也就是说, 对于任何违反安全性质 P 的路径σ, 总是存在σ的一个前缀σ', 使得任何σ'的延伸都违反 P. 当执行到σ'时, 我们已经可以肯定σ违反了性质. 常见的安全性质如: “每一次红灯出现后总是立即出现绿灯”.
- 一个性质是活性性质, 如果性质的违反永远无法在有限步内发现. 也就是说, 对于任何路径的前缀σ', 总是能够找到σ'的一个延伸σ, 使得σ满足 P. 常见的活性性质如: “绿灯总是会出现”或“绿灯将会出现无穷多次”.

除了理论意义之外, 安全性和活性的分类在现实应用中也有指导作用. 一般来说, 安全性和活性的验证方法不同. 安全性的验证一般通过寻找系统的不变式, 而活性的验证则需要寻找秩函数. 由于安全性的违反总能在有限步内察觉, 系统的公平性假设对安全性的验证没有帮助. 然而, 在很多情况下, 活性的验证需要对系统做出公平性假设.

安全性和活性的定义还被扩展到分支时间(计算树)的场景[73]. 在这个场景下有 4 种类型: 一致安全(universally safe)、一致活性(universally live)、存在安全(existentially safe)和存在活性(existentially live). 这 4 种类型可以简单解释为“所有路径坏的事情不会发生”“所有路径好的事情一定发生”“存在路径坏的事情不会发生”, 以及“存在路径好的事情一定发生”. 严格的定义需要将线性时间性质所用的路径之间的前缀和延伸关系扩展到计算树之间的前缀和延伸关系.

对于线性时间性质, 任意性质 E 都可以表示成一个安全性质和活性的交集, 即存在一个安全性质 $SAFE$ 和活性 $LIVE$, 使得 $E=SAFE\cap LIVE$. 对于分支时间性质, 任何性质 E 都可以表示为存在安全和存在活性的交集、一致安全和一致活性的交集, 以及存在安全和一致活性的交集.

2.3.2 线性时间性质的层次结构

除了安全性与活性的分类之外, 还有其他的线性时间性质的分类方法. Manna 等人[74]提出了线性时间性质的层次结构, 包含安全(safety)、保证(guarantee)、重现(recurrence)、持续(persistence)、义务(obligation)和反应(reactivity)这 6 种类型. 这 6 种类型之间的包含关系如图 1 所示. 设 p 为一个仅包含过去算子的性质, 安全性质表示一直满足 p(即$\Box p$); 保证性质表示最终满足 p(即$\Diamond p$); 重现性质表示无穷次满足 p(即$\Box\Diamond p$); 持续性质表示最终一直满足 p(即$\Diamond\Box p$); 义务性质由安全性质和保证性质经过任意布尔组合(交、并、取补)得到; 反应性质由重现性质和持续性质经过任意的布尔组合得到, 具有最强的表达能力.

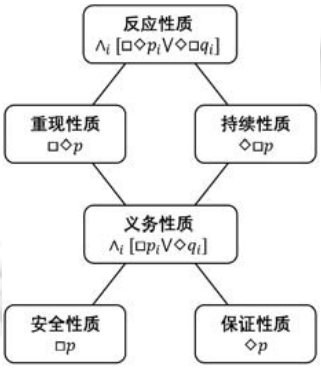


图 1 时序性质间的包含关系[74]

2.4 验证工具

目前, 常用的反应式系统的时序性质验证工具有 NuSMV[75]、SPIN[76]、Murphi[77]、TLA$^+$[78]等.

NuSMV 用于 LTL 模型检验、CTL 模型检验和限界模型检验, 是对 SMV 工具的再实现和改进. NuSMV 整合了基于 BDD 的符号模型检验和基于 SAT 的限界模型检验, 接受自动机模型作为输入, 验证该模型是否满足由 LTL、CTL 描述的性质, 在不满足时给出反例. NuSMV 可用于验证通信协议[79,80]、硬件协议标准[81,82]是否存在漏洞, 以及验证硬件系统[83,84]、软件系统[85]是否满足正确性性质等. 例如, Clarke 等人[82]使用 SMV 验证了 Futurebus+缓存一致性协议, 并且找到了该协议中存在的此前未被发现的漏洞; Anderson 等人[85]使用 SMV 验证了飞机避碰系统是否满足 TCAS II (traffic alert and collision avoidance system II)的规约要求.

SPIN (simple promela interpreter)是用于验证并发系统和分布式系统的开源工具, 起源于贝尔实验室形式化方法与验证小组开发的 PAN. 在 SPIN 中, 使用 Promela (process meta language)语言对需要验证的系统进行建模. SPIN 支持验证安全性和活性、是否存在死锁、是否违反给定断言等性质. SPIN 可用于验证分布式算法[76,86]、进程调度协议[87]、通信协议[88−90]等的正确性. 例如, Ruane 等人[87]使用 SPIN 验证了一个操作系统进程调度协议的正确性; Holzmann[86]使用 SPIN 验证了领导人选举算法的正确性, 保证最多只有 1 人被选为领导人.

Murphi 是一种描述语言, 也是用于验证有限状态并发系统的验证工具. 使用者需先用 Murphi 语言对协议建模, 进而使用 Murphi 工具验证该协议是否满足性质. 性质一般表示为不变式. Murphi 已被成功应用于许多实际问题的验证, 特别是微处理器缓存一致性问题[91−93], 这是 Murphi 在设计时面向的主要问题, 验证发现了 IEEE/ANSI 的缓存一致性协议标准实现中存在的几处 C 代码错误. 另外, Murphi 也被用于通信协议的验证[94], 例如, 使用 Murphi 可在几秒内发现 Needham-Schroeder 协议[95]存在的漏洞, 并且发现了 TMN 密钥分发协议[96]的漏洞.

基于动作的时序逻辑(temporal logic of actions, TLA)[97]是由 Leslie Lamport 提出的系统规约语言, 主要用于分布式系统与并行系统的开发与验证. TLA 的一个特点是在规约语言中不仅可以使用关于状态的谓词, 还可以包含动作, 因此可以表述“当动作 A 发生时, 动作 B 一定会在将来发生”等性质. TLA 语言同时用来描述系统行为和性质, 从而验证问题可以转化为验证逻辑公式 $System \rightarrow \Box Property$ 的正确性. TLA$^+$在 TLA 基础上引入了多种数据结构和数学符号以方便使用. TLC 是 TLA$^+$的模型检验工具, 支持验证不含时序存在量词的一大类性质, 目前, TLA$^+$正在逐渐被工业界接纳, 包括英特尔、微软、亚马逊在内的诸多公司已经尝试将 TLA$^+$工具集成在系统设计的流程中[98,99], 如云服务器的分布式系统、芯片上的一致性协议等, TLA$^+$能够发现其他技术难以检测到的漏洞, 帮助完善系统设计.

3 并发与通信系统

当多个系统并发执行时, 它们通过共享数据或者通道通信的方式来相互沟通信息. 对于共享数据, 需要关注的性质是不发生数据竞争, 即一个线程在使用某个共享数据时, 另一个线程不能对这个数据进行修改. 避免数据竞争的一个常见方法是在共享的数据上加锁, 通过获取锁和释放锁来控制共享数据的访问和修改. 另一个性质是公平性. 当线程的执行顺序通过一个调度器决定时, 调度策略需要保证每个线程都能分配到一定的执行时间, 没有线程出现饿死的情况, 这称为调度策略的公平性. 公平性还体现在锁和信号量分配的策略中, 保证每个线程都有机会获得这些资源.

基于通道通信的并发具有多种实现方式.

- 同步通信, 即发送方和接收方必须同时准备好执行时, 同步通信才能发生.
- 异步通信, 即发送方不需要等待总是立刻发生, 而只要发送的数据集合不为空, 接收方随时从发送的数据中接收数据. 当发送的数据为空时, 接收方需要等待.
- 广播通信, 即发送方总是立刻发生, 同时, 所有同通道上可行的接收方都与发送方同步发生.

对于基于通道通信的并行系统, 关注的性质有无死锁、无活锁.

(1) 无死锁: 在并发程序中, 2 个或多个线程为了获取信号量或其他资源相互等待其他进程, 造成所有进程都无法继续执行, 这称为死锁. 系统设计的问题, 例如线程忘记释放锁, 或不合理的获取锁的顺序, 都可能造成死锁情况.

(2) 无活锁: 当 2 个或多个线程之间无限次地重复通信, 而无法在实际任务中取得进展, 这称为活锁. 不合理的线程间通信的协议可能造成活锁现象.

对于不同的并发与通信的建模机制, 相应的性质规约和验证方法差别很大. 进程代数是一种常用的系统建模的方法. 它使用代数公式来描述系统模型, 非常适合描述系统的并发和通信特征. 目前, 研究最多的进程代数语言包括通信系统演算 CCS 和通信顺序进程 CSP. 我们下面将首先介绍以进程代数为基础的并发通信系统, 然后介绍 Petri 网以及并发系统的验证工具.

3.1 通信系统演算

通信系统演算(calculus of communicating systems, CCS)由 Robin Milner 在 20 世纪 70 年代提出[100], 是进程代数领域的开创性工作. CCS 的基本成分是动作和进程, 每个进程都有自己的动作. CCS 的动作分成两类.

- 一类是外部动作, 用于描述进程间通信的同步和交互. 外部动作发生在一个系统的两个进程之间, 因此通常成对出现, 其中一个进程产生发送信号的动作, 相应地, 另一个进程产生接收信号的动作, 我们通常用 a 和 $\bar{a}$ 分别表示动作的发送和接收. 例如, 有一个自动贩卖机系统, 想要通过机器购买的顾客也是一个系统, 顾客的购买行为可以看作发送行为, 自动贩卖机的售出行为可以理解为接收信号的动作.
- 另一类动作是内部动作, 即非交互动作, 由单独一个进程产生, 与环境无关, 通常用 τ 表示. 例如, 上述自动贩卖机的商品页面切换动作就可以理解为系统的内部动作.

外部动作和内部动作构成了 CCS 的动作集. CCS 进程由并发系统中比较直观的语法操作构成, 包括动作前缀、选择操作、并发操作、限制操作和重命名操作等.

CCS 通过定义结构操作语义来定义其语义, 表示为一个标号迁移系统. 每个迁移关系形式为 $p \xrightarrow{a} p'$, 表示进程 p 执行一个动作 a 后, 到达进程 p'. 对于动作前缀 $a.p$, a 是一个动作, $a.p \xrightarrow{a} p$ 说明进程 $a.p$ 在执行动作 a 后到达进程 p, 之后会执行进程 p 中的相关动作. 我们用“+”表示选择操作, 在系统 $p+q$ 中, 我们选择执行一个动作, 这个动作要么是 p 进行执行, 要么是 q 进行执行. 若有 $p \xrightarrow{a} p'$, 整个系统 $p+q$ 选择 p 系统部分, 执行动作 a 后就到达进程 p'的状态; 同理, 若有 $q \xrightarrow{a} q'$, 整个系统 $p+q$ 选择 q 系统部分, 执行动作 a 后就到达进程 q'的状态. 例如, 考虑系统 $a.b.nil+a.c.nil$, 这里的 nil 表示成功完成了执行. 接下来系统执行动作 a, 虽然左右两个子系统都可以执行动作 a, 但选择操作只会选择其中 1 个子系统, 因此接下来的系统要么到达 $b.nil$ 的状态, 要么到达 $c.nil$ 的状态. 对于并发操作, 我们用 $p|q$ 表示两个子系统 p 和 q 的并发, 若动作 a 只对应在 1 个系统中执行, 如有 $p \xrightarrow{a} p'$, 则 $p|q \xrightarrow{a} p'|q$, q 动作的执行类似. 而当动作 a 在两个子系统中都可以执行时, 即有 $p \xrightarrow{a} p'$ 以及 $q \xrightarrow{a} q'$, 则这个动作 a 变成了整个系统的一个内部动作, 与环境无关, 相应地有 $p|q \xrightarrow{\tau} p'|q'$.

3.1.1 强互摸拟

在 CCS 中, 互模拟等价关系是一个非常重要的概念, 是 CCS 系统验证的基础. 下面介绍相关的概念.

我们用$\langle Q,A,\rightarrow\rangle$表示一个迁移系统, 其中, Q 是 CCS 的进程集合, A 是动作集合, 内部动作$\tau\in A$, 定义 $R\subseteq Q\times Q$ 为互模拟关系. 给定两个系统 p 和 q, p 和 q 满足强互模拟关系, 记$\langle p,q\rangle\in R$, 如果对于任意动作 a 和 $p',q'\in Q$, 下面两条性质成立.

1. 若存在动作 a 使得 $p \xrightarrow{a} p'$, 则有 $q \xrightarrow{a} q'$ 满足$\langle p',q'\rangle\in R$;

2. 若存在动作 a 使得 $q \xrightarrow{a} q'$, 则有 $p \xrightarrow{a} p'$ 且满足$\langle p',q'\rangle\in R$.

若$\langle p,q\rangle\in R$, 则记作 $p\sim q$, 表示状态 p 和 q 满足强互模拟等价关系. 从直观上看, 这种互模拟等价关系可以理解成一种执行行为上的完全等价关系.

基于强互摸拟关系的性质规约可以使用 Hennessy-Milner 逻辑(HML)[101]来表示. 它能够描述系统是否可能出现死锁等性质. 这里, 我们对 HML 做一个简单的介绍. 给定一个动作集 A, 令 $a\in A$, HML 的语法公式定义如下:

$$\phi:=tt|ff|\phi\wedge\phi|\phi\vee\phi|[a]\phi|\langle a\rangle\phi.$$

前面 4 个公式的定义和传统逻辑一致. 对于最后两个公式: $[a]\phi$表示一个状态执行动作 a 后到达的所有状态都满足性质ϕ; $\langle a\rangle\phi$表示一个状态执行动作 a 后到达的所有状态中, 存在一个状态满足性质ϕ. 例如, 我们可以用序列$\langle a_1\rangle\langle a_2\rangle\ldots\langle a_n\rangle tt$ 表示整个系统中存在一条 $a_1a_2\ldots a_n$ 的动作序列. 公式$([b_1]ff\wedge[b_2]ff\wedge\ldots\wedge[b_n]ff)$表示一个状态被 $b_1,b_2,\ldots,b_n$ 里的所有动作拒绝(即无法执行 $b_1,b_2,\ldots,b_n$ 中的任意动作), 若满足集合关系 $b_1\cup b_2\cup\ldots\cup b_n=A$, 则表明该状态进入了一个死锁状态(即无法执行任何动作), 这样我们可以对一个系统的无死锁性质进行验证.

HML 与上述强互摸拟性质的关系为: 系统 p 和 q 满足强互摸拟关系, 当且仅当对于所有的 HML 公式, p 和 q 要么同时满足, 要么同时不满足.

例如, 对于系统 $a.b.nil+a.b.nil$ 和系统 $a.b.nil$, 根据强互摸拟的定义, 这两个系统是满足强互摸拟的, 对应地, 对于一个 HML 公式, 如$[a][b]tt$, 两个系统均满足这个 HML 公式. 对于两个系统 $a.b.c.nil+a.b.d.nil$ 和 $a.b.(c.nil+d.nil)$, 根据强互摸拟的定义, 这两个系统是不满足强互摸拟的, 当前面一个系统需要依次执行动作 a, b, c 时, 由于选择操作可能选择了 $a.b.d.nil$, 该系统可能会出现死锁的情况; 而对于后面这个系统, 由于选择操作发生在执行动作 a 和 b 之后, 因此不会产生死锁. 从 HML 公式满足性来看, 存在一个 HML 公式$[a][b]\langle c\rangle tt$, 前一个系统不满足该公式, 后面的系统是满足该公式的.

3.1.2 弱互模拟

除了上述定义的互模拟关系以外, 在 CCS 中还通常使用观测等价性, 它忽略内部动作的行为. 首先定义几个概念. 给定 $s\in A^*$, 其中, A^*表示任意长度的动作序列, 那么 $\hat{s}\in(A-\{\tau\})^*$ 定义为从 s 动作序列中删除所有内部动作τ后得到的动作序列. 给定 $s\in(A-\{\tau\})^*$, 定义 $q\overset{s}{\Rightarrow}q'$, 表示 q 可以通过一个外部动作序列 s 迁移到 q', 但是中间过程中可能有任意数量的内部动作. 给定两个进程 p 和 q, p 和 q 满足弱互模拟关系 R, 记作 $p\approx q$, 如果对于任意动作 $a\in A$ 和 $p',q'\in Q$, 满足下面两条性质.

1. 若存在动作 a 使得 $p\xrightarrow{a}p'$, 则有 $q\overset{\hat{a}}{\Rightarrow}q'$ 且满足$\langle p',q'\rangle\in R$;
2. 若存在动作 a 使得 $q\xrightarrow{a}q'$, 则有 $p\overset{\hat{a}}{\Rightarrow}p'$ 且满足$\langle p',q'\rangle\in R$.

从直观上看, 由于内部动作是环境中不可观测的, 故这个弱等价关系也被称为观测等价性. 它是一种更加宽泛的互模拟关系.

MWB 是一个基于 CCS 和π-演算的并发系统分析与验证工具[102,103], 由 Uppsala 大学开发. 对于一个并发系统, 需要在 MWB 中输入并发系统的 CCS 公式, 然后, MWB 会对 CCS 描述的进程进行分析和验证. MWB 支持通过输入相关指令进行死锁检测、进程等价检测等.

3.2 通信顺序进程

通信顺序进程(communicating sequential processes, CSP)由 C.A.R Hoare 提出 [104]. CSP 与 CCS 类似, CSP 在语法中同样定义了动作和进程, 但与 CCS 不同的是, CSP 语言语法上不再区分同一个动作的发送和接收, 而通过“?”“!”分别表示输入和输出动作; 同时, CSP 不再显式表示内部动作, 加入了外部选择和内部选择. 在文献[105,106]中, 分别对 CSP 和 CCS 从不同的角度进行了比较.

CSP 的语法规则可以定义如下:

$$\text{STOP}|\text{SKIP}|a\rightarrow P|P\sqcap Q|P\Box Q|P||Q|P\backslash a|\mu X.F(X),$$

其中,

- STOP 表示死锁的系统.
- SKIP 表示系统成功完成了执行.

- $a\rightarrow P$ 表示前缀操作, 与 CCS 语义是一致的.
- $P\sqcap Q$ 和 $P\Box Q$ 分别表示内部和外部选择: 这里的内部操作由系统内部自己选择, 与环境无关; 而外部选择由环境动作直接决定. 例如, 一个自动贩卖机进行找零, 找零 1 块钱时, 返回一个 1 块钱的硬币还是两个 5 角的硬币由系统内部直接决定, 而贩卖机弹出哪个商品是由顾客的选择决定的.
- $P\|Q$ 表示系统的并行操作.
- $P\backslash a$ 表示对动作 a 进行隐藏, 使其变为内部动作.
- $\mu X.F(X)$表示系统一直在 F 操作中进行递归.

CSP 具有多种不同的指称语义[107], 主要包括: 迹语义(trace semantics), 即进程可以执行的事件序列的集合; 失败迹语义(failure semantics), 即在执行一个序列后, 进程可以拒绝发生的事件集合.

对于系统$(a\rightarrow \mathrm{STOP})\Box(b\rightarrow \mathrm{STOP})$和系统$(a\rightarrow \mathrm{STOP})\sqcap(b\rightarrow \mathrm{STOP})$而言, 它们的迹集合是完全相等的, 均为$\{\langle\cdot\rangle,\langle a\rangle,\langle b\rangle\}$, 即可以不执行任何动作, 或者执行 a 或 b 中的一个动作. 但是这两个系统的失败迹集合是不相同的, 用Σ表示所有动作的集合, 由于前一个动作是外部选择操作, 因此动作 a 和 b 都会使得系统继续执行, 因此其失败集合是$\Sigma\backslash\{a,b\}$, 而由于后面这个动作是内部选择, 系统只能执行 a 或者 b 中的一个动作, 因此系统的失败集为$\Sigma\backslash\{a\}$和$\Sigma\backslash\{b\}$.

在失败迹语义的基础上, 失败-发散迹语义(failure-divergence semantics)对其进一步进行了扩充, 即除了失败迹以外, 还刻画进程可以执行无穷τ动作的发散迹的集合. 例如, 系统$(\mu X.a\rightarrow X)\backslash a$ 表示一个系统无限执行动作 a, 若加上隐藏操作$(\mu X.a\rightarrow X)\backslash a$, 动作 a 变成了内部动作, 从迹集合和失败集合的角度, 该系统和 STOP 是完全相同的; 但从发散迹集合来说, STOP 的发散迹集合是$\varnothing$, 而$(\mu X.a\rightarrow X)\backslash a$ 的发散迹集合为任意的动作序列的集合(因为可以无限执行内部动作 a).

这 3 种语义的表达能力依次增强, 在确定语义模型之后, 可以定义精化的概念. 给定两个进程 P 和 Q, Q 是 P 的迹精化(trace refinement), 记为 $P\sqsubseteq_T Q$, 如果 Q 的迹集合是 P 的迹集合的子集. 类似地, 可定义失败迹精化 $P\sqsubseteq_F Q$ 以及失败-发散迹精化 $P\sqsubseteq_{FD} Q$. 接下来, 可以通过指称语义以及对应的精化关系进行模型验证.

首先, 性质本身定义为一个 CSP 规范, 记为进程 *Spec*, 行为实现定义为进程 *Imp*, 那么实现满足规范表示为 $Spec\sqsubseteq Imp$, 从而可以根据不同的指称语义及其对应的精化关系进行验证. 无死锁、无活锁、确定性等性质都可以转化为对应的 CSP 进程以及精化关系进行验证.

CSP 还被扩展到实时和混成系统领域中[108].

3.2.1 FDR

FDR (failures divergence refinement)[109]是一个应用非常广泛的模型验证工具. FDR 是基于 CSP 进行开发的, 可以验证进程之间的精化关系, 通常用于并发系统的模型验证. FDR 用 $\mathrm{CSP_M}$ 语言(机器可读取的 CSP)对模型和性质进行描述, 并自动验证 *Spec* 和 *Imp* 是否满足精化关系. 一般而言, 在 FDR 中需要人工写出并发系统行为以及规约性质对应的进程. 通过定义相应的精化关系, FDR 可以用于检测 CSP 的无死锁、无活锁、确定性等性质. FDR 还支持对时间 CSP (timed CSP, CSP 的实时扩展)的模型验证[110,111].

FDR 已经成功验证了很多通信协议, 若协议满足所需要验证的安全性质, 则 FDR 正常退出; 否则, FDR 会给出一种不满足给定安全性质的攻击. 在实际的应用中, Donovan 等人[112]通过 FDR 和 Casper 对 ISO Public Key Two-Pass Mutual Authentication 协议进行了分析, 证明了该协议不满足身份认证安全性质, 并给出了一种身份伪装攻击方式. 研究者还使用 FDR 验证了很多经典通信协议的安全性质, 例如, Brackin[113]验证了 ISO Two-Pass Unilateral Authentication with CCFs 和修改后的 Needham-Schroeder 协议满足隐私性以及身份认证安全性质, 找到了 Needham-Schroeder 签名协议不满足身份认证安全性质的反例, 等等. 总的来说, FDR 能够对安全通信协议的安全性质进行验证, 并且能够给出不安全的协议的一种攻击.

3.3 Petri网

Petri 网是对离散并行系统的数学表示, 由 Carl Adam Petri 于 1962 年提出, 是针对复杂系统进行模型分析、验证、仿真以及性能分析的重要工具[114,115]. 形式化地, 一个 Petri 网 PN 表示为一个三元组(P,T,F), 其中, P 表示有限位置(place)的集合, T 表示有限迁移状态(transition)的集合, $F\subseteq(P\times T)\cup(T\times P)$表示流迁移关系的集合. F 包含了从 P 转移到 T 的输入集合 $In(t):=\{p|(p,t)\in F\}$以及从 T 迁移到 P 的输出集合 $Out(t):=\{p|(t,p)\in F\}$.

在一个 Petri 网中, 定义格局(configuration)c 为一个 Petri 网的状态, c 定义了在每个位置 p 下令牌(token)的数量. 这里的令牌数通常是系统中进程的数目、共享变量的数目等. 通过 Petri 网迁移的执行, 不同位置 p 中令牌的数量会发生变化. 并发系统可以使用 Petri 网形式定义.

Petri 网具有很强的表达能力, 可以验证系统的多种性质, 其主要的验证问题如下.

1. 可达性(reachability)问题: 判断一个系统是否能够达到一个特定的状态, 或者表现出某种特定功能.
2. 可覆盖性问题[116]: 给定初始状态和目标状态, 是否存在一个从初始状态可达的状态, 使得该可达状态大于等于目标状态. 如果存在这样的可达状态, 则目标状态是可覆盖的; 否则, 是不可覆盖的. 它与可达性问题是可以互相转换的.

在使用 Petri 网进行性质验证时, 主要包含两种方法.

- 一种是覆盖树(the coverability tree)[117]方法. 该方法基于递归搜索进行验证, 首先给出一个 Petri 网的初始结构 c, 这样, 由于迁移状态集合 T 的大小是有限的, 位置集合 P 的大小也是有限的, 因此只需遍历并枚举所有可以达到的结构 c 的可能情况即可. 对于 p 中具有无限令牌的情况, 引入参数ω表示无穷数量即可. 这样, 对任意的一个 Petri 网, 都能构造出一棵有限的覆盖树, 通过在覆盖树上找到满足或者违反某种需要验证的性质的状态来判断原系统是否满足该性质.
- 第二, 关联矩阵与状态方程法(incidence matrix and state equation)[118]也常用于 Petri 网的性质验证. 该方法利用矩阵方程来表示 Petri 网的动态行为. 这种方法的基础是关联矩阵. 它定义了 Petri 网中所有可能的位置和转换之间的互连. 通过解关联矩阵的方程来进行性质的验证.

目前, 基于 Petri 网的验证和分析工具主要有 Travis[119]和 PNR[120]等, 在性能评价[121]、通信协议分析[122]、分布式软件系统[120]中具有明显的优势.

3.4 其他验证工具

3.4.1 PAT

PAT (process analysis toolkit)[123]是一个用于并发系统、实时系统与概率系统的仿真与验证工具. 它最初的版本主要是验证用 LTL 描述的系统的公平性[124]. 后来, PAT 逐渐发展成能够验证多领域系统的验证工具[35]. 它定义了 11 个模块, 每个模块对应不同的领域语言, 面向分布式系统、安全协议、实时系统等. 每个领域语言都有不同的语法. 例如, CSP 模块用进程代数定义, 实时系统是用时间自动机定义, 等等. PAT 的处理方法是对这些不同模块定义统一的语义模型, 即使用最广泛的迁移系统来表示, 分别有标号迁移系统、实时迁移系统、马尔可夫决策过程, 以支持不同的模型. 同时, PAT3 实现了多个模型检验算法, 用于验证不同的系统和性质, 比如用时序逻辑 LTL、CTL、CTL*表示的性质、公平性质、无发散、无死锁、精化关系检查以及概率模型性质等. 针对不同的领域模型, PAT 设计了友好的用户交互界面与相应的编辑器, 将模型自动转换到统一语义模型再进行分析与验证. 为了获取良好的性能, PAT 在自身内部运用了很多优化技术, 如进程计数器抽象、并行进行模型检验等等. 作者分别将 PAT 与 SPIN、UPPAAL 以及 PRISM 进行了比较[125].

对于 CSP 模块, PAT 扩展了以上介绍的标准 CSP 语法, 加入共享变量、赋值以及选择、循环等控制流结构, 语言描述更加丰富. CSP 的语义被定义成一个标号迁移系统. PAT 支持 CSP 模型之间的精化关系检查, 因此可以用于 CSP 的验证.

PAT 在实际中具有广泛的应用. 2PC (two phase commit)协议是许多分布式关系型数据库用来完成分布式事务的重要协议. 该协议包含协调者和参与者两个进程, 对保证数据的一致性具有重要作用. PAT 通过对该协

议的两个阶段进行建模, 可以验证 2PC 协议的无死锁性质, 并可以根据需要定义相关 LTL 性质来进行验证. 此外, Fernando 等人[126,127]通过 PAT 验证了概率 BAR 系统的纳什均衡(Nash-equilibrium)性质, Thin 等人[128]利用 PAT 对区块链进行了形式化分析. 此外, PAT 也被用于带有时间属性的通信协议的验证[129]以及概率系统中概率相关性质的验证[130].

4 实时系统

实时系统是指对于时间有严格要求的系统, 系统的状态、动作或运行轨迹需要满足一定的时间约束条件, 例如, 必须在一定时间期限内做出响应等. 汽车的控制系统、网络实时监控系统都有这些要求, 因此可以称为实时系统. 下面将从实时系统的行为规约、性质规约和验证、验证工具进行展开, 最后还对实时系统的鲁棒性进行介绍.

4.1 行为规约

时间自动机(timed automata, TA)是一个常用的实时系统建模语言, 这一模型最初由 Alur 等人提出[131,132]. TA 可以看作在有穷状态自动机的基础上引入了有限个时钟变元, 用字母 x, y, z 表示. 它们的取值范围是 $R_{\geqslant 0}$, 在进入初始状态时, 所有时钟变元都被赋值为 0, 之后具有相同的增长速率. TA 还包含“时钟约束条件”和“时钟变元的重置”: 一个时钟约束条件是若干形如 $x\sim c$ 的原子公式的合取, 其中, c 是常数, $\sim\in\{<,>,\leqslant,\geqslant\}$, 时钟约束条件可以标注在迁移上, 表示迁移发生的必要条件, 也可以标注在状态上, 表示停留在该状态的必要条件; 时钟变元 x 的重置只能标注在迁移上, 表示进入下一个状态时 x 重置为 0. 此外, 标注在迁移上的字母表示动作.

图 2 是一个时间自动机的样例, 描述了包含两种亮度的声控灯的行为: 初始状态为关闭(off), 第 1 次触发后进入暗状态(dim), 如果没有再次触发, 则在暗状态持续 10 s 后关闭; 如果在暗状态再次触发, 则会进入亮状态(bright), 在亮状态持续 10 s 后自动关闭. 以从 dim 到 bright 的迁移为例, 在处于 dim 状态时, 首先要满足 dim 状态的条件 $x\leqslant 10$, 迁移上的标注 $b,x>1,\{x\}$表示该迁移对应 b 动作, 发生的必要条件是 $x>1$, 并且如果发生了迁移, 则把时钟变元 x 重置为 0, 之后才进入 bright 状态.

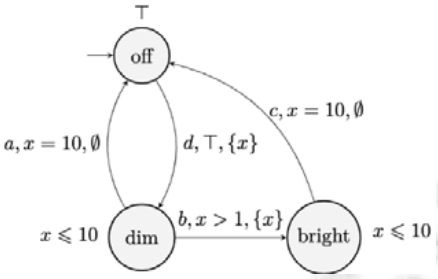


图 2 两档声控灯时间自动机

我们称 TA 的一个格局(configuration)是由当前位置 s 与时钟变元赋值 v 构成的二元组(s,v). TA 允许两种类型的格局变化: “迁移”和“停留”. 迁移发生的必要条件是满足该迁移对应的时钟约束条件, 而在迁移的过程中, 允许重置部分时钟变元; 若想在自动机的某一状态停留, 同样需要满足该状态对应的时钟约束条件. TA 的一次执行可以看成由有限或无限个格局构成的序列, 如 σ=(off,x=0) (off,x=2) (dim,x=0) (dim,x=2) (bright,x=0) (bright,x=2) (off,x=2) (off,x=4) (dim,x=0)…就是一次执行, 描述的是依次在 off、dim 和 bright 停留 2 s 后转移到下一状态的无限循环.

一个带时间戳的动作是二元组(t,a), t 是自开始运行经历的时间, a 是在 t 时刻执行的动作. 对于 TA 的一次执行, 两个相邻的格局之间的转移可由带时间戳的动作描述, 所以由一次执行可以生成一个动作序列, 如上一段中的执行σ对应的动作序列是$(2,d)(4,b)(6,c)(8,d)$… 我们称所有这样的动作序列为该 TA 接受的语言.

4.2 性质规约和验证

对于 TA 本身和 TA 之间的关系, 我们关注 3 类性质: 可达性、包含性和互模拟性. 其中, 可达性判定的复杂度是 PSPACE 的, 借助于 region 等价类划分的概念, 将 TA 的无限状态空间划分成有穷个区域(region), 从而将 TA 的可达性转化为有穷迁移系统上的可达性[132]. 包含性和互模拟性考虑 2 个或多个 TA 之间的关系: 两个 TA 之间满足包含关系说明一个 TA 接受的语言也被另一个接受; 互摸拟则说明两个 TA 之间满足某种等价关系. 其中, 包含性是不可判定的, 互模拟性是可判定的[133].

在文献[134]中, 作者提出了几类具体的实时性质的模式, 例如, 一个系统在多长时间内一直满足某个性质(minimum and maximum duration); 一个性质多长时间满足一次(periodic 或 recurrence); 一个状态出现之后, 多长时间内另一个状态出现(bounded response). 用于刻画这些 TA 性质的规约语言有多种, 其中比较重要的是基于分支时间的 Timed CTL (TCTL)和基于线性时间的 Timed Propositional Temporal Logic (TPTL)[135].

和 CTL 一样, TCTL 语法的定义分为状态公式ϕ和路径公式φ两部分[136,137]:

$$\phi:=\text{true}|a|g|\phi_1\wedge\phi_2|\neg\phi|\exists\varphi|\forall\varphi,$$

$$\varphi:=\phi_1\mathrm{U}^J\phi_2,$$

其中, a 表示原子命题, g 表示时钟约束条件. TCTL 在 CTL 的基础上额外允许状态公式ϕ可以是时钟约束条件 g. 路径公式只允许$\varphi:=\phi_1\mathrm{U}^J\phi_2$一种形式, 其中, J 是 $R_{\geqslant 0}$上的区间, 表示存在一个 J 中的时刻 t 满足ϕ_2, 并且在此之前一直满足ϕ_1. 例如, $legal\mathrm{U}^{[0,30]}deadlock$ 是一个 TCTL 公式, 表示正常运行 30 s 内会死锁. 给定 TA 和一个 TCTL 公式θ, 验证 TA 是否满足θ可以归结为迁移系统上 CTL 性质的验证, 难点在于如何构建对应的迁移系统 TS 与 CTL 性质ϕ, 使得 TA 满足θ当且仅当 TS 满足ϕ. TCTL 模型检验算法首先通过引入新的时钟变元将 TCTL 公式θ转化为等价的 CTL 公式ϕ, 再借助 region 等价类划分的方法将 TA 转化为有限大小的迁移系统 TS, 从而得到了一个等价的 CTL 验证问题. TCTL 模型检测算法的复杂度是 PSPACE 完全的[136].

TPTL 语法定义如下:

$$\varphi:=a|\pi_1\leqslant\pi_2|\pi_1\equiv_d\pi_2|\text{false}|\varphi_1\rightarrow\varphi_2|\circ\varphi|\varphi_1\mathrm{U}\varphi_2|x.\varphi,$$

其中, $\pi=x|x+c$, c 为常数. TPTL 在 LTL 的基础上允许公式中出现带时钟变元的表达式比较以及同余运算和时钟变元的初始化. $x.\varphi$表示将 x 赋值为 0 后φ成立, 时态算子□, ◊也可以按 LTL 的方式定义. 例如, $\Box x.(p\rightarrow\Diamond y.(q\wedge y\leqslant x+10))$是一个 TPTL 公式, 表示每一个 p 请求能够在 10 s 内得到 q 响应. 可以证明: TPTL 的表达能力严格强于本文第 6 节将要介绍的 MTL[138], 且 TPTL 公式的永真性判定问题是 EXPSPACE 完全的[139].

4.3 验证工具

目前已有多种用于时间自动机验证的工具, 如 Uppaal、KRONOS、RED 等.

Uppaal 是目前最常用的基于 TA 的实时系统建模、仿真、验证工具, 目前仍在维护[140]. Uppaal 建模使用的模型是 TA 网络, 即多个并发执行、相互可以通信的 TA, 支持共享整数变元、紧急通道、提交状态等特殊机制以刻画不同需求的系统. 在性质验证方面, Uppaal 定义了一个可验证的 TCTL 安全子集, 并支持验证一些限界活性性质[141]. Uppaal 集成了多种分析和验证工具, 在工业界的应用场景丰富. 验证方面, 应用于音频控制协议、传输协议、通信协议的验证, 成果包括在 Bang&Olufsen 公司的一个音频设备控制器中发现了 3 个设计漏洞[142]; 测试方面, 集成了 TRON 在线测试工具以及 COVER 测试生成工具, 能够基于 TA 模型生成满足特定可达性性质的测试样例, 成果包括基于测试建模冰箱控制系统[143]; 综合方面, 集成了 STRATEGO、CORA, 基于 PricedTA 分析综合最优策略和最优调度算法, 成果包括卫星电池的控制系统、房屋温控系统以及交通信号灯控制器的综合[144].

KRONOS 的特点在于支持多种形式的建模, 并且除了基本的 TCTL 性质以外, 还允许通过定义一个 TA 从行为的方式描述性质, 将特定性质的验证归结为忽略时钟变元后两个自动机的互模拟问题[145]. KRONOS 主要用于多种通信协议的验证, 包括 ATM 协议、CSMA-CD 协议和 FDDI 协议等; 除了协议的验证以外, 也曾应用于芯片验证和实时调度器的综合[146]. RED 主要用于验证带同步操作的分布式实时系统, 支持基于时间自动

机的仿真与模型检验, 采用基于 CRD (clock-restriction diagram)的模型检验算法[147].

4.4 鲁棒性

鲁棒性是实时系统的重要性质. 在实时系统中, 时间扮演着重要的角色, 而时间的测量、时钟的实现以及控制导致的误差会干扰系统的正常运行. 对于实时系统而言, 系统执行过程可以看成一个带时间戳的执行序列. 我们可以在所有执行构成的空间中定义一个度量, 借助度量空间的相关理论, 研究实时系统的鲁棒性[148]. TA 鲁棒性研究主要应用于设计与实现的过程中, 给定一个 TA 模型 A, 我们用 A_δ 表示把 A 中所有时钟约束条件都松弛 δ 后得到的模型(如 $x>c$ 松弛成 $x>c-\delta$), 则理论模型 A 的实现可以看成 A_δ. 如果希望实现的模型鲁棒满足性质 ϕ, 则需要验证松弛后的模型 A_δ 满足 ϕ[149].

5 概率随机系统

概率系统指行为具有随机性的系统. 随机性可能来自于随机算法, 也可能来自于真实环境下系统组件失效或外界干扰等. 此外, 也可以选择将系统描述的一些不确定性使用随机性刻画. Katoen 为概率模型检验的现状做了较为完整的综述[150].

5.1 行为模型

概率系统的行为模型可以从多个维度分类: 离散时间/连续时间、确定性/非确定性、是否涉及奖励等. 由此可以延伸出多种概率模型. 下面我们介绍 3 种主要模型: 离散时间马尔可夫链、马尔可夫决策过程以及连续时间马尔可夫链.

离散时间马尔可夫链(discrete-time Markov chain, DTMC)是最基础的概率系统模型[151,152]. DTMC 是一类迁移系统, 在每个状态下通过一个迁移概率分布选择后继状态, 从而对于每一条有穷执行路径, 都可以计算出该执行发生的概率. DTMC 也有其他变种, 如马尔可夫奖励模型(Markov reward model, MRM)和马尔可夫决策过程(Markov decision processes, MDP)[153,154]. MRM 在 DTMC 的基础上给每个状态附加了一个表示奖励(或费用)的值, 从当前状态迁移到后继状态时, 当前状态对应的值会累加, 从而可以定义一条路径的奖励(或费用). 在这个模型中, 主要考虑计算奖励的期望或带有约束条件的事件发生的概率. MDP 在 DTMC 的基础上引入了非确定性决策, 不同于 DTMC 中每一个状态都对应着唯一的后继状态迁移概率分布, MDP 中每一个状态可能有多个概率分布, 需要先非确定性地选择一个概率分布, 然后依照这个概率分布选择后继状态. 非确定性选择可以由一个策略(scheduler)表示, 当策略提前给定时, MDP 可以转化为等价的 DTMC.

图 3 是一个 DTMC 的例子, 描述了 IPv4 中 Zeroconf 协议的控制过程. 在配置 IP 地址时, 主机每次需要随机选择一个地址, 并广播询问该地址是否被使用了, 广播最多重复 n 次. 在该样例中, n=4. 我们假设有 q 的概率选择的地址是已经被使用的, 并且每次广播后未收到回复的概率是 p.

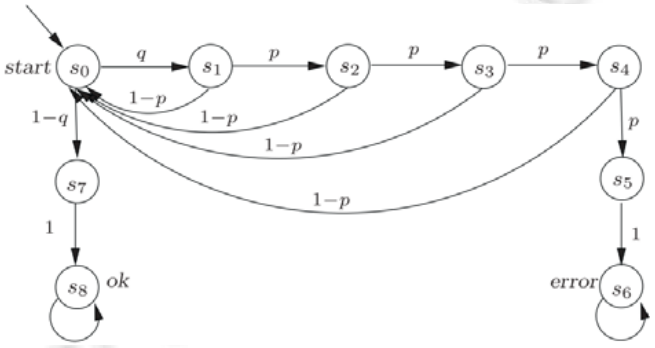


图 3 IPv4 协议对应的离散时间马尔可夫链(n=4 情况)[4]

与 DTMC 对应, 刻画连续时间概率系统的基础模型是连续时间马尔可夫链(continuous-time Markov chain,

CTMC), 而对连续时间下概率行为更早期的研究可以追溯到泊松过程[155]. CTMC 在迁移系统的基础上引入了迁移速率矩阵, 记矩阵第 i 行第 j 列的元素为λ_{ij}, 表示从状态 i 到状态 j 的迁移发生的概率满足以λ_{ij}为参数的指数分布, 当有多个可能的后继状态时, 选择最先触发的迁移. 另一方面, 根据指数分布的性质, 易知"离开状态 i"事件发生的概率服从以 $\sum_{j=1}^{n}\lambda_{ij}$ 为参数的指数分布(假设一共有 n 个状态), 且迁移到状态 j 的概率是 $\lambda_{ij}\Big/\sum_{j=1}^{n}\lambda_{ij}$, 可以把迁移过程看成先根据一个指数分布确定迁移发生的时间, 再仿照 DTMC 根据概率进行迁移. 图 4 是一个 CTMC 的例子, 有一个待执行任务的队列, 初始为空, 最多储存 3 个任务. 任务进入的速率是 3/2 (平均到达时间间隔为 2/3), 任务完成的速率是 3 (平均完成的时间是 1/3).

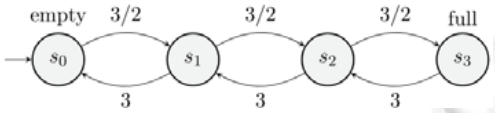


图 4 工作任务队列对应的连续时间马尔可夫链

5.2 性质规约

DTMC 中的线性时间性质可以分为定性性质(qualitative properties)与定量性质(quantitative properties), 使用 LTL 公式描述. 在 DTMC 的线性时间性质验证中, 4 种基础的 LTL 定性性质(可达性、约束可达性、重复性和持续性)可以通过忽略 DTMC 中迁移概率的具体数值, 仅区分迁移概率大于 0 (存在一条边)和等于 0 (不存在边), 并分析图的结构验证[156]. 对于状态数有限的 DTMC, LTL 定性性质验证问题是 PSPACE 完全的[157]. 定量线性时间性质则是从某一状态出发, 所有满足特定线性时间性质的路径概率之和. 当 LTL 性质为正则性质时, 可以通过构造该性质的补所对应的有限状态自动机以及原马尔可夫链与该自动机的乘积, 将验证问题转化为马尔可夫链的可达性问题. 当 LTL 性质为 ω-正则性质时, 可以通过构造确定性 Rabin 自动机, 转化为马尔可夫链的可达性问题[158].

DTMC 和 CTMC 的分支时间性质可以分别由概率计算树逻辑(probabilistic CTL, PCTL)和连续随机逻辑(continuous stochastic logic, CSL)描述, 这两种逻辑都是在 CTL 中引入概率.

PCTL 的语法定义如下, 同样分为状态公式ϕ和路径公式φ两部分:

$$\phi:=\text{true}|a|\phi_1\wedge\phi_2|\neg\phi|P_J(\varphi);$$

$$\varphi:=\circ\phi|\phi_1\text{U}\phi_2|\phi_1\text{U}^{\leqslant n}\phi_2.$$

PCTL 移除了 CTL 状态公式中的全称量词和特称量词, 补充了概率算子 P_J, 其中, J 是 $R_{\geqslant 0}$ 上的区间, $s\vDash P_J(\varphi)$表示从 s 状态出发的所有满足φ的路径概率之和在 J 区间内[159]. 在路径公式中, 引入了带界限的 until 算子 U_I. 时态算子□、◊以及带界限的$\Diamond^{\leqslant n}$可以仿照 LTL 中的方式定义.

例如, $P_{=1}(\square(try_to_send\rightarrow P_{\geqslant 0.99}(\Diamond^{\leqslant n}delivered)))$表示几乎一定每次尝试发送信息后有至少 0.99 的概率会在 3 步内完成发送. PCTL 性质的验证可以仿照迁移系统中 CTL 公式的验证, 由公式的语法树自底向上分析, 需要额外处理算子 P_J 和 U_J 两种情况. 如果要求 PCTL 中的 $P_J(\varphi)$要么是 $P_{>0}(\varphi)$, 要么是 $P_{=1}(\varphi)$, 即只能描述事件几乎一定发生或几乎一定不发生, 则得到了一个描述定性性质的子集 QPCTL, 其表达能力与 CTL 是不可比较的. PCTL*与 PCTL 的关系类似于 CTL*与 CTL 的关系, PCL*在 PCTL 的基础上允许路径公式是任意形式的 LTL 公式, PCTL*表达能力严格强于 PCTL 或 LTL. PCTL/PCTL*与多个 DTMC 之间模拟/互模拟关系有密切联系, 例如, 两个互模拟等价(bisimulation equivalence)的状态满足相同的 PCTL*公式[160,161].

CSL 的语法定义如下, 同样分为状态公式ϕ和路径公式φ:

$$\phi:=\text{true}|a|\phi_1\wedge\phi_2|\neg\phi|P_J(\varphi)|S_J(\varphi);$$

$$\varphi:=\circ\phi|\phi_1\text{U}^{\leqslant n}\phi_2.$$

CSL 在 CTL 的基础上引入了概率算子 P 和稳态算子(steady state operator)S. $P_J(\varphi)$表示在某一时刻满足φ性质的所有路径概率之和属于 J 区间, 即瞬态性质; $S_J(\varphi)$则表示在长期稳态情况下, 满足φ的所有路径的概率之和属于 J 区间, 即稳态性质. 例如, $S_{<0.1}$(full)表示在长时间运行时, 任选一个时刻观察所处状态, 处于 full 的概率小

于 0.1. CSL 同样与 CTMC 之间的互模拟关系有联系[162]. CSL 公式的验证同样按照语法树自底向上分析, 对瞬态性质的验证要借助均匀化(uniformisation)方法, 稳态性质的验证归结为线性方程组的求解[163].

5.3 安全-活性分类

Katoen 等人将安全-活性的分类方式拓展到概率模型中[164]. 他们提出了概率树的概念以表示概率意义下的系统执行行为, 并定义了概率树的前缀和延伸操作. 性质可以表示为概率树的集合, 一个性质 E 是安全性质当且仅当任何违反 E 的概率树 T 必然有一个有限深度的前缀 T', 使其不能延伸到任何满足 E 的概率树(也就是说, 执行到 T'时错误已经不可挽回了). 一个性质是活性当且仅当任何有限深度的概率树都可以延伸为某个 E 中的概率树. 在概率树的意义下, 同样可以证明, 任何一个概率树性质都可以表示成一个安全性质和一个活性性质的合取.

5.4 验证工具

PRISM 模型检验工具支持对概率系统的建模、仿真和验证[165]. 该工具支持不同形式的概率系统建模, 除了前面介绍的 DTMC、MRM、MDP 和 CTMC 以外, 还包括概率自动机(PA)、部分可观测马尔可夫决策过程(POMDP)等. 建模使用 PRISM 定义的基于状态的语言, 通过声明状态和迁移规则定义模块, 支持模块的复用. PRISM 能够验证由 PCTL、CSL、LTL 和 PCTL*规约语言描述的性质, 并使用了高效的数据结构和符号化方法优化模型检验算法[166]. 目前, PRISM 已经被用于通信协议验证、分布式算法、生物系统等不同领域中概率系统的分析和验证, 成果包括如 CSMA-CA 协议的正确性验证[167]、计算自稳定协议中的各种概率问题[168]、验证生物计算模型 DNA-walker 的可靠性[169]等一系列工作.

STORM 是 2017 年发布的一款功能强大的概率模型检验工具[170], 支持 DTMC、CTMC、MDP 以及连续时间 MDP 的建模, 性质规约语言支持 PCTL 和 CSL, 但不支持 LTL. 其主要特点是允许多种输入形式, 除了支持 PRISM 输入语言外, 还支持广义随机 Petri 网、动态故障树等形式, 并采用了更高效的建模和验证算法.

EPMC (原 iscasMC)是由中国科学院软件研究所开发的一款基于网页的模型检验工具[171], 具有模块化的特点, 能够将建模和验证的计算过程分离. EPMC 主要目标是验证复杂的 LTL 性质, 对于这类性质的验证实现了相应的启发式算法, 具有较高的效率.

类似的验证工具还有 MRMC[172]、LiQuor[173]等, 它们针对的模型、性质以及使用的技术都略有区别. 如 MRMC 只支持 DTMC 和 CTMC 以及带权重的形式, 使用了基于稀疏矩阵的表示方式以及概率互模拟. MRMC 一个重要的应用是针对体系分析与设计语言 AADL (architecture analysis and design language)的验证. AADL 是一个系统级别的建模语言, 同时描述硬件系统、软件系统以及组合系统, 在航天器、卫星控制系统以及其他安全攸关系统的建模中扮演重要角色. 目前, MRMC 是 AADL 的验证工具平台 COMPASS 的组成部分, 成果包括对于一个包含 90 个控制组件的卫星平台控制系统的安全性验证[174,175]. LiQuor 主要针对 MDP 上的 ω-正则性质的定性、定量验证, 使用了偏序归约(partial order reduction)技术避免搜索空间爆炸.

5.5 概率程序和概率霍尔逻辑

以上工作都是从自动机的角度出发描述系统的概率行为, 使用模型检验方法验证系统的概率性质. 近年来, 在程序模型中加入概率, 并以概率霍尔逻辑或最弱前期望(weakest pre-expectation)语义为基础验证概率性质, 是另一种基于逻辑推理的形式化验证方法. 目前主要集中于离散程序, 在命令式程序中加入概率选择或概率赋值. 例如, $P_1[p]P_2$ 表示程序以 p 的概率执行 P_1, 以 $1-p$ 的概率执行 P_2; 概率程序还可以表示 x 赋值为一个遵从 D 分布的量. 对于概率程序, 可以使用最弱前期望描述一个概率程序在终止时某个表达式的期望值或某个性质成立的期望值. 这方面的难点包括带概率程序的终止性, 以及带概率循环程序和递归程序的验证问题, Katoen 等人在这方面做了很多的工作[176−178].

5.6 可靠性和可用性

概率性质验证的一个主要应用方向是可靠性和可用性验证: 系统的可靠性(reliability)指的是在给定条件、

给定时间区间下, 系统能够无失效地执行要求的能力; 而可用性(availability)指的是系统处于按要求执行状态的能力. 在一个系统中, 可靠性和可用性是密切相关的. 从概率的角度来看, 系统可靠性指的是在给定的时间区间 t 内, 系统能够不失效执行的概率; 可用性指的是整个系统在某个时间区间内处于正常运行的比例.

在本节内容中, 我们将介绍系统可靠性和可用性常用的方法——故障树(fault tree, FT)分析以及可靠性框图(reliability block diagram, RBD), 并对相关的可靠性可用性验证工具进行介绍.

5.6.1 故障树分析

标准故障树(也称静态故障树)是最基本的故障树, 由 Bell 实验室在 20 世纪 60 年代提出, 并用于分析导弹[179]. 故障树是一个将系统抽象得到的树形结构(更一般地, 可以看作是有向无环图)[180]. 故障树由事件和门两部分组成.

- 事件分成基本事件和中间事件(包括未展开事件、条件事件等等): 基本事件为在系统中要分析的基本元组件, 在故障树模型中用一个圆圈表示; 出现在故障树最顶部的事件称为顶事件, 一般为所分析的系统的需求事件, 用矩形表示. 例如, 研究一台计算机是否正常运行时, 基本事件包含这台计算机的 CPU、内存、磁盘等等, 顶事件就是这台计算机没有正常运行, 非基本事件包含这台计算机是否在使用等.
- 故障树中所使用的门和数字逻辑中的门电路类似, 包含了与门、或门、表决门(在 N 个输入中, 有超过 k 个为 1 则输出为 1)、禁止门(输入为 1 且条件事件成立则输出为 1)等. 通过门结构与事件的组合, 我们可以将系统建模成一个树形结构.

图 5 展示一个基本的故障树, $E1$–$E4$ 表示 4 个基本事件, 4 个基本事件接入两个或门, 再连入一个与门, 最上层顶事件表示系统失败. 在该系统中, 只要 $E1$ 或 $E2$ 其中一个组件出故障, 且 $E3$ 或 $E4$ 其中一个组件出故障, 就会出现整个系统的故障.

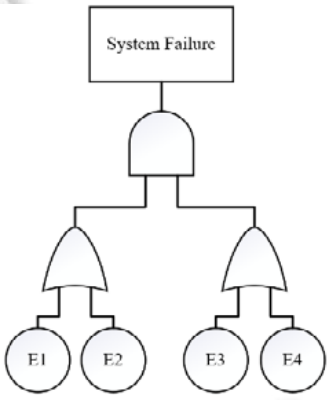


图 5 一个基本的故障树示例

在 Codetta-Raiteri 等人的研究[181]中, 将故障树形式化地表示成一个四元组 $F=\langle BE,G,T,I\rangle$. 其中, BE 表示基本事件集合; G 为门集合; T 为门到门类型的映射函数, 将每一个 G 中的元素映射到对应的门类型中; I 为门的输入大小映射, $I(g)$表示门 g 的输入大小.

接着, 可以在树结构中进行性质的分析. 从定性性质上, 可以找到树中的最小割集(minimal cut sets, MCS)和最小路径集(minimal path sets, MPS), 从而确定整个系统正常运行的条件. 最小割集是一个系统的最少组件集合, 这个集合中的组件全部出故障, 则整个系统出故障. 上图中的$\{E1,E3\}$就是一个最小割集. 对应地, 最小路径集和最小割集是相反的, 即一个系统的最少组件集合, 这个集合中的组件全部正常工作, 整个系统将会正常工作, 上图中的一个最小路径集为$\{E1,E2\}$. 在一个故障树中, 最小割集和最小路径集都不一定是唯一的.

给定一个故障树, 经典的求最小割集和最小路径集的算法是自顶向下和自底向上的方法[180]. 此外, 文

献[182–184]使用了一种更高效的算法——二分决策图(binary decision diagram, BDD)来寻找一个故障树的最小割集和最小路径集. Monte Carlo 算法也常用于最小割集和最小路径集的求解.

从定量性质上, 首先考虑单一时刻, 一个系统的可靠性是系统在该时刻下不出现错误的概率[185]. 令 X_F=1 表示系统 F 出现错误这一事件, 那么在某一时刻下, 系统的可靠性 $Re(F)=\Pr(X_F=0)$. 因此, 当我们知道系统在某一时间下每一个基本事件故障的概率时, 就能通过对故障树自底向上的分析得到整个系统的可靠性. 当计算连续一段时间内系统的可靠性时, 只需知道每一个基本事件在这段时间内故障所满足的分布. 这样, 定义 $Y_F=\max\{t|\forall_{s<t}X_F(s)=0\}$, 系统的可靠性 $Re(F)=\Pr(Y_F>t)$. 对应上面可用性的定义, 系统的可用性 $A_F(t)=E(X_F(t))$. 在以往的工作中, 研究者通常会用 BDD、Monte Carlo[186]、贝叶斯网络分析(Bayesian network analysis)[187]等方法计算系统的可靠性和可用性.

标准故障树还能进行一系列的扩展, 例如, 使用动态故障树(dynamic fault tree, DFT)来处理包含序列的组件[188]. 在实际情况中, 组件的失效按照某种特定的序列出现时, 整个系统不会实效, 而组件的失效以其他顺序出现时, 系统又会出现失效情况. 动态故障树就是对基本故障树进行了序列依赖上的扩展. 动态故障树在标准故障树的基础上加入了一些考虑优先级的门结构, 通过添加这些组件, 动态故障树相对于标准故障树具有更强的表达能力. 这里, 我们将介绍几个 DFT 中重要的门结构.

- PAND 门: 当输入按顺序从左到右依次为 1 时, 门的输出才为 1. 例如, 一个设备有一个备份, 只有当备用触发开关激活后, 若此备份失效, 这部分组件才会失效. 我们用 PAND 门表示了这种组件序列依赖的关系.
- FDEP(function dependency)门: FDEP 的输出是一个伪输出(即恒等于 0), FDEP 门有一个触发器, 当触发器触发时, FDEP 门会使所有的门输入进行激活(即输入等于 1), 当然, 其输入不一定需要触发器进行触发. 例如, 电源的失效可以导致所有 CPU 的失效, 我们可以把电源看作一个触发器, 用 FDEP 门来表示电源与 CPU 之间的关系.
- SPARE 门: 表示可由一个徽章多个备件替换的组件. 当主设备发生故障时, 从左到右激活第 1 个可用的备件, 一个备件可以连接到多个 SPARE 门, 但只能被一个 SPARE 门使用. 例如, 一个计算机系统中可以有内存备份, 当一个内存失效时, 可以用备份内存替换进行工作.

图 6 是一个动态故障树的示例.

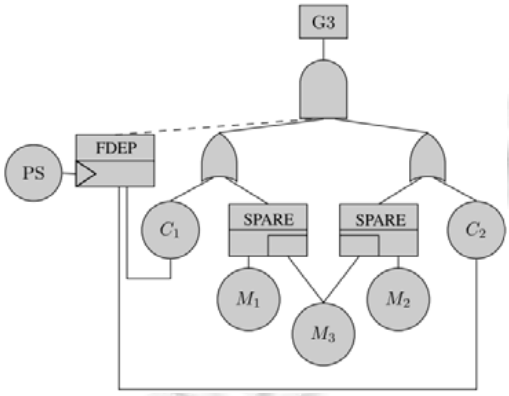


图 6 动态故障树的实例[189]

这个故障树实例加入了两个动态故障树的门结构 FDEP 门和 SPARE 门, 对于 FDEP 门而言, 当 PS 触发时, C_1 和 C_2 值为 1, $G3$ 的值为 1(即系统失效). 当 PS 不触发时, C_1 和 C_2 也会根据具体系统的情况取值为 1 或 0. 我们考虑 SPARE 门, 当 M_1 和 M_2 中只有一个失效时, 备件 M_3 能够对失效的组件进行替换; 而当 M_1 和 M_2 都失效时, 备件 M_3 不能同时对两个门进行替换, 2 个 SPARE 门中有 1 个会失效.

此外, 研究者还考虑了一些其他方面的拓展. 例如, 由于实际的故障分布难以获取, 研究者考虑用模糊数

代替组件的故障分布[190]; 文献[181,191]中考虑在故障树中加入故障修复的情况, 使得一些组件的故障以一定的概率分布进行修复.

有很多基于故障树的工具用于进行系统的可靠性和可用性分析. 例如, CORAL[192]借助 Markov 链对故障树进行分析; DFT2GSPN[193]通过引入 Petri 网进行计算, 其效率比基于 Markov 链的工具有所提升; DFTSim[194]以及 DRSIM[186]基于 Monte Carlo 方法进行分析, 并支持了多种故障树的拓展.

5.6.2 可靠性框图

可靠性框图[195]也是一个常用于评估各种系统可靠性和可用性的分析方法, 其基本思想和故障树类似, 将一个系统分成各个组件, 再将各个组件通过并联、串联等方式组成一个无向图模型. 因此, 在一个图结构中, 可以根据单个组件的故障概率分析来计算整个系统的可靠性和可用性. 对于各种不同的结构, 有对应不同的公式.

同样地, 类似于标准故障树到动态故障树的拓展, RBD 可以考虑动态依赖因素, 拓展成为 DRBD. Distefano 等人[196]通过在 RBD 中引入动态依赖模块, 对系统组件之间的动态依赖关系进行建模, 从而扩展了 RBD 的表达能力.

基于 RBD 和 DRBD 也存在一些形式化验证的工具. 例如, Object-Z[197]通过将 DRBDs 转化成一个带颜色的 Petri 网(colored Petri net, CPN)中, 接着利用 Petri 网的分析工具来完成对 RBD 的分析; 文献[198,199]利用 HOL4 高阶逻辑验证工具[200]对 RBD 以及 DRBD 进行了形式化的建模与分析, 从而完成对系统可靠性和可用性的评估.

6 混成系统

动力系统指的是随着时间连续演化的系统, 其连续行为通常使用微分方程等数学模型来描述. 混成系统是既包含离散行为也包含连续行为的系统, 典型的例子是受离散程序控制的动力系统: 每个时间周期(或每个事件发生时)离散程序执行一次, 改变系统的离散状态或参数; 然后, 系统以微分方程刻画的规律随时间演化. 混成系统广泛存在于航空航天、自动驾驶等安全攸关领域, 这些系统的失效会带来人员伤亡等灾难性后果. 因此, 对这些系统的安全设计及验证变得极为重要. 基于严格数学基础的形式化方法是确保这些系统安全的重要支撑.

6.1 行为规约

混成系统可以使用多种形式化建模语言表达, 其中, 由 Henzinger 提出的混成自动机模型[201]是常用建模语言之一. 混成自动机可以看作有限状态自动机和时间自动机的扩展. 当系统停留在每个位置上时, 系统的变量沿着给定的微分方程演化. 每个位置可以带有不动条件: 只要系统状态满足不动条件, 系统就可以继续停留在该位置. 另外, 每条迁移可以带有条件: 仅当系统状态满足条件, 迁移才可以执行. 每条迁移也可以修改系统中的状态变量. 例如, 图 7 展示了混成自动机的一个实例. 模型代表一个保温系统, 该系统具有两个状态: ON(开)和 OFF(关). 假设初始温度为 23 °C, 系统进入 OFF 状态, 温度 T 以 $-0.1T$ 的速率变化, 当温度满足小于 22 °C 时, 可跳转至 ON 状态, 也可继续降温直至温度不再满足 $T \geqslant 20$ °C 时跳转到 ON 状态. 关于 ON 状态的解释类似.

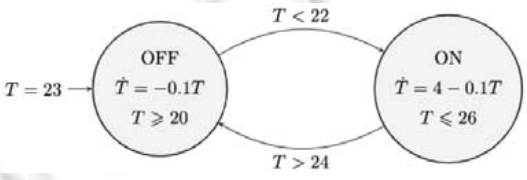


图 7 混成自动机的实例

混成自动机的优点是语言相对简单. 然而和有限状态自动机一样, 混成自动机的可组合性较弱, 对于复

杂的系统, 很难体现系统的模块化结构. 除了混成自动机以外, 其他混成系统建模语言包括微分动态逻辑(differential dynamic logic, dL)[202]和混成通信顺序进程(hybrid communicating sequential process, HCSP)[203,204]. 微分动态逻辑由Platzer提出, 是动态逻辑(dynamic logic)的一个扩展, 在动态逻辑的基础上加入了沿着给定微分方程演化的语句. HCSP是CSP的一个扩展, 在CSP的基础上添加了微分方程演化和中断语句, 表示系统沿着微分方程运行的过程中同时等待给定的通信. 当其中一个通信可以发生时, 中断连续演化并执行通信.

除了形式化模型之外, 工业界普遍使用的嵌入式系统建模语言可用于描述混成系统, 例如Simulink[205]、Modelica[206]等. Simulink采用图形建模语言, 通过将简单的块(block)连接起来对混成系统建模. 这种图形建模语言更易于工程师理解和使用.

Modelica则定义了一个能够表达微分方程和微分代数方程(differential algebraic equation)的程序语言.

6.2 性质规约

对于混成系统, 我们最关注的性质依然是安全性, 也就是系统不会进入危险状态, 例如自动驾驶的车辆不会与其他车辆相撞. 除了安全性外, 我们也可以考虑活性性质, 例如自动驾驶的车辆最终能够到达目的地.

混成系统的任何时序性质可以使用LTL的扩展MTL和STL表达. MTL (metric temporal logic)[207]是LTL向连续系统的第1步扩展, 应用于连续时间而不是离散时间的运行轨迹. 假设系统状态由多个布尔值组成, 系统的运行轨迹可以表达为时间(正实数)到状态的函数. MTL的语法是:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathrm{U}_{[a,b]} \varphi_2,$$

其中, 主要添加的语法构造是$\varphi_1 \mathrm{U}_{[a,b]} \varphi_2$, 含义为: 存在一个时间点$t \in [a,b]$, 系统的运行轨迹在时间$0–t$之间满足性质$\varphi_1$, 并且在时间$t$满足性质$\varphi_2$. 在$\mathrm{U}_{[a,b]}$的基础上, 我们可以定义$\Box_{[a,b]}\varphi$和$\Diamond_{[a,b]}\varphi$, 分别代表"在$[a,b]$区间内总是满足$\varphi$"和"在$[a,b]$区间内存在一个时间满足$\varphi$". 例如, $\Box_{[0,10]}(p_1 \rightarrow \Diamond_{[2,4]} p_2)$的含义为: 在[0,10]区间内的任何时间, 如果性质p_1被满足, 则在那个时间之后的[2,4]时间区间内, 存在一个时间使得p_2被满足. MTL性质的模型检验是不可判定的, 主要问题来自形如$[a,a]$的约束, 要求对时间有精确的计算. 因此, Alur等人提出了MTL的变种MITL (metric interval temporal logic)[208], 在时间约束上仅允许$a<b$的情况, 并证明了因此得到的模型检验问题在EXPSPACE中的可判定性.

STL (signal temporal logic)[209,210]是MTL和MITL的进一步扩展, 允许系统的状态由实数组成. STL公式中可以包含关于状态中实数变量的谓词以及和MTL一样的组合方式. 例如, STL公式$\Box_{[0,10]}(x>7 \rightarrow \Diamond_{[2,4]} x<5)$的含义是: 在[0,10]时间区间内, 如果系统中x的值超过7, 那么一定在[2,4]时间之后, x的值会返回到5以下. 这可以用于描述一个控制系统的正确性, 要求控制策略在x过高时, 总能在确定时间内将x返回到正常范围.

目前讨论的MTL和STL的语义都是布尔语义, 也就是说, 每条轨迹或者满足性质或者不满足性质. 对于连续和混成系统, 这类语义在很多场景下并不恰当. 例如, 如果需要的STL安全性性质是$\Box_{[0,\infty]} x<5$, 即x的值永远小于5, 那么一个能够保证x永远小于4的系统与一个可能让x为4.99的系统相比显然是"更安全"的. 后者可能因为较小的扰动变得不安全, 而前者需要更大的扰动. 因此, 我们可以定义MTL和STL的量化语义[211,212], 对每个系统轨迹和STL性质赋予一个实数, 代表该轨迹满足性质的程度.

关于STL的模型检验已有一些研究[213], 然而, 通过模型检验验证一个混成系统总是满足STL性质还是相对困难的. 因此, 在更多情况下, STL应用于对系统的离线或实时监控, 通过观察和分析系统的运行轨迹判断系统是否违反了STL性质. 文献[214]提出了STL性质的离线监控方法. 文献[215]进一步给出了STL性质的在线监控算法, 可以在系统运行中实时判断其行为是否已经违反了STL性质.

6.3 可达集计算

可达集计算是混成系统安全设计及验证的主要技术手段, 其中, 系统连续行为可达集计算是主要技术发展瓶颈. 可达集一般不可准确计算, 因此文献中主要研究可达集上、下近似的计算.

可达集上近似是包含准确可达集的集合, 一般用于安全性验证. 如果可达集上近似与危险状态不相交, 则准确可达集与危险状态不相交, 从而系统是安全的. 可达集上近似是近30年混成系统安全验证的研究热点

之一, 其理论及计算方法得到了较好的研究, 如泰勒模型方法[216,217]、区间计算方法[218]、线性化方法[219]、矩方法[220]、PAC 方法[221,222]等.

可达集下近似是准确可达集的子集, 一般用于安全设计. 以可达-规避集合为例来说明可达集下近似如何用于系统安全设计, 如通过计算向后可达集的下近似, 可以确定使系统进入目标区域且在进入目标区域前规避危险状态的初始状态集合(也称为可达-规避集合). 可达集下近似计算近 8 年来才开始研究, 理论及方法相对匮乏. 但经过近几年的研究, 涌现出了一些理论方法, 如区间计算方法[223,224]、边界可达性方法[225]、凸优化方法[226,227]等.

6.4 验证工具

混成系统的模型检验工具包括 SpaceEx[228]、Flow*[229]、JuliaReach[230]、CORA[231]、BACH[232]、C2E2[233]等, 这些工具主要通过计算可达集的近似验证混成系统的安全性质. 另外一些验证工具, 例如 Breach[234], 可用于验证 STL 性质. 基于定理证明的混成系统验证工具包括用于微分动态逻辑的 KeYmaera X[235]和用于 HCSP 的 HHLProver[236].

SpaceEx[237]是一个混成系统的验证平台, 常用于验证带有非确定输入的分段仿射混成系统, 将显式多面体表示、隐式支持函数表示等结合起来表示近似可达集, 提高可处理系统的规模和效率的同时保持精确度. Johnson 等人[238]使用 SpaceEx 对多卫星系统的可靠性进行了分析, 即两颗卫星进行轨道转移时, 根据可靠性要求是否会进行会合的性质. Minopoli 等人[239]通过模型转化, 将 Matlab/Simulink 的模型转化成一个 SpaceEx 模型. 这样, SpaceEx 可以对所有 Simulink 建模的模型进行验证.

C2E2 在实际应用中对很多信息物理融合系统进行了验证分析, 如 Duggirala 等人[240]基于 C2E2 工具, 用信号时序逻辑(STL)对系统性质进行刻画, 对汽车的动力传动控制(powertrain control)系统进行了分析和验证.

KeYmaera 在混成系统的验证中具有广泛应用: 在文献[241,242]中, 作者基于 KeYmaera 工具分析了火车控制系统, 对其中系统的速度性质进行了验证; 文献[243]验证了网络物理运输系统的安全性; 文献[244]对飞行器控制系统的安全性质进行了验证. 除了交通运输领域以外, 文献[245,246]利用 KeYmaera 对机器人的控制系统进行了验证. 同时, KeYmaera 还常用于一些其他信息物理系统的形式化的验证和安全性分析[247,248]. HHLProver 被用于月球登陆车控制系统[249]和列车控制系统[250]的验证.

7 硬件系统

硬件验证是模型检验最早也最成功的工业应用案例. 很多模型检验技术的发展最初用于硬件模型检验, 例如符号模型检验[9]、限界模型检验[10]、插值法[30]、反例引导的抽象精化[11]、IC3 算法等[251,252]. 硬件模型检验的应用也很大程度上驱动了布尔可满足性问题(SAT)求解技术的发展[253]. 硬件模型检验的成功有两个主要因素: 第一, 由于硬件的每次流片非常昂贵, 并且不像软件那样易于更新, 任何设计错误都会带来巨大的损失; 第二, 相比于软件, 硬件设计的形式化模型相对简单, 一般不会牵涉无穷状态空间、递归调用等复杂特征. 因此, 硬件验证, 包括等价性验证和模型检验, 已经是硬件 EDA 工具中普遍带有的功能.

7.1 行为和性质规约

硬件模块的设计可以建模为一个迁移系统, 其状态空间由 n 个布尔变量组成(因此总状态数量为 2^n). 硬件模块是时钟驱动的, 每个时钟周期进行一次迁移. 因此, 硬件的行为模型可以由一个初始条件 $Init(x)$和一个迁移关系 $T(x,x')$组成. 其中, x 代表所有状态变量, $Init$ 是 x 上的一个谓词, T 是 x 和 x'上的谓词. 其中, x 代表迁移前的变量, x'代表迁移后的变量, 例如以下系统:

$$Init(x,y,z)=(x=1\wedge y=0\wedge z=0);$$

$$T(x,y,z,x',y',z')=(y'=x\wedge z'=y\wedge x'=z)$$

描述一个初始值为$\langle x,y,z\rangle=\langle 1,0,0\rangle$, 并且在$\langle 1,0,0\rangle$, $\langle 0,1,0\rangle$, $\langle 0,0,1\rangle$之间循环的系统. 在给定 $Init$ 和 T 之后, 迁移系统的运行轨迹包括所有满足 $Init(x_0)\wedge T(x_0,x_1)\wedge\ldots\wedge T(x_{n-1},x_n)$的 $x_0,x_1,\ldots,x_n$.

在硬件系统的验证中, 最关注的性质依然是安全性. 给定一个由谓词 P 描述的安全性质, 我们希望判断是否所有可以到达的状态都满足 P. 也就是说, 对于任何状态序列的 $x_0,x_1,\ldots,x_n$, 满足:

$$Init(x_0)\wedge T(x_0,x_1)\wedge\ldots\wedge T(x_{n-1},x_n)\rightarrow P(x_n).$$

例如, 以上给出的系统满足任何时候 x, y, z 中至少 1 个变量为 1, 也就是满足安全性性质 $x=1\vee y=1\vee z=1$. 除了安全性质, 我们也可以考虑硬件系统的活性性质, 以及一般的 LTL 性质.

由于硬件模型检验普遍应用, 已经产生了标准的行为和性质描述语言和文件格式. 在行为描述方面, 一般采用基于网表的格式, 例如 AIGER[254]. 网表的格式是在布尔级的, 因此会失去字节级的信息, 例如加法、乘法和比较等运算. 最近的硬件模型检验竞赛开始推荐使用字节级的 Btor2 格式[255]. 在性质描述语言方面, 工业界普遍使用的标准包括 PSL (property specification language)[256]和 SVA (system verilog assertion)[257].

7.2 验证方法

硬件模型行为和性质的描述本身相对简单, 可以使用一般的 LTL 模型检验方法验证. 因此, 硬件验证的主要挑战来自待验证的模型规模非常庞大, 可能涉及上万个布尔变量, 而状态空间则是变量个数的指数. 硬件模型检验的研究主要关注如何通过符号化的方法, 结合基于不变式生成的定理证明技术来处理庞大的状态空间.

第 1 个在显式状态空间基础上的突破是符号模型检验[9], 其主要思想是, 使用二元决策图(binary decision diagram, BDD)代表可达的状态空间. 在确定变量之间的排序后, 二元决策图可以唯一代表任何布尔表达式, 并提供模型检验所需要的在布尔表达式上的操作. 下一个重大突破是限界模型检验[10], 通过展开迁移关系, 将模型在有限步内是否正确转换为一个 SAT 求解问题. 这种方法可以用于快速发现模型中的错误, 但一般无法证明模型的正确性.

为了能够验证模型的正确性, 也就是说在任意多步内都不会出错, 主要使用的想法是寻找模型的归纳不变式. 一个表达式 I 对于初始条件 $Init$、迁移关系 T 和正确性性质 P 是归纳不变式, 如果满足以下条件:

$$Init(x)\rightarrow I(x),\ I(x)\wedge T(x,x')\rightarrow I(x'),\ I(x)\rightarrow P(x).$$

如果存在这样一个归纳不变式, 则可证明模型的确满足安全性性质 P. 归纳不变式可以扩展到 k 步的归纳不变式, 即如果性质 I 连续 k 步成立, 可以推导出下一步也成立. 基于 k 步不变式的正确性验证方法称为 k-induction[258]. 在文献[30]中, McMillan 提出了基于插值的不变式生成方法, 其主要思想是: 如果限界模型检验没有找到系统在 n 步内的反例, 则说明转换后的布尔可满足性问题是无解的. 从这个问题无解的证明可以生成一个插值(interpolant), 仅包含系统在 1 步迁移之后的变量. 这个插值是系统 1 步之后可达集的上近似. 通过不断迭代, 可以得到系统在多步之后可达集的上近似, 最终希望获得不变式.

继插值法之后的下一个主要进展是 IC3 算法[251,252], 也称为 PDR (property directed reachability). 这个算法在验证过程中维护一个表达式的序列 $F_0,F_1,\ldots,F_n$, 其中, F_0 代表初始集合, 每个 F_0 代表在 i 步之内可达集的上近似. 通过一系列对于一步迁移可达性的求解, 算法不断从 F_0 移出元素, 使其更加精确, 直到出现 $F_i=F_{i+1}$, 则 F_0 是需要的归纳不变式. 与限界模型检验不同, IC3 算法不需要展开迁移关系, 因此减小了 SAT 求解问题的规模. 最近的一个研究热点是 IC3 算法用于字节级的模型. 最初的一些工作[259,260]将 IC3 的算法直接扩展到字节级, 将 SAT 求解问题替换为 SMT 求解问题. 近期的研究首先将字节级的模型通过谓词抽象或其他方法抽象为布尔模型, 在布尔模型上使用 IC3 算法, 并基于反例进行抽象精化. 这方面的代表性工作包括 IC3+IA (implicit abstraction, 隐式抽象)[261]和 IC3+SA (syntax-guided abstraction, 语法引导抽象)[262].

硬件模型检验的算法在近期依然是一个研究热点. 除了 IC3 之外, 其他一些方法, 包括 CAR (complementary approximate reachability)[263]和语法引导的引理生成(SyGuS-APDR)[264]也在特定问题上取得了成功. 除了传统的模型检验算法, 其他硬件验证方法包括符号轨迹执行(symbolic trajectory execution, STE)[265]和基于定理证明的方法[266].

7.3 验证工具

硬件模型检验竞赛(hardware model checking competition, HWMCC)是每 1 年或 2 年举行一次的模型检验工具之间的竞赛, 在工业界提供的实例上比较工具的性能. 目前, 性能处于前列的工具包括 AVR[267]、nuXmv[268]和 Pono[269]. 其他一些早期的应用于硬件模型检验的工具包括 NuSMV[75,270]和 ABC[271]. Forte[272,273]是一个基于符号轨迹执行和定理证明方法的硬件验证工具, 在 Intel 内部有广泛的使用.

8 通信协议

通信协议是现代计算机网络连接的约定标准, 是保障计算机之间信息传递的安全性的重要基础. 通信协议可以用于用户之间或用户和服务之间进行认证, 并产生双方共享的密钥, 保证双方通信的安全性. 然而, 如果通信协议的设计或实现存在漏洞, 则可能允许入侵者监听通道上发送的私密信息, 也可能允许入侵者伪装成其中一方与另一方进行通信, 造成信息泄露等通信安全问题. 因此, 保障通信协议的安全性对于计算机网络是十分重要的. 基于模型检验的方法在验证通信协议方面已经有了很多研究. 我们在本节将主要阐述验证使用的协议和攻击者的模型、性质描述, 以及验证方法和工具.

8.1 协议和攻击者模型

在描述通信协议时, 我们首先需要确定以下概念[274].

- 通信方(agent): 表示能够执行通信操作的实体/用户.
- 角色(role): 表示通信方在一次协议运行中所扮演的角色(例如发起方(initiator)、响应方(responder)以及中间的服务器方(server)).
- 协议(protocol): 表示多个角色之间发送消息的规则和顺序.
- 会话(session): 表示协议的一次执行.

需要注意的是, 每个通信方可以参与同一个协议的多个会话(session), 并在其中扮演不同的角色. 例如, 用户 A 可以作为协议的发起方与 B 通信, 并同时在另一个会话中作为响应方响应 C 的通信.

在通信协议的分析中, 最普遍使用的攻击者模型是 Dolev-Yao 模型[275]. 这个模型假设攻击者可以截获任何网络中传输的数据, 并对其进行修改, 例如应用加密和解密函数. 但是, 攻击者不能在没有密钥的情况下对消息进行解密(即假设加密算法是不能被攻破的).

我们使用 Needham-Schroeder 协议[95]作为案例. 这是两个角色之间的身份认证协议. 角色 A 和 B 分别拥有各自的公钥 $pk(A)$, $pk(B)$和私钥 $sk(A)$, $sk(B)$. 首先, A 产生一个随机的 nonce N_A, 然后将$\{|A,N_A|\}$使用 $pk(B)$加密后发送给 B; 接下来, B将$\{|N_A,N_B|\}$使用$pk(A)$加密后发送给A; 最后, A将$\{|N_B|\}$使用$pk(B)$加密后发送给 B. 经过这个协议, A 和 B 确认了各自的身份.

然而, 这个协议在 Dolev-Yao 攻击者模型下是不安全的. 设 I 为攻击者, 存在以下“中间人攻击”, 可以使得 I 伪装成 A 与 B 通信.

1. $A\rightarrow I$ 发送$\{|A,N_A|\}_{pk(I)}$: A 希望和 I 通信, 因此将$\{|A,N_A|\}$使用 $pk(I)$加密后发送给 I.
2. $I\rightarrow B$ 发送$\{|A,N_A|\}_{pk(B)}$: I 将 A 发送的 nonce 解密后, 按照协议的第 1 步伪装成 A, 将$\{|A,N_A|\}$加密发送给 B.
3. $B\rightarrow I\rightarrow A$ 发送$\{|N_A,N_B|\}_{pk(A)}$: B 正确地回复给 A. 但我们可以假设这个消息被 I 截获, 并未经修改传递给 A.
4. $A\rightarrow I$ 发送$\{|N_B|\}_{pk(I)}$: A 收到消息后, 认为 N_B 是由 I 产生的, 并按协议的第 3 步将$\{|N_B|\}$加密发送给 I.
5. $I\rightarrow B$ 发送$\{|N_B|\}_{pk(B)}$: I 收到消息后, 对其解密获得 N_B, 然后再加密发送给 B, 从而完成与 B 的协议.

当这个攻击过程完成时, B 认为和 A 建立了联系, 并且只有 A 和 B 拥有 N_A 和 N_B. 但实际上, A 并没有向 B 发起会话, 而且攻击者 I 也拥有 N_A 和 N_B. 这可能造成 B 泄漏私有信息.

除了 Dolev-Yao 模型之外, 最近也有对更强的攻击者模型的研究, 例如 eCK[276]. 这个攻击者模型假设攻

击者具有所有Dolev-Yao模型中的能力, 另外允许攻击者可能获得某个通信方的私钥. 使用eCK模型, 我们可以描述完全正向保密(perfect forward secrecy)性质: 即使攻击者获取了某个通信方的私钥, 也无法破解之前发生的会话的密钥, 或者面对该通信方时冒充为其他人.

8.2 性质描述

在本节, 我们列举通信协议的一些安全性性质. 对于通信协议, 一类非常重要的性质是保证某些信息不会被攻击者截获. 这类性质称为私密性(secrecy): 对于通信方 A 在 R 角色下的一条信息 t, 我们说 t 是私密的当且仅当 A 在 R 角色下与其他可信的通信方进行通信时, t 不能被入侵者通过监听信息推断.

Lowe 的文章[277]中提出了一个身份认证协议性质的分类, 将性质分为如下 4 个等级.

1. 存活性(aliveness): 当通信方 A(发起者)认为与通信方 B 完成一个通信协议时, 若 B 之前运行过这个协议, 我们就说通信方 B 对于 A 具有存活性. 这个性质是 4 类性质中最弱的一个.
2. 弱一致性(weak agreement): 当通信方 A(发起者)认为与通信方 B 完成一个通信协议时, 若 B 在之前运行过这个协议, 并且确信是与 A 运行这个协议, 我们就说通信方 B 对于 A 具有弱一致性.
3. 非单射一致性(non-injective agreement): 当通信方 A(发起者)认为与通信方 B 完成一个通信协议时, 在协议中设定一个双方交换的数据 ds, 如果通信方 B 确信与 A 运行过这个协议, 并且双方对数据 ds 达成了一致, 我们说通信方 B 对于 A 具有非单射一致性.
4. 单射一致性(injective agreement): 当通信方 A(发起者)认为与通信方 B 完成一个通信协议时, 在协议中设定一个双方交换的数据 ds, 如果通信方 B 确信与 A 运行过这个协议, 并且双方对数据 ds 达成了一致, 且对于 A 的每一次协议的运行, 均在 B 上拥有唯一对应的运行, 我们就说通信方 B 对于 A 具有单射一致性. 与非单射一致性相比, 这个性质表达不会出现重放攻击.

在以上性质中, 我们也可以加入近期性(recentness)的要求, 即定义一个验证时间 t, 在通信方 A 完成一个通信协议时, 要求对应的通信方 B 在最多 t 时间之前运行过这个协议.

根据以上的定义, 我们可以看出, Needham-Schroeder满足存活性, 但不满足弱一致性: 通过中间人攻击, I 可以使 B 认为和 A 完成了通信协议. 虽然 A 之前运行过这个协议, 但它确信是与 I, 而不是 B 运行的协议. 如果将协议第 2 步传输的信息从$\{|N_A,N_B|\}$改为$\{|N_A,N_B,B|\}$, 则可以避免这种攻击, 得到的协议满足单射一致性.

8.3 验证方法和工具

Lowe 最早基于 FDR 开发了通信协议验证工具 Casper[278], 发现了对 Needham-Schroeder 协议的攻击[279], 使得模型检验用于验证通信协议获得广泛关注. 目前, 主要的用于验证通信协议的工具包括 Tamarin[280,281]、ProVerif[282]、AKiSs[283]等.

Tamarin 将通信协议和攻击者建模为一个迁移系统. 系统的状态表示攻击方所知的信息、网络上的信息、新产生的值和协议方的状态, 表达为一个由已知事实(fact)组成的多重集. 系统的迁移由多重集的重写规则表达, 描述协议的规则和攻击者的能力. 加密、解密等函数的代数性质使用一个等式理论(equational theory)表达. 安全性质表达为关于迁移系统的路径的一阶逻辑公式. Tamarin 使用约束求解算法对性质进行验证, 通过反向搜索的方式, 试图找到一个违反性质的路径: 如果找到了这个路径, 则说明性质不成立, 并在用户界面上展示这个反例; 如果不能找到路径, Tamarin 可以生成一个该性质的证明. 由于所涉及的验证问题是不可判定的, Tamarin 的自动验证不能保证终止. 但是在很多实际例子中, Tamarin 能够在很短的时间内完成验证或找到反例. 除了安全性质, Tamarin 也能验证协议中的一些隐私性质及观察等价性(observational equivalence). Tamarin 在很多安全通信协议上取得了非常成功的应用, 例如, Schmidt 等人使用 Tamarin 验证了 NAXOS 协议在 eCK 攻击者模型下的安全性[280]; Cremers 等人使用 Tamarin 验证了 TLS 1.3 协议的抽象模型[284]; Basin 等人使用 Tamarin验证了 5G AKA 协议[285], 验证结果表明该协议存在安全漏洞, 并提供了修改建议; 另外, Tamarin 也被用于验证带有循环和可变全局状态的协议, 例如, Kunnemann 等人使用 Tamarin 验证用于网络设备的身份认证的 Yubikey 协议[286].

ProVerif 和 AKiSs 基于 Applied pi 演算对协议建模. Applied pi 演算[287]是 pi 演算的一个更接近应用的变种, 主要用于对通信协议进行建模. Pi 演算可以看作 CCS 的一个扩展, 允许进程之间的通信通道结构发生变化, 例如一个进程将通道发送给另一个进程. Pi 演算的一个弱点是提供的原语较少, 因此很多常用的构造, 例如整数和多元组, 需要额外的通信表达. 这使得对协议中原有的通信进行推理变得更加复杂. Applied pi 演算在 pi 演算的基础上添加了内置函数(例如加密和解密), 并把名称、变量和通信通道区分开来, 避免混淆. 其中, 名称(names)专门用于代表新产生的通道、随机数、密钥等.

基于 Applied pi 演算验证的基础是进程之间的等价性概念, 其中, 静态等价性(static equivalence)表达两套替换之间的不可区分性. 观察等价性和带标注的互模拟关系(labeled bisimilarity)表达了进程之间行为的不可区分性, 其中, 观察等价性的定义需要考虑所有可能与进程交互的环境, 因此直接验证非常复杂. 相比之下, 带标注的互模拟关系更易于直接验证. Applied pi 演算理论的一个核心定理是, 观察等价性和带标注的互模拟关系是等价的.

ProVerif 工具实现了 Applied pi 演算模型的可达性(reachability)、一致性(correspondence)和观察等价性的验证算法, 其中, 一致性[288]表达如果某个事件 e 执行, 那么另一个事件 e' 必然在之前执行. 单射一致性是满足单射关系的一致性, 即如果事件 e 执行, 那么此前一定执行过事件 e', 且若事件 e 不同, 对应的 e' 也不同. 验证一致性时, 先将密码协议写成 Horn 子句的集合, 将需验证的一致性性质写成目标结论, 然后通过饱和过程将子句集进行简化, 最后通过深度优先搜索算法判断目标结论是否能从子句集中推导出来[289]. 验证观察等价性时, 用 bad 来表示不满足观察等价性的结果, 通过类似一阶逻辑中的 resolution 过程对子句集进行饱和处理, 如果饱和子句集中不包括结果为 bad 的子句, 则观察等价性成立[290]. 根据上述算法, ProVerif 可用于验证通信协议的保密性(secrecy)、认证的正确性等, 例如, Abadi 等人[291]使用 ProVerif 验证了 JFK (just fast keying)[292]快速密钥交换协议, 结果表明, 该协议保证了密钥的保密性并能抵抗拒绝服务攻击; Abadi 等人[293]还使用 ProVerif 验证了认证邮件协议, 证明当且仅当发送方收到消息回执时, 接收方接收到消息. 此外, ProVerif 还被用于验证电子投票协议的安全性[294−296], 可验证投票者投票内容的隐私性、投票结果的公平性等.

AKiSs 可用于验证有限会话假设下通信协议的观察等价性, 输入语法与 Applied pi 演算语法相似, 输出结果为所查询的两个进程是否满足观察等价性. 由于 AKiSs 只考虑有限会话的场景, 并使用了不同的算法, 可以用于验证一些 ProVerif 无法验证的协议, 例如 Okamoto 电子投票协议[283].

8.4 可证明的安全性

另一种完全不同的验证方式是使用逻辑推理来证明协议的安全性, 称为可证明的安全性(provable security). 其主要步骤是: 首先, 定义协议模型和攻击者模型; 然后, 定义对密码系统的安全假设; 最后, 通过归约(reduction)将攻击者针对协议安全性质的攻击行为最终转化为密码系统安全假设的违反, 从而说明协议的安全性质(在密码系统安全假设成立的情况下)不会被违反. Barthe 等人使用概率程序对协议和攻击者行为进行建模, 并使用概率关系霍尔逻辑(probabilistic relational Hoare logic)证明协议的安全性, 实现了工具 EasyCrypt[297], 并用于证明安全协议的保密性[298].

9 信息流

信息流性质一般指信息不会在未经授权的情况下被访问、修改和泄露, 同时保证被授权的一方在给定的权限内, 合理地访问和使用信息. 我们这里主要介绍 3 类信息流性质: 无干扰性(noninterference), 即高机密级别的操作不能影响低机密级别的状态和输出; 不透明性(opacity), 即系统的秘密对外界的入侵者来说不透明; 以及隐私性, 即用户和用户的数据之间的对应关系一定程度地被隐藏起来, 使得攻击者无法推导. 我们针对这 3 类性质, 分别从模型、性质描述和验证这些方面进行展开.

9.1 无干扰性

无干扰性的定义依赖一个安全模型(security model), 描述哪些用户/进程可以访问或修改哪些资源, 或者

哪些用户/进程之间可以发送信息. Rushby 基于不同迹之间的 unwinding 关系等价性, 首次给出了具有非传递性的无干扰性的形式定义[299]. 假设系统 M 里存在多个域(domain), 记为 D, 这些域可以理解为用户或进程. A 为动作集合, S 为状态集合, O 为输出集合. 定义 dom:$A\to D$, 对于任意的动作 a, $dom(a)$返回它所在的域. 定义两个函数: $step$:$S\times A\to S$, 记录动作执行后, 状态的改变; 以及 $output$:$S\times A\to O$, 记录动作执行后, 产生一个输出. 一个安全政策(policy)定义为域上的一个干扰关系$\rightsquigarrow$, 其中, $d\rightsquigarrow b$ 表示域 d 允许向域 b 发送信息, 即 d 干扰 b. 很多情况下, 无干扰性是不满足传递性的, 例如, 我们可能需要描述用户 d 可以向用户 b 传输信息, 但只能通过通信渠道 c. 这可以描述为 $d\rightsquigarrow c$, $c\rightsquigarrow b$ 和$\neg(d\rightsquigarrow b)$. 另一种常见的情况是信息存在多个保密等级, 高保密等级的信息可以通过一个解密机制降为低安全等级. 因此, 高安全等级可以向解密机制传递信息, 解密机制可以向低安全等级传递信息, 但高安全等级不能直接向低安全等级传递信息.

在给出定理之前, 先介绍几个定义. 系统 M 是可视角分区的(view-partitioned), 如果对于任意 $u\in D$, 存在 S 上的一个等价关系$\sim^u$. 这个等价关系是输出一致(output consistent)的, 如果

$$s\sim^{dom(a)}t\to output(s,a)=output(t,a).$$

即如果 s 和 t 是动作 a 域上的等价状态, 那么它们执行 a 后, 输出是一致的. M 是弱步调一致的(weakly step consistent), 如果 $s\sim^u t\wedge s\sim^{dom(a)}t\to step(s,a)\sim^u step(t,a)$, 即如果 s 和 t 关于域 u 是等价的, s 和 t 关于动作 a 的域是等价的, 那么 s 和 t 执行 a 后, 关于域 u 还是等价的. M 局部满足$\rightsquigarrow$关系, 如果$\neg(dom(a)\rightsquigarrow u)\to s\sim^u step(s,a)$, 即如果动作 a 的域不干扰 u, 那么 s 和从 s 执行 a 后的状态关于 u 是等价的. 可以得到如下的定理, 称为非传递无干扰政策的 unwinding 定理.

定理. 设$\rightsquigarrow$为一个非传递无干扰政策, 系统 M 是可视角分区的, 且满足: (1) M 是输出一致的; (2) M 是弱步调一致的; (3) M 局部满足$\rightsquigarrow$关系, 那么 M 对于政策$\rightsquigarrow$是安全的, 即对于任何域 u 和动作序列α, 执行α然后执行 u 的输出应该等于仅执行α中能够影响 u 的行为然后执行 u 的输出.

后来, von Oheimb 对这个定义进行了扩展, 能够处理非确定系统的情况, 并且能够描述更多的性质, 例如无泄露性质(nonleakage)和无影响性质(noninfluence)[300].

以上的工作都是基于状态机, 除此之外, 还有基于进程代数的无干扰性的建模和验证[301−303]. 还有一类重要的工作是基于语言的类型系统的验证, 首先针对无干扰需求, 定义类型系统; 然后, 定义无干扰性的语义; 最后证明, 只要系统满足类型系统, 那么系统的执行必定满足无干扰性的语义. 这样, 判断一个系统是否满足无干扰性, 只需要检查是否满足类型规则. 这方面的工作很多[304−307].

以上的工作都是针对静态政策, 无干扰性还被推广到动态政策, 即系统的信息流政策可以在执行过程中改变, Eggert 等人给出了形式定义[308,309].

对于无干扰性的验证, Hadj-Alouane 等人[310]首次提出一个充分且必要的条件, 用于判断无干扰性是否成立, 然而算法的复杂度是双指数的. Eggert 等人针对几种不同类型的无干扰性, 提出了多项式复杂度的判定算法[311]. 对于复杂的操作系统内核, 无干扰性也是一类重要的需求, 验证通常需要交互式定理证明的方法. 通过交互式定理证明验证无干扰性质的工作包括文献[312,313].

9.2 不透明性

不透明性(opacity)可以用来描述系统的安全与隐私性质. 有的研究者认为, 不透明性可以归类到关于系统秘密行为的隐私问题. 另外, 匿名(anonymity)和保密(secrecy)性质也都可以用不透明性来阐释. 不透明性的目标是保护系统的秘密, 使其对外界的入侵者来说不透明. 具体来说, 外界入侵者掌握系统的模型构造的全部内容, 但是在观测系统的行为时, 信息流的局部不能被观测到, 或者不能被准确地观测到. 外界入侵者将基于上述信息, 对系统的秘密进行推断. 如果在系统的任何一种运行下, 外界入侵者都不能准确发现系统的秘密, 则称系统对该入侵者而言是不透明的. 由上面的描述可知, 不透明性的研究中带有两个参数: 系统的秘密, 记作 S; 入侵者的观测能力, 记作 P, 因此, 不透明性表述为: 系统关于其秘密 S 与入侵者观测能力 P 而言是不透明的.

基于秘密的类型, 不透明性可以分为两大类: 当系统的秘密是一个语言, 即一个“秘密的行为序列”的集

合时, 称为基于语言的不透明性(language-based opacity, LBO); 当系统的秘密是系统的一个状态集合, 即一个“秘密的状态”的集合时, 称为基于语言的不透明性(state-based opacity, SBO). 在系统为离散动态系统的背景下, 共有两种系统模型: 自动机和 Petri 网. 另外, 在考虑物理时间的条件下, 系统模型为时间自动机. 不透明性的验证问题一般情况下是不可判定的. 对于自动机模型, Saboori 和 Hadjicostis 以有限状态机为模型, 分别研究了初始状态不透明性(Initial state opacity)、K 步不透明性(K-step opacity)以及无穷步不透明性(infinite step opacity)[314−317]. 在两人的研究基础上, 其他研究者进一步考虑了多种不透明性的验证, 包括系统不透明性验证、确保不透明性的控制器/调度器分析综合等, 使用的方法包括但不限于监督(supervision)和增强(enforcement)等. 总的来说, 无论对于确定性转移函数还是非确定性转移函数, 针对给定秘密和入侵者的不透明性验证, 对逻辑模型来说都是一个是或否的问题. 另一方面, 如果研究者对于量化系统中可能的信息泄露风险感兴趣, 则可以考虑使用概率模型求解问题[318−320].

Petri 网模型是一种用来研究离散事件系统的基本数学工具. Petri 网模型不仅像自动机模型一样具备直观的图表达与严格的数学表达, 而且在某种程度上还能扩展自动机模型的建模能力. 文献[321]是第一篇基于 Petri 网模型研究系统不透明性的工作, 其给出了不透明性的初步定义(初始透明、最终透明、一直透明), 并且证明了在有界 Petri 网模型中的上述 3 种不透明性是可验证的. 后续的以 Petri 网作为系统模型的不透明性研究基本可分为两类, 即针对基于状态的不透明性研究[322]与针对基于语言的不透明性研究.

对于时间自动机模型, 问题的复杂程度将大幅度上升, 甚至出现不可判定的情况. 解决不可判定的问题有以下方式: (1) 找到时间自动机的子集, 使得不透明性在该子集上变成可判定问题; (2) 缩小不透明性的范围, 使问题变得可解[323,324].

9.3 隐私性

一个系统满足隐私性, 如果它能够将用户和用户的系统数据之间的对应关系(correspondence)一定程度地隐藏起来, 而攻击者无法由已知信息推导出这个对应关系. 按照隐藏的程度不同, 系统将提供不同强度的隐私性. 最常见的隐私性质有[325]: 不可关联性(unlinkability), 即攻击者不能确认用户和系统数据是否相关联; 匿名性(anonymity), 即攻击者不能将某个用户从一个匿名用户集合中识别出来; 似是而非的否认(plausible deniability), 即当用户否认曾经调用系统时, 攻击者既不能确认也不能反驳; 不可观测性(unobservability), 即攻击者无法确定用户是否调用过这个系统; 等等.

Bohli 等人[326]形式化地定义了各种隐私概念以及攻击者模型, 并描述了不同强度隐私之间的层次关系. 有些情况下, 系统的目标用户有可能与攻击者合作, 而第三方(其他用户)也有可能与攻击者或者目标用户合作. 这些都将影响目标用户的数据隐私. 这类隐私称作增强隐私. Dong 等人[327]使用 applied pi 演算, 对增强隐私进行了形式建模.

9.4 性质的分类

前面所介绍的功能性质、时序性质、通信性质、协议的鉴权性质等, 都是单个轨迹的性质, 即描述系统的每个运行轨迹都满足某个性质. 但是, 很多信息安全性质不能如此表示, 因为它们需要表达系统的两个或多个运行轨迹之间的关系, 例如上面介绍的无干扰性. Clarkson 等人提出了超性质(hyperproperties)[328], 它是对基于轨迹的性质的一个扩充, 用于描述系统的多个轨迹之间满足的性质.

Clarkson 等人后来又将时序逻辑扩展到超性质中[329]. HyperLTL 是对 LTL 的扩展, 可以对定义在路径集上的超性质进行描述. 一个 HyperLTL 公式是路径集合的集合. 在 LTL 的基础上, HyperLTL 加入了原子命题 a_h, 表示路径 h 满足命题 a; 同时, 允许存在和任意路径量词. 因此, 在一个 HyperLTL 公式中, 可以同时描述几个不同路径之间的关系. 类似地, 对 CTL*进行扩展, 定义了 HyperCTL*. Finkbeiner 等人[330]研究了 HyperLTL 的可满足问题, 证明了复杂度是 PSPACE 完全的. 他们后来在此基础上实现了超性质的验证工具 EAHyper[331], 用于验证某个超性质是否满足或者两个超性质之间是否等价.

最后, 对于安全-活性的分类, 已经证明, 任何一个超性质都可以表示为某一个超安全性和某一个超活性

的合取[328].

10 人工智能系统

近些年来, 基于深度神经网络的人工智能系统得到了飞快的发展以及越来越广泛的应用, 智能系统的正确性也因此受到越来越多的关注. 关于深度神经网络的可信性和安全性已经有了很多研究, 文献[332]为这些研究提供了完整的综述. 本节首先从深度神经网络的定义和基本原理入手; 然后对深度神经网络的攻击方式和性质进行分类, 主要关注安全性以外的其他性质; 最后简要概述这些性质的验证方法和工具.

10.1 深度神经网络定义

人工神经网络, 简称神经网络, 是一种模仿生物神经网络(动物中的中枢神经系统, 特别是大脑)的结构和功能的数学模型或计算模型, 用于对函数进行估计或近似. 经过几十年的发展, 神经网络理论在模式识别、自动控制、信号处理、辅助决策、人工智能等众多研究领域取得了广泛的成功. 深度神经网络是深度学习的一种框架, 它是具备至少1个隐藏层的神经网络. 与浅层神经网络类似, 深度神经网络也能够为复杂非线性系统提供建模, 但多出的隐藏层为模型提供了更多抽象层次, 因此提高了模型的抽象能力. 神经网络的基本结构可以分为3层: 输入层、隐藏层、输出层, 各层由神经元和神经元之间的权值组成.

10.2 深度神经网络的工作原理及特点

在深度神经网络中, 信号从一个神经元传入到下一个神经元之前是通过线性加权和来计算的, 而进入下一层神经元需要非线性的激活函数, 继续往下传递, 如此循环下去. 由于这些非线性函数的反复叠加, 才使得神经网络有足够的能力来抓取复杂的特征.

如果不使用(非线性)激活函数, 每一层输出都是输入的线性函数. 因此, 无论神经网络有多少层, 输出都是输入的线性函数, 这样就和只有一个隐藏层的效果是一样的, 无法逼近复杂的非线性函数. 这就是为什么激活函数都是非线性的. 比较常用的激活函数包括 Sigmoid、tanh 和 ReLU 函数.

- Sigmoid 函数: $f(x)=\dfrac{1}{1+\mathrm{e}^{-w^T x}}$.
- tanh 函数: $f(x)=\dfrac{1-\mathrm{e}^{-2w^T x}}{1+\mathrm{e}^{-2w^T x}}$.
- ReLU 函数: $f(x)=\begin{cases}0, \text{ for } x\leqslant 0\\ x, \text{ for } x>0\end{cases}$.

深度神经网络具有以下特点.

(1) 神经网络与传统的参数模型最大不同之处在于, 它是数据驱动的自适应技术, 不需要对问题模型做任何先验假设. 在解决问题的内部规律未知或难以描述的情况下, 神经元可以通过对样本的学习训练, 获取数据之前隐藏的函数关系. 适用于解决一些利用假设和现存理论难以解释, 却具备足够多数据和观察变量的问题.

(2) 神经元具备泛化能力. 神经网络可以通过对输入样本数据的学习训练, 获得隐藏在数据内部的规律, 并利用学习到的规律来预测未来的数据.

(3) 神经网络是一种具有普遍适用性的函数逼近器. 它能以任意精度逼近任何非线性函数. 传统的预测模型由于存在各种限制, 不能对复杂的变量函数进行有效的估计. 神经网络的内部函数形式比传统的统计方法更为灵活、有效.

(4) 神经网络算法是非线性的方法. 神经元之间相互制约和相互影响, 使得整个网络从输入空间到输出空间形成了非线性映射, 可用于处理一些环境信息十分复杂、知识背景不清楚和推理规则不明确的问题.

10.3 深度神经网络的攻击和性质分类

我们对神经网络的可能攻击方式和期望满足的性质进行分类.

首先, 神经网络可能遇到的攻击有以下几类.

- 训练时攻击(training-time attack): 攻击者试图通过在训练集里掺入恶意的数据, 从而影响训练出来的神经网络.
- 测试时攻击(test-time attack): 攻击者试图构造对抗性的测试案例(adversarial example), 使得神经网络做出错误的判断. 这是最常见也是研究最广泛的攻击方式.
- 模型提取攻击(model extraction attack): 攻击者通过对一个模型询问, 试图获取同样的模型. 这可能会造成商业数据或个人隐私数据的泄露.

深度神经网络的性质可以分为以下几类[333].

- 输入-输出的鲁棒性. 这种正确性性质强调: 如果输入没有很大的改变, 输出也不应有很大的改变. 这个定义对应着大多数对抗性攻击, 通过对输入微小的扰动来误导神经网络得出错误的判断. 此外, 我们还可以把鲁棒性定义为: 如果输入相似, 则神经网络的输出相似.
- 输入-输出关系. 这种正确性性质规定: 当神经网络的输入满足条件 P 时, 输出应当满足条件 Q.
- 系统正确性. 这种定义方式的主要思想是: 一个神经网络应该考虑为一个更大系统的一部分. 例如, 神经网络可以作为这个系统的感知部件. 因此, 神经网络的正确性应当从整个系统的正确性考虑. 例如, 整个系统的正确性可以通过 STL 等时序逻辑公式描述. 在这个定义下, 神经网络的对抗样本是一个可以造成整个系统违反 STL 公式的样本. 同样地, 在对抗样本上进行重新训练的目标是, 使得系统在更多情况下能够满足设置的 STL 公式. 这种对抗和重新训练的概念被称为语义对抗深度学习(semantic adversarial deep learning)[334].
- 语义不变性. 对于一些应用, 输入空间 X 可以被分为多个等价类, 记为 $X_1,X_2,X_3,\ldots$ 正确性的定义是, 神经网络对于每个等价类都必须得到同样的输出. 这里, 等价类的划分通常和领域相关. 例如, 在图像识别领域, 我们可以说在一个图片里对一个物体进行平移和缩放将得到同一个等价类的图片.
- 单调性. 在一些应用里, 输入空间 X 和输出空间 Y 上存在自然的偏序. 例如, 如果神经网络用于决定一个贷款申请是否应该被批准, 那么仅仅增加申请人的输入应当只会增加批准的可能性.
- 公平性. 近期神经网络的公平性越来越受到关注. 我们期望神经网络在做出判断时不会受到某些因素(例如年龄、种族等)的影响. 这种公平性的定义有多种方式, 例如, 可以要求神经网络给出某个输出的概率与某些敏感的特征在统计意义下是独立的; 而另一种描述方式则依赖因果模型, 定义为神经网络对于某个输入是公平的, 如果对这个输入修改某些敏感特征后, 神经网络的输出保持不变.

10.4 深度神经网络的验证

深度神经网络的输入/输出常常受到环境、设备及安全等因素的限制, 并且在实际应用中, 也面临着实际任务需求的限制. 因此, 如果在一些安全攸关应用领域使用深度神经网络, 例如商用飞机防撞、大规模发电厂、化工厂控制等, 我们必须对部署的神经网络进行形式验证, 以确保其能够安全运行. 深度神经网络形式验证主要涉及验证网络的输出是否满足给定的安全性质规约. 考虑到实际应用中深度神经网络的非线性、高度复杂等特性, 对其进行形式验证极具挑战性.

目前已有的神经网络验证工具包括 Reluplex[335]、Planet[336]、ReluVal[337]等. Reluplex 和 Planet 是两个基于 SMT 求解器的神经网络验证工具, 使用 Davis-Putnam-Logemann-Loveland (DPLL)算法的体系结构来分割案例和排除冲突子句. Reluplex 扩展了 Simplex 算法中的规则, 首先, 通过主元规则找到线性约束的解; 然后再调整赋值, 使每一个 ReLU 激活函数的输入输出匹配. Planet 采用线性近似的方法对神经网络进行过近似, 并通过逻辑公式判断 ReLU 和 max-pooling 节点的条件是否被满足. ReluVal 使用区间算术来验证神经网络的性质, 在给出运算符输入值的上下界后, 都可以得到输出结果的上近似范围, 并且这种上近似的计算可以随神经网

络递进.

ERAN (ETH robustness analyzer for neural networks)是苏黎世联邦理工大学开发的神经网络鲁棒性的验证工具, 针对多种扰动实现了多个验证算法, 例如基于 zonotopes 抽象域的 DeepZono[338]和基于 DeepPoly 抽象域[339]的方法. 其他神经网络鲁棒性验证工具包括 Fast-Lin[340]和 CROWN[341], 其中 Fast-Lin 仅考虑 ReLU 激活函数的情况, 并利用了 ReLU 网络的特性; CROWN 针对更一般的神经网络, 使用激活函数的线性或二次函数上下界对其进行逼近.

此外, 还有许多工作从不同的角度和方向对神经网络的验证进行了探索. Seshia 等人[333]对深度神经网络的形式规范进行了综述. 关于 DNN 的设计、(对抗)分析和验证的大量文献给出了各种性质, 在展现这些性质的同时, 关于两种不同的角度进行了讨论: 第 1 种是基于性质含义和相关的语义行为类型, 第 2 种是基于由轨迹集定义出发的轨迹理论. Dreossi 等人[342]提出了一种形式化的神经网络(和机器学习模型)的鲁棒性对抗分析问题的规约, 将对抗性输入生成问题表述为一个形式化的验证问题, 包括对抗干扰的可容许限制、距离限制和目标行为限制. 这种统一的形式鲁棒性公式可以概括不同例子中的鲁棒性定义、不同的对抗环境(黑盒、白盒攻击)和不同的对抗生成技术. 在清晰地描述不同的对抗目标的同时, 也便于不同技术和对抗分析方法(基于形式化方法和基于优化等)之间的比较.

Dutta 等人借助于局部搜索及混合整数线性规划, 提出了计算网络输出集合上近似的迭代算法[343]. 首先, 随机选取输入集合的一个初始样本, 利用局部搜索的方法来估计神经网络输出的上下界. 局部搜索往往不能得到神经网络输出的全局上下界, 而是局部上下界. 基于搜索到的局部上下界构建一个混合整数线性规划, 此规划的一个可行解将改善此局部上下界. 如果此规划无解, 则算法终止, 返回的上下界为全局上下界; 如果有解, 再利用局部搜索的方法来估计神经网络输出的上下界, 迭代此过程.

Ruan 等人证明了大多数已知的 DNN 层都是 Lipschitz 连续的, 无论其层深、激活功能和神经元数量, 且提出一种基于全局优化的自适应嵌套优化算法来进行可达性分析, 解决神经网络的安全性和鲁棒性验证[344]. 通过神经网络的 Lipschitz 常数来计算一系列上下界, 最终收敛到神经网络输出集的极值.

利用抽象解释来验证深度神经网络的基本思想是: 利用合适的抽象域(比如 boxes、zonotopes 和 polyhedra)来上近似神经网络的输入集及其计算过程, 得到神经网络输出集的上近似, 并验证其是否满足给定性质. Li 等人[345]利用符号传播技术提高了基于抽象解释的 DNN 验证的精度, 利用神经网络中大量存在的仿射变换, 通过将节点值表示为符号变量的仿射表达式, 通过线性变换传播的方式来避免了直接在隐藏层进行区间计算所产生的误差; 同时, 更精确的数值上下界也会减轻 ReLU 激活函数和 max-pooling 层的计算量.

对基于精确可达性分析的方法, 如何高效计算出神经网络的输出集, 则是验证问题的核心, 比如利用面-顶点关联矩阵(FVIM)来表示凸集组合结构的完整编码[346]. FVIMs 结构对于集合操作具有非常有用的性质. 可以直接利用集合的顶点来确定输入集合是否跨越了 ReLU 函数在神经元中的负输入和正输入范围, 从而避免使用线性规划进行判断, 更高效地解决问题. 当输入集跨越了 ReLU 函数的两个输入范围时, 也可以将其快速划分为两个子集. 该方法的另一个特性是支持从输出到输入的回溯, 允许计算导致安全性质被违反的完整输入集.

针对一个黑盒神经网络, Xue 等人基于可能近似正确学习的理论(PAC learning)以及场景优化(scenario optimization), 提出了计算“黑盒”系统(包括大规模非线性深度神经网络)安全特征输入集下近似的线性规划方法[347].

当神经网络仅作为系统中的一部分时, 例如一个信息物理融合系统包含了复杂的机器学习组件, 整个系统的安全性验证问题则更加复杂. Dreossi 等人提出一个复合框架[348]来解决 CPSML 证伪问题. 通过时序逻辑伪造器和机器学习分析器共同收集信息, 合作搜索闭环系统中违反目标特性的行为和 ML 组件的输入空间, 找到系统中不满足 STL 的执行路径. 通过结合系统正确性的概念、证伪技术和 STL 性质的检验, Dreossi 等人构造了 VerifAI 工具链[349], 并将其应用于基于神经网络的飞机滑行系统的找错和改善[350].

ReachNN[351]工具考虑基于神经网络的控制器的验证问题. 该工具首先从神经网络计算出一个 Bernstein

多项式的近似, 类似于知识蒸馏(knowledge distillation)的技术, 并通过基于 Lipschitz 连续性质的理论计算和自适应采样技术得到近似的误差估计. 从这个多项式近似, 进而可以计算整个控制系统可达集的上近似. 与之前的方法不同, 这种方法可以处理多种激活函数, 并具备更好的可扩展性和计算效率.

11 总 结

如何设计安全可靠的计算机系统, 是计算机科学的重大挑战. 尤其近年来智能系统在计算机领域的广泛应用, 给系统的可靠性带来更大的挑战. 具有严格数学基础的形式化方法已被公认为设计可信系统的有效方法, 在工业领域中得到越来越广泛地应用. 然而, 与传统的仿真和测试相比, 形式化方法对使用者的要求依然较高; 另一方面, 智能系统的广泛应用也带来新的形式化问题. 本文从系统的可信需求出发, 对系统进行不同维度的分类, 并调研系统设计中所用的主要形式化方法. 我们分别根据系统的特征和应用场景, 将系统进行分类. 针对每一类系统和场景, 从行为建模、性质描述、验证方法和工具等方面分别介绍有关的形式化方法. 本文最终的目的是对系统的行为和性质建立一个形式化的分类和技术框架, 以支撑包括智能系统在内的可信系统的设计.

致谢 感谢成文过程中, 詹乃军研究员的指导.

References:

[1] Becker S, Boskovic M, Dhama A, Giesecke S, Happe J, Hasselbring W, Koziolek H, Lipskoch H, Meyer R, Muhle M, Paul A, Ploski J, Rohr M, Swaminathan M, Warns T, Winteler D. Trustworthy software systems: A discussion of basic concepts and terminology. ACM SIGSOFT Software Engineering Notes, 2006, 31(6): 1−18.

[2] Woodcock J, Larsen PG, Bicarregui J, Fitzgerald J. Formal methods: Practice and experience. ACM Computing Surveys, 2009, 41(4): Article No.19.

[3] Gleirshcer M, Foster S, Woodcock J. New opportunities for integrated formal methods. ACM Computing Surveys, 2020, 52(6): Article No.117.

[4] Baier C, Katoen J. Principles of Model Checking. MIT Press, 2008.

[5] Clarke EM, Henzinger TA, Veith H. Handbook of Model Checking. Springer, 2018.

[6] Clarke EM, Grumberg O, Kroening D, *et al*. Model Checking. MIT Press, 2018.

[7] Robinson A, Voronkov A. Handbook of Automated Reasoning. Elsevier, 2001.

[8] Harrison J. Handbook of Practical Logic and Automated Reasoning. Cambridge University Press, 2009.

[9] Burch JR, Clarke EM, McMillan KL, Dill DL, Hwang LJ. Symbolic model checking: 10^{20} states and beyond. Information and Computation, 1992, 98(2): 142−170.

[10] Biere A, Cimatti A, Clarke EM, Zhu Y. Symbolic model checking without BDDs. In: Proc. of the TACAS. 1999. 193−207.

[11] Clarke EM, Grumberg O, Jha S, Lu Y, Veith H. Counterexample-guided abstraction refinement for symbolic model checking. Journal of the ACM (JACM), 2003, 50(5): 752−794.

[12] Wang J, Zhan NJ, Feng XY, Liu ZM. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 33−61 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[13] Bu L, Chen LQ, Chen Z, *et al*. Development and trends of research in formal methods. In: China Computer Federation Proc. 2018. 1−68 (in Chinese).

[14] Peled DA. Software Reliability Methods. Springer, 2001.

[15] Nielson F, Nielson HR. Formal Methods, An Appetizer. Springer, 2019.

[16] Zhang J, Zhang C, Xuan JF, Xiong YF, Wang QX, Liang B, Li L, Dou WS, Chen ZB, Chen LQ, Cai Y. Recent progress in program analysis. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 80−109 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

[17] Nielson F, Nielson HR, Hankin C. Principles of Program Analysis. Springer, 1999.

[18] Plotkin G. A structural approach to operational semantics. Journal of Logic and Algebraic Programming, 1981, 60: 17−139.

[19] Scott DS. Outline of a mathematical theory of computation. 1970. https://www.researchgate.net/publication/242406560

[20] Floyd RW. Assigning meanings to programs. Proc. of Symposia in Applied Mathematics, 1967, 19: 19−32.

[21] Hoare CAR. An axiomatic basis for computer programming. Communications of the ACM, 1969, 12(10): 576−580.
[22] Dijkstra EW. A Discipline of Programming. Prentice Hall, 1976.
[23] Apt KR, Olderog ER. Fifty years of Hoare's logic. Formal Aspects of Computing, 2019, 31: 751−807.
[24] Reynolds JC. Separation logic: A logic for shared mutable data structures. In: Proc. of the LICS. 2002. 55−74.
[25] Chin WN, David C, Nguyen H, Qin S. Enhancing modular OO verification with separation logic. In: Proc. of the POPL. 2008. 87−99.
[26] Parkinson M, Bierman GM. Separation logic, abstraction and inheritance. In: Proc. of the POPL. 2008. 75−86.
[27] Bjorner N, Browne A, Manna Z. Automatic generation of invariants and intermediate assertions. Theoretical Computer Science, 1997, 173(1): 49−87.
[28] Cook B, Podelski A, Rybalchenko A. Proving program termination. Communications of the ACM, 2011, 54(5): 88−98.
[29] Yang L, Zhou CC, Zhan NJ, *et al*. Recent advances in program verification through computer algebra. Frontiers of Computer Science, 2010, 4(1): 1−16.
[30] McMillan KL. Interpolation and SAT-based model checking. In: Proc. of the CAV. 2003. 1−13.
[31] Garg P, Loding C, Madhusudan P, Neider D. ICE: A robust framework for learning invariants. In: Proc. of the CAV. 2014. 69−87.
[32] Si X, Dai H, Raghothaman M, Naik M, Song L. Learning loop invariants for program verification. In: Advances in Neural Information Processing Systems. 2018. 7751−7762.
[33] Yao J, Ryan G, Wong J, Jana S, Gu R. Learning nonlinear loop invariants with gated continuous logic networks. In: Proc. of the PLDI. 2020. 106−120.
[34] Podelski A, Rybalchenko A. A complete method for the synthesis of linear ranking functions. In: Proc. of the VMCAI. 2004. 239−251.
[35] Heizmann M, Hoenicke J, Podelski A. Termination analysis by learning terminating programs. In: Proc. of the CAV. 2014. 797−813.
[36] Chen Y, Heizmann M, Lengál O, Li Y, Tsai M, Turrini A, Zhang L. Advanced automata-based algorithms for program termination checking. In: Proc. of the PLDI. 2018. 135−150.
[37] Jhala R, Majumdar R. Software model checking. ACM Computing Surveys, 2009, 41(4): Article No.21.
[38] Alur R, Benedikt M, Etessami K, Godefroid P, Reps TW, Yannakakis M. Analysis of recursive state machines. ACM Trans. on Programming Languages and Systems, 2005, 27(4): 786−818.
[39] Bouajjani A, Esparza J, Maler O. Reachability analysis of pushdown automata: Application to model checking. In: Proc. of the CONCUR. 1997. 135−150.
[40] Esparza J, Hansel D, Rossmanith P, Schwoon S. Efficient algorithms for model checking pushdown systems. In: Proc. of the CAV. 2000. 232−247.
[41] Alur R, Madhusudan P. Adding nesting structure to words. Journal of the ACM (JACM), 2009, 56(3): Article No.16.
[42] Jacobs B, Smans J, Philippaerts P, Vogels F, Penninckx W, Piessens F. VeriFast: A powerful, sound, predictable, fast verifier for C and Java. In: Proc. of the NASA Formal Methods. 2011. 41−55.
[43] Rustan K, Leino M. Dafny: An automatic program verifier for functional correctness. In: Proc. of the LPAR. 2010. 348−370.
[44] Filliatre JC, Paskevich A. Why3—Where programs meet provers. In: Proc. of the ESOP. 2013. 125−128.
[45] Kirchner F, Kosmatov N, Prevosto V, Signoles J, Yakobowski B. Frama-C: A software analysis perspective. Formal Aspects of Computing, 2015, 27(3): 573−609.
[46] Ball T, Levin V, Rajamani SK. A decade of software model checking with SLAM. Communications of the ACM, 2011, 54(7): 68−76.
[47] O'Hearn PW. Continuous reasoning: Scaling the impact of formal methods. In: Proc. of the LICS. 2018. 13−25.
[48] Keller R. Formal verification of parallel programs. Communications of the ACM, 1976, 19(7): 371−384.
[49] Kropf T. Introduction to Formal Hardware Verification. Springer, 1999.
[50] Büchi, JR. On a decision method in restricted second order arithmetic. In: Proc. of the Int'l Congress on Logic, Method, and Philosophy of Science. 1962. 1−12.
[51] McNaughton R. Testing and generating infinite sequences by a finite automaton. Information and Control, 1966, 9: 521−530.
[52] Rabin MO. Decidability of second order theories and automata on infinite trees. Trans. of the American Mathematical Society, 1969, 141: 1−35.
[53] Vardi MY, Wolper P. An automata-theoretic approach to automatic program verification (preliminary report). In: Proc. of the LICS. 1986. 332−344.

[54] Pnueli A. The temporal logic of programs. In: Proc. of the FOCS. 1977. 46−57.
[55] Clarke EM, Emerson EA. Design and synthesis of synchronization skeletons using branching-time temporal logic. Logics of Programs, 1981, 131: 52−71.
[56] Kupferman O, Vardi MY. Model checking of safety properties. Formal Methods in System Design, 2001, 19(3): 291−314.
[57] Courcoubetis C, Vardi MY, Wolper P, Yannakakis M. Memory-efficient algorithms for the verification of temporal properties. Formal Methods in System Design, 1992, 1(2-3): 275−288.
[58] Ben-Ari M, Pnueli A, Manna Z. The temporal logic of branching time. Acta Informatica, 1983, 20: 207−226.
[59] Queille JP, Sifakis J. Specification and verification of concurrent systems in CESAR. In: Proc. of the Int'l Symp. on Programming, Vol.137. 1982. 337−351.
[60] Clarke EM, Emerson EA, Sistla AP. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. on Programming Languages and Systems, 1986, 8(2): 244−263.
[61] Iwashita H, Nakata T, Hirose F. CTL model checking based on forward state traversal. In: Proc. of the ICCAD. 1996. 82−87.
[62] Emerson EA, Halpern JY. "Sometimes" and "not never" revisited: On branching versus linear-time temporal logic. Journal of the ACM (JACM), 1986, 33(1): 151−178.
[63] Janin D, Walukiewicz I. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In: Proc. of the Int'l Conf. on Concurrency Theory. 1996. 263−277.
[64] Thomas W. Infinite games and verification. In: Proc. of the Int'l Conf. on Computer Aided Verification. 2002. 58−65.
[65] Kozen D. Results on the propositional mu-calculus. Theoretical Computer Science, 1983, 27(3): 333−354.
[66] Kozen D. A finite model theorem for the propositional mu-calculus. Studia Logica, 1988, 47(3): 233−241.
[67] Berwanger D, Gradel E, Lenzi G. The variable hierarchy of the mu-calculus is strict. Theory of Computing Systems, 2007, 40(4): 437−466.
[68] McNaughton R. Infinite games played on finite graphs. Annals of Pure and Applied Logic, 1993, 65(2): 149−184.
[69] Jurdziński M. Small progress measures for solving parity games. In: Proc. of the Annual Symp. on Theoretical Aspects of Computer Science. 2000. 290−301.
[70] Schewe S. Solving parity games in big steps. In: Proc. of the Int'l Conf. on Foundations of Software Technology and Theoretical Computer Science. 2007. 449−460.
[71] Lamport L. Proving the correctness of multiprocess programs. IEEE Trans. on Software Engineering, 1977, 3(2): 125−143.
[72] Alpern B, Schneider FB. Defining liveness. Information Processing Letters, 1985, 21(4): 181−185.
[73] Manolios P, Trefler RJ. Safety and liveness in branching time. In: Proc. of the LICS. 2001. 366−374.
[74] Manna Z, Pnueli A. A hierarchy of temporal properties. In: Proc. of the ACM Symp. on Principles of Distributed Computing. 1990. 377−410.
[75] Cimatti A, Clarke EM, Giunchiglia F, Roveri M. NuSMV: A new symbolic model checker. Int'l Journal on Software Tools for Technology Transfer, 2000, 2(4): 410−425.
[76] Holzmann G. The model checker SPIN. IEEE Trans. on Software Engineering, 1997, 23(5): 279−295.
[77] Dill D. The Murphi verification system. In: Proc. of the CAV. 1996. 390−393.
[78] Lamport L. Specifying Systems. Boston: Addison-Wesley, 2002.
[79] Havelund K, Shankar N. Experiments in theorem proving and model checking for protocol verification. In: Proc. of the IBAFM. 1996. 662−681.
[80] Biere A, Kick A. A case study on different modeling approaches based on model checking-verifying numerous versions of the alternating bit protocol with SMV. 1995. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.7109&rep=rep1&type=pdf
[81] McMillan KL, Schwalbe J. Formal verification of the encore Gigamaxcache consistency protocol. In: Proc. of the Int'l Symp. on Shared Memory Multiprocessors. 1991. 242−251.
[82] Clarke EM, Grumberg O, Hiraishi H, Jha S, Long DE, McMillan KL, Ness LA. Verification of the futurebus+ cache coherence protocol. Formal Methods in System Design, 1995, 6(2): 217−232.
[83] Moon I. Modeling programmable logic controllers for logic verification. IEEE Control Systems, 1994, 14(2): 53−59.
[84] Barrett G. Model checking in practice: The T9000 virtual channel processor. IEEE Trans. on Software Engineering, 1995, 21(2): 69−78.
[85] Anderson RJ, Beame P, Burns S, Chan W, Modugno F, Notkin D, Reese JD. Model checking large software specifications. In: Proc. of the SIGSOFT FSE. 1996. 156−166.
[86] Holzmann GJ. Protocol design: Redefining the state of the art. IEEE Software, 1992, 9(1): 17−22.

[87] Ruane LM. Process synchronization in the UTS kernel. Computer Systems, 1990, 3(3): 387−421.
[88] D'Argenio PR, Katoen JP, Ruys T, Tretmans J. Modeling and verifying a bounded retransmission protocol. In: Proc. of the Workshop Applied Formal Methods in System Design. 1996. 114−127.
[89] Jensen HE, Larsen KG, Skou A. Modelling and analysis of a collision avoidance protocol using SPIN and UPPAAL. In: Proc. of the Spin Verification System. 1996. 261−277.
[90] Ruys T, Langerak R. Validation of Bosch' mobile communication network architecture with SPIN. In: Proc. of the 3rd SPIN Workshop. 1997. 75−89.
[91] Dill DL, Park S, Nowatzyk AG. Formal specification of abstract memory models. In: Proc. of the Symp. on Research on Integrated Systems. 1993. 38−52.
[92] Dill DL, Drexler AJ, Hu AJ, Yang CH. Protocol verification as a hardware design aid. In: Proc. of the IEEE Int'l Conf. on Computer Design. 1992. 522−525.
[93] Lenoski D, Laudon J, Gharachorloo K, Weber WD, Gupta A, Hennessy J, Horowitz M, Lam M. The Stanford DASH multiprocessor. Computer, 1992, 25(3): 63−79.
[94] Mitchell JC, Mitchell M, Stern U. Automated analysis of cryptographic protocols using Mur/spl phi/. In: Proc. of the IEEE Symp. on Security and Privacy. 1997. 141−151.
[95] Needham RM, Schroeder MD. Using encryption for authentication in large networks of computers. Communications of the ACM, 1978, 21(12): 993−999.
[96] Tatebayashi M, Matsuaaki N, Newman D. Key distribution protocol for digital mobile communication systems. In: Proc. of the CRYPTO. 1990. 324−333.
[97] Lamport L. The temporal logic of actions. ACM Trans. on Programming Languages and Systems (TOPLAS), 1994, 16(3): 872−923.
[98] Beers R. Pre-RTL formal verification: An Intel experience. In: Proc. of the DAC. 2008. 806−811.
[99] Newcombe C, Rath T, Zhang F, Munteamu B, Brooker M, Deardeuff M. How Amazon web services uses formal methods. Communications of the ACM, 2015, 58(4): 66−73.
[100] Milner R. A Calculus of Communicating Systems. New York: Springer, 1980.
[101] Hennessy MCB, Milner R. Algebraic laws for nondeterminism and concurrency. Journal of the ACM (JACM), 1985, 32(1): 137−161.
[102] Victor B. The mobility workbench user's guide: Polyadic version 3.122. Department of Information Technology, Uppsala University, 1995. https://www.it.uu.se/profundis/mwb-dist/guide-3.122.pdf
[103] Victor B. A verification tool for the polyadic pi-calculus [MS. Thesis]. Department of Computer Systems, Uppsala University, 1994.
[104] Hoare CAR. Communicating Sequential Processes. Englewood: Prentice-Hall, 1985.
[105] Brookes SD. On the relationship of CCS and CSP. In: Proc. of the ICALP. 1983. 83−96.
[106] Bundgaard M, Milner R. Unfolding CSP. In: Proc. of the Reflections on the Work of C.A.R. Hoare. 2010. 213−228.
[107] Roscoe AW. Understanding concurrent systems. New York: Springer, 2010.
[108] Schneider SA, Davies J. Timed CSP: Theory and practice. In: Proc. of the REX Workshop. 1991. 640−675.
[109] Gibson-Robinson T. FDR3—A modern refinement checker for CSP. In: Proc. of the TACAS. 2014. 187−201.
[110] Armstrong P, Lowe G, Ouaknine J, Roscoe AW. Model Checking Timed CSP. In: HOWARD-60. A Festschrift on the Occasion of Howard Barringer's 60th Birthday, Vol.42. 2014. 2014. 13−33.
[111] Ouaknine J. Discrete analysis of continuous behaviour in real-time concurrent systems. Technical Report, University of Oxford, 2000. https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.365293
[112] Donovan B, Norris P, Lowe G. Analyzing a library of security protocols using Casper and FDR. In: Proc. of the Workshop on Formal Methods and Security Protocols. 1999. https://www.cs.ox.ac.uk/publications/publication676-abstract.html
[113] Brackin SH. Evaluating and improving protocol analysis by automatic proof. In: Proc. of the 11th IEEE Computer Security Foundations Workshop. 1998. [doi: 10.1109/CSFW.1998.683164]
[114] Peterson JL. Petri Net Theory and the Modeling of Systems. Englewood: Prentice-Hall, 1981.
[115] Murata T. Petri nets: Properties, analysis and applications. In: Proc. of the IEEE. 1989. 541−580.
[116] Abdulla PA. General decidability theorems for infinite-state systems. In: Proc. of the LICS. 1996. 313−321.
[117] Reisig W, Rozenberg G. Lectures on Petri Nets I: Basic Models: Advances in Petri Nets. New York: Springer, 1998.
[118] Kostin A. Using Transition Invariants for Reachability Analysis of Petri Nets. INTECH Open Access Publisher, 2008.

[119] Benjamin M, Bergenthum R. Travis—An online tool for the synthesis and analysis of Petri nets with final states. In: Proc. of the Int'l Conf. on Application and Theory of Petri Nets and Concurrency. 2017. 101−111.

[120] Hillah LM, Kordon F. Petri nets repository: A tool to benchmark and debug Petri net tools. In: Proc. of the Int'l Conf. on Application and Theory of Petri Nets and Concurrency. 2017. 125−135.

[121] Hu X, Li J, Li ZJ. Modelling and performance analysis of IEEE 802.11 DCF using coloured Petri nets. The Computer Journal, 2016, 59(10): 1563−1580.

[122] Berthelot G, Terrat R. Petri nets theory for the correctness of protocols. IEEE Trans. on Communications, 1982, 30(12): 2497−2505.

[123] PAT: Process analysis toolkit. 2021. https://pat.comp.nus.edu.sg

[124] Sun J, Liu Y, Dong JS, Pang J. PAT: Towards flexible verification under fairness. In: Proc. of the CAV. 2009. 709−714.

[125] Liu Y, Sun J, Dong JS. PAT 3: An extensible architecture for building multi-domain model checkers. In: Proc. of the ISSRE. 2011. 190−199.

[126] Fernando D, Dong N, Jégourel C, Dong JS. Verification of strong Nash-equilibrium for probabilistic BAR systems. In: Proc. of the Formal Engineering Methods. 2018. 106−123.

[127] Fernando D, Dong N, Jégourel C, Dong JS. Verification of Nash-equilibrium for probabilistic BAR systems. In: Proc. of the ICECCS. 2016. 53−62.

[128] Thin WYMM, Dong NP, Bai GD, Dong JS. Formal analysis of a proof-of-stake blockchain. In: Proc. of the 23rd Int'l Conf. on Engineering of Complex Computer Systems (ICECCS). 2018. 197−200.

[129] Li L, Sun J, Liu Y. A formal specification and verification framework for timed security protocols. IEEE Trans. on Software Engineering, 2017, 44(8): 725−746.

[130] Ling S, Liu S, Hao J. Towards solving decision making problems using probabilistic model checking. In: Proc. of the ICECCS. 2017. 150−153.

[131] Alur R, Dill D. Automata for modeling real-time systems. In: Proc. of the ICALP. 1990. 322−335.

[132] Alur R, Dill D. A theory of timed automata. Theoretical Computer Science, 1994, 126(2): 183−235.

[133] Cerans K. Decidability of bisimulation equivalences for parallel timer processes. In: Proc. of the CAV. 1992. 302−315.

[134] Konrad S, Cheng BHC. Real-time specification patterns. In: Proc. of the ICSE. 2005. 372−381.

[135] Bouyer P, Laroussinie F, Markey N, *et al*. Timed temporal logics. In: Proc. of the Models, Algorithms, Logics and Tools. 2017. 211−230.

[136] Alur R, Courcoubetis C, Dill D. Model-checking for real-time systems. In: Proc. of the LICS. 1990. 414−425.

[137] Alur R, Courcoubetis C, Dill D. Model-checking in dense real-time. Information and Computation, 1993, 104(1): 2−34.

[138] Bouyer P, Chevalier F, Markey N. On the expressiveness of TPTL and MTL. In: Proc. of the FSTTCS. 2005. 432−443.

[139] Alur R, Henzinger TA. A really temporal logic. Journal of the ACM (JACM), 1994, 41(1): 181−203.

[140] Larsen KG, Pettersson P, Yi W. UPPAAL in a nutshell. Int'l Journal on Software Tools for Technology Transfer, 1997, 1(1): 134−152.

[141] Bengtsson J, Yi W. Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets. 2003. 87−124.

[142] Havelund K, Larsen KG, Skou A. Formal verification of a power controller using the real-time model checker Uppaal. In: Proc. of the Int'l AMAST Workshop on Aspects of Real-time Systems and Concurrent and Distributed Software. Berlin, Heidelberg: Springer, 1999. 277−298.

[143] Larsen K G, Mikucionis M, Nielsen B, *et al*. Testing real-time embedded software using UPPAAL-TRON: An industrial case study. In: Proc. of the 5th ACM Int'l Conf. on Embedded Software. 2005. 299−306.

[144] Larsen KG, Lorber F, Nielsen B. 20 years of UPPAAL enabled industrial model-based validation and beyond. In: Proc. of the Int'l Symp. on Leveraging Applications of Formal Methods. Cham: Springer, 2018. 212−229.

[145] Yovine S. Kronos: A verification tool for real-time systems. Int'l Journal on Software Tools for Technology Transfer, 1997, 1(1−2): 123−133.

[146] Bozga M, Daws C, Maler O, *et al*. Kronos: A model-checking tool for real-time systems. In: Proc. of the Int'l Symp. on Formal Techniques in Real-time and Fault-tolerant Systems. Berlin, Heidelberg: Springer, 1998. 298−302.

[147] Wang F, Yao LW, Yang YL. Efficient verification of distributed real-time systems with broadcasting behaviors. Real-time Systems, 2011, 47(4): 285−318.

[148] Gupta V, Henzinger TA, Jagadeesan R. Robust timed automata. In: Proc. of the HART. 1997. 331−345.

[149] Bouyer P, Markey N, Sankur O. Robustness in timed automata. In: Proc. of the RP. 2013. 1−18.

[150] Katoen JP. The probabilistic model checking landscape. In: Proc. of the 31st Annual ACM/IEEE Symp. on Logic in Computer Science. 2016. 31−45.

[151] Kemeny JG, Snell JL. Finite Markov Chains. Princeton: D. Van Nostrand, 1960.

[152] Kulkarni VG. Modeling and Analysis of Stochastic Systems. New York: Chapman & Hall, 1995.

[153] Bellman R. A Markovian decision process. Journal of Mathematics and Mechanics, 1957, 6(5): 679−684.

[154] Vardi MY. Automatic verification of probabilistic concurrent finite state programs. In: Proc. of the 26th Annual Symp. on Foundations of Computer Science. 1985. 327−338.

[155] Anderson WJ. Continuous-time Markov Chains: An Applications-oriented Approach. New York: Springer Science & Business Media, 2012.

[156] Hart S, Sharir M, Pnueli A. Termination of probabilistic concurrent program. ACM Trans. on Programming Languages and Systems (TOPLAS), 1983, 5(3): 356−380.

[157] Vardi MY, Wolper P. Reasoning about infinite computations. Information and Computation, 1994, 115(1): 1−37.

[158] Klein J, Baier C. Experiments with deterministic ω-automata for formulas of linear temporal logic. Theoretical Computer Science, 2006, 363(2): 182−195.

[159] Hansson H, Jonsson B. A logic for reasoning about time and reliability. Formal Aspects of Computing, 1994, 6(5): 512−535.

[160] Aziz A, Singhal V, Balarin F, Brayton RK, Sangiovanni-Vincentelli AL. It usually works: The temporal logic of stochastic systems. In: Proc. of the Int'l Conf. on Computer Aided Verification. 1995. 155−165.

[161] Baier C, Katoen JP, Hermanns H, Wolf V. Comparative branching-time semantics for Markov chains. Information and Computation, 2005, 200(2): 149−214.

[162] Aziz A, Sanwal K, Singhal V, Brayton R. Model-checking continuous-time Markov chains. ACM Trans. on Computational Logic (TOCL), 2000, 1(1): 162−170.

[163] Baier C, Haverkort B, Hermanns H, Katoen JP. Model-checking algorithms for continuous-time Markov chains. IEEE Trans. on Software Engineering, 2003, 29(6): 524−541.

[164] Katoen JP, Song L, Zhang L. Probably safe or live. In: Proc. of the Joint Meeting of the 23rd EACSL Annual Conf. on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symp. on Logic in Computer Science (LICS). 2014. 1−10.

[165] Kwiatkowska M, Norman G, Parker D. PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. of the Int'l Conf. on Computer Aided Verification. 2011. 585−591.

[166] Baier C, Clarke EM, Hartonas-Garmhausen V, Kwiatkowska M, Ryan M. Symbolic model checking for probabilistic processes. In: Proc. of the Int'l Colloquium on Automata, Languages, and Programming. 1997. 430−440.

[167] Fruth M. Probabilistic model checking of contention resolution in the IEEE 802.15. 4 low-rate wireless personal area network protocol. In: Proc. of the 2nd Int'l Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006). IEEE, 2006. 290−297.

[168] Kwiatkowska M, Norman G, Parker D. Probabilistic verification of Herman's self-stabilisation algorithm. Formal Aspects of Computing, 2012, 24(4): 661−670.

[169] Barbot B, Kwiatkowska M. On quantitative modelling and verification of DNA walker circuits using stochastic Petri nets. In: Proc. of the Int'l Conf. on Applications and Theory of Petri Nets and Concurrency. Cham: Springer, 2015. 1−32.

[170] Dehnert C, Junges S, Katoen JP, *et al*. A storm is coming: A modern probabilistic model checker. In: Proc. of the Int'l Conf. on Computer Aided Verification. Cham: Springer, 2017. 592−600.

[171] Hahn EM, Li Y, Schewe S, *et al*. iscasMc: A Web-based probabilistic model checker. In: Proc. of the Int'l Symp. on Formal Methods. Cham: Springer, 2014. 312−317.

[172] Katoen JP, Khattri M, Zapreevt IS. A Markov reward model checker. In: Proc. of the 2nd Int'l Conf. on the Quantitative Evaluation of Systems (QEST 2005). 2005. 243−244.

[173] Baier C, Ciesinski F. Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In: Proc. of the 3rd Int'l Conf. on the Quantitative Evaluation of Systems (QEST 2006). 2006. 131−132.

[174] Bozzano M, Cimatti A, Katoen JP, *et al*. Spacecraft early design validation using formal methods. Reliability Engineering & System Safety, 2014, 132: 20−35.

[175] Esteve MA, Katoen JP, Nguyen VY, *et al*. Formal correctness, safety, dependability, and performance analysis of a satellite. In: Proc. of the 2012 34th Int'l Conf. on Software Engineering (ICSE). IEEE, 2012. 1022−1031.

[176] Kaminski BL, Katoen JP, Matheja C, Olmedo F. Weakest precondition reasoning for expected run-times of probabilistic programs. In: Proc. of the European Symp. on Programming. 2016. 364−389.

[177] McIver A, Morgan C, Kaminski BL, Katoen JP. A new proof rule for almost-sure termination. In: Proc. of the Symp. on Principles of Programming Languages (POPL 2018). 2018. 33:1–33:28.

[178] Gretz F, Katoen JP, McIver A. PRINSYS—On a quest for probabilistic loop invariants. In: Proc. of the 10th Int'l Conf. on Quantitative Evaluation of Systems (QEST 2013). 2013. 172–187.

[179] Ericson CA. Fault tree analysis—A history. In: Proc. of the 17th Int'l System Safety Conf. 1999. 1–9.

[180] Vesely WE, Goldberg FF, Roberts NH, Haasl DF. FaultTree Handbook. Washington: Nuclear Regulatory Commission, 1981.

[181] Raiteri DC, Franceschinis G, Iacono M, Vittorini V. Repairable fault tree for the automatic evaluation of repair policies. In: Proc. of the Dependable Systems and Networks. 2004. 659–668.

[182] Coudert O, Madre JC. Fault tree analysis: 1020 prime implicants and beyond. In: Proc. of the Reliability and Maintainability Symp. 1993. 240–245.

[183] Rauzy AB. New algorithms for fault tree analysis. Reliability Engineering and System Safety, 1993, 40(3): 203–211.

[184] Remenyte-Prescott R, Andrews J. An enhanced component connection method for conversion of fault trees to binary decision diagrams. Reliability Engineering and System Safety, 2008, 93(10): 1543–1550.

[185] Barlow RE, Proschan F. Statistical Theory of Reliability and Life Testing. New York: Holt, Rinehartand Winston, 1975.

[186] Rao KD, Gopika V. Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment. Reliability Engineering & System Safety, 2009, 94(4): 872–883.

[187] Montani S, Portinale L, Bobbio A, Varesio M, Codetta-Raiteri D. DBNet, a tool to convert dynamic fault trees into dynamic Bayesian networks. Technical Report, 2005. https://www.di.unipmn.it/en/publications-en/technical-reports-en.html

[188] Dugan JB, Bavuso SJ, Boyd MA. Fault trees and sequence dependencies. In: Proc. of the Annual Reliability and Maintainability Symp. IEEE, 1990. 286–293.

[189] Ruijters E, Stoelinga M. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. Computer Science Review, 2015, 15-16: 29–62.

[190] Tanaka H, Fan L, Lai F, Toguchi K. Fault-tree analysis by fuzzy probability. IEEE Trans. on Reliability, 1983, 32(5): 453–457.

[191] Bobbio A, Codetta-Raiteri D. Parametric fault trees with dynamic gates and repair boxes. In: Proc. of the Reliability and Maintainability Symp. (RAMS). IEEE, 2004. 459–465.

[192] Boudali H, Crouzen P, Stoelinga M. CORAL—A tool for compositional reliability and availability analysis. In: Proc. of the Workshop on the CAV. 2007.

[193] Codetta-Raiteri D. The conversion of dynamic fault trees to stochastic Petri nets, as a case of graph transformation. In: Proc. of the PNGT. 2005. 45–60.

[194] Boudali H, Nijmeijer AP, Stoelinga MIA. DFTSim: A simulation tool for extended dynamic fault trees. In: Proc. of the Spring Simulation Multiconference. 2009. 1–8.

[195] Hasan O, Ahmed W, Tahar S, Hamdi MS. Reliability block diagrams based analysis: A survey. AIP Conf. Proc., 2015, 1648(1). [doi: 10.1063/1.4913184]

[196] Distefano S, Puliafito A. Dynamic reliability block diagrams: Overview of a methodology. In: Proc. of the European Safety and Reliability. 2007. 1059–1068.

[197] Smith G. The Object-Z Specification Language. New York: Springer. 2012.

[198] Ahmed W, Hasan O, Tahar S. Formalization of reliability block diagrams in higher-order logic. Journal of Applied Logic, 2016, 18: 19–41.

[199] Yassmeen E, Osman H. A formally verified algebraic approach for dynamic reliability block diagrams. In: Proc. of the ICFEM. 2019. 253–269.

[200] HOL4. 2019. https://hol-theorem-prover.org/

[201] Henzinger TA. The theory of hybrid automata. In: Proc. of the Logic in Computer Science. 1996. 278–292.

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

[203] He J. From CSP to hybrid systems. In: Proc. of the a Classical Mind: Essays in Honour of C.A.R. Hoare. 1994. 171–189.

[204] Zhou C, Wang J, Ravn A. A formal description of hybrid systems. In: Proc. of the Hybrid Systems. 1995. 511–530.

[205] MathWorks. Simulink user's guide. 2018. http://www.mathworks.com/help/pdf_doc/simulink/sl_using.pdf

[206] Modelica Association. Modelica—A Unified Object-oriented Language for Systems Modeling—Version 3.3 Revision 1. Standard Specification, 2014.

[207] Koymans R. Specifying real-time properties with metric temporal logic. Real Time System, 1990, 2(4): 255–299.

[208] Alur R, Feder T, Henzinger T. The benefits of relaxing punctuality. Journal of the ACM (JACM), 1996, 43(1): 116–146.

[209] Maler O, Dickovic D. Monitoring temporal properties of continuous signals. In: Proc. of the FORMATS. 2004. 152−166.

[210] Maler O, Dickovic D, Pnueli A. Checking temporal properties of discrete, timed and continuous behaviors. Pillars of Computer Science, 2008. 475−505.

[211] Fainekos GE, Pappas GJ. Robustness of temporal logic specifications for continuous-time signals. Theoretical Computer Science, 2009, 410(42): 4262−4291.

[212] Donzé A, Maler O. Robust satisfaction of temporal logic over real-valued signals. In: Proc. of the FORMATS. 2010. 92−106.

[213] Roehm H, Oehlerking J, Heinz T, Althoff M. STL model checking of continuous and hybrid systems. In: Proc. of the ATVA. 2016. 412−427.

[214] Donzé A, Ferrère T, Maler O. Efficient robust monitoring for STL. In: Proc. of the CAV. 2013. 264−279.

[215] Deshmukh JV, Donzé A, Ghosh S, Jin X, Juniwal G, Seshia SA. Robust online monitoring of signal temporal logic. Formal Methods System Design, 2017, 51(1): 5−30.

[216] Makino K, Berz M. Rigorous integration of flows and ODEs using Taylor models. In: Proc. of the SNC. 2009. 79−84.

[217] Chen X, Ábrahám E, Sankaranarayanan S. Taylor model flowpipe construction for non-linear hybrid systems. In: Proc. of the RTSS. 2012. 183−192.

[218] Nedialkov NS, Jackson KR, Corliss GF. Validated solutions of initial value problems for ordinary differential equations. Applied Mathematics and Computation, 1999, 105(1): 21−68.

[219] Althoff M, Stursberg O, Buss M. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In: Proc. of the CDC. 2008. 4042−4048.

[220] Henrion D, Korda M. Convex computation of the region of attraction of polynomial control systems. IEEE Trans. on Automatic Control, 2014, 59(2): 297−312.

[221] Fan C, Qi B, Mitra S, Viswanathan M. DryVR: Data-driven verification and compositional reasoning for automotive systems. In: Proc. of the CAV. 2017. 441−461.

[222] Xue B, Zhang M, Easwaran A, Li Q. PAC model checking of black-box continuous-time dynamical systems. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2020. 39(11): 3944−3955.

[223] Goubault E, Mullier O, Putot S, Kieffer M. Inner approximated reachability analysis. In: Proc. of the HSCC. 2014. 163−172.

[224] Goubault E, Putot S. Forward inner-approximated reachability of non-linear continuous systems. In Proc. of the HSCC. 2017. 1−10.

[225] Xue B, She Z, Easwaran A. Under-approximating backward reachable sets by polytopes. In: Proc. of the CAV. 2016. 457−476.

[226] Xue B, Fränzle M, Zhan N. Inner-approximating reachable sets for polynomial systems with time-varying uncertainties. IEEE Trans. on Automatic Control, 2020, 65(4): 1468−1483.

[227] Xue B, Zhan N, Fränzle M. Inner-approximating reach-avoid sets for discrete-time polynomial systems. In: Proc. of the CDC. 2020. 867−873.

[228] Frehse G, Le Guernic C, Donzé A, Cotton S, Ray R, Lebeltel O, Ripado R, Girard A, Dang T, Maler O. SpaceEx: Scalable verification of hybrid systems. In: Proc. of the CAV. 2011. 379−395.

[229] Chen X, Ábrahám E, Sankaranarayanan S. Flow*: An analyzer for non-linear hybrid systems. In: Proc. of the CAV. 2013. 258−263.

[230] Bogomolov S, Forets M, Frehse G, Potomkin K, Schilling C. JuliaReach: A toolbox for set-based reachability. In: Proc. of the HSCC. 2019. 39−44.

[231] Althoff M. An introduction to CORA 2015. In: Proc. of the ARCH@CPSWeek. 2015. 120−151.

[232] Bu L, Li Y, Wang L, Chen X, Li X. BACH 2: Bounded reachability checker for compositional linear hybrid systems. In: Proc. of the DATE. 2010. 1512−1517.

[233] Fan C, Qi B, Mitra S, Viswanathan M, Duggirala PS. Automatic reachability analysis for nonlinear hybrid models with C2E2. In: Proc. of the CAV. 2016. 531−538.

[234] Donzé A. Breach, a toolbox for verification andparameter synthesis of hybrid systems. In: Proc. of the CAV. 2010.167−170.

[235] Fulton N, Mitsch S, Quesel J, Völp M, Platzer A. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Proc. of the CADE. 2015. 527−538.

[236] Wang S, Zhan N, Zou L. An improved HHL prover: An interactive theorem prover for hybrid systems. In: Proc. of the ICFEM. 2015. 382−399.

[237] Frehse G. An introduction to SpaceEx. 2020. http://spaceex.imag.fr/sites/default/files/introduction_to_spaceex_0.pdf

[238] Johnson T, Green J, Mitra S, Dudley R, Erwin RS. Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems. In: Proc. of the FM. 2012. 252−266.

[239] Minopoli S, Frehse G. SL2SX translator: From Simulink to SpaceEx models. In: Proc. of the HSCC. 2016. 93−98.

[240] Duggirala PS, Fan C, Mitra S, Viswanathan M. Meeting a powertrain verification challenge. In: Proc. of the CAV. 2015. 536−543.

[241] Platzer A, Quesel JD. Logical verification and systematic parametric analysis in train control. In: Proc. of the HSCC. 2008. 646−649.

[242] Platzer A, Quesel JD. European train control system: A case study in formal verification. In: Proc. of the ICFEM. 2009. 246−265.

[243] Platzer A. Verification of cyberphysical transportation systems. IEEE Intelligent Systems, 2009, 24(4): 10−13.

[244] Loos S, Renshaw DW, Platzer A. Formal verification of distributed aircraft controllers. In: Proc. of the HSCC. 2013. 125−130.

[245] Kouskoulas Y, Renshaw D, Platzer A. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In: Proc. of the HSCC. 2013. 263−272.

[246] Mitsch S, Ghorbal K, Vogelbacher D. Formal verification of obstacle avoidance and navigation of ground robots. CoRR abs/1605.00604, 2016.

[247] Herber P, Adelt J, Liebrenz T. Formal verification of intelligent cyber-physical systems with the interactive theorem prover KeYmaera X. In: Proc. of the Software Engineering (Satellite Events). 2021.

[248] Staroletov S. Automatic proving of stability of the cyber-physical systems in the sense of Lyapunov with KeYmaera. In: Proc. of the FRUCT. 2021. 431−438.

[249] Zhao H, Yang M, Zhan N, Gu B, Zou L, Chen Y. Formal verification of a descent guidance control program of a lunar lander. In: Proc. of the FM. 2014. 733−748.

[250] Ahmad E, Dong YW, Larson BR, Lü JD, Tang T, Zhan NJ. Behavior modeling and verification of movement authority scenario of Chinese train control system using AADL. Science China Information Sciences, 2015, 58(11): 1−20.

[251] Bradley A. SAT-based model checking without unrolling. In: Proc. of the VMCAI. 2011. 70−87.

[252] Eén N, Mishchenko A, Brayton R. Efficient implementation of property directed reachability. In: Proc. of the FMCAD. 2011. 125−134.

[253] Biere A, Heule M, van Maaren H, Walsh T. Handbook of Satisfiability. IOS Press, 2009.

[254] Biere A, Heljanko K, Wieringa S. AIGER 1.9 and beyond. Technical Report, 11/2, Institute for Formal Models and Verification, Johannes Kepler University, 2011.

[255] Niemetz A, Preiner M, Wolf C, Biere A. Btor2, BtorMC and Boolector 3.0. In: Proc. of the CAV. 2018. 587−595.

[256] IEEE Standard for Property Specification Language (PSL), Annex B. IEEE Std 1850™-2010, 2010.

[257] IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language, Annex F. IEEE Std 1800™-2009, 2009.

[258] Sheeran M, Singh S, Stålmarck G. Checking safety properties using induction and a SAT-solver. In: Proc. of the FMCAD. 2000. 108−125.

[259] Cimatti A, Griggio A. Software model checking via IC3. In: Proc. of the CAV. 2012. 277−293.

[260] Hoder K, Bjørner N. Generalized property directed reachability. In: Proc. of the SAT. 2012. 157−171.

[261] Cimatti A, Griggio A, Mover S, Tonetta S. Infinite-state invariant checking with IC3 and predicate abstraction. Formal Methods in System Design, 2016, 49(3): 190−218.

[262] Goel A, Sakallah K. Model checking of verilog RTL using IC3 with syntax-guided abstraction. In: Proc. of the NFM. 2019. 166−185.

[263] Li J, Zhu S, Zhang Y, Pu G, Vardi M. Safety model checking with complementary approximations. In: Proc. of the ICCAD. 2017. 95−100.

[264] Zhang H, Gupta A, Malik S. Syntax-guided synthesis for lemma generation in hardware model checking. In: Proc. of the VMCAI. 2021. 325−349.

[265] Seger CJ, Bryant R. Formal verification by symbolic evaluation of partially-ordered trajectories. Formal Methods in System Design, 1995, 6(2): 147−189.

[266] Melham T. Higher order logic and hardware verification. Cambridge Tracts in Theoretical Computer Science 31. Cambridge University Press, 1993.

[267] Goel A, Sakallah K. AVR: Abstractly verifying reachability. In: Proc. of the TACAS. 2020. 413−422.

[268] Cavada R, Cimatti A, Dorigatti M, Griggio A, Mariotti A, Micheli A, Mover S, Roveri M, Tonetta S. The nuXmv symbolic model checker. In: Proc. of the CAV. 2014. 334−342.

[269] Mann M, Irfan A, Lonsing F, Yang Y, Zhang H, Brown K, Gupta A, Barrett C. Pono: A flexible and extensible SMT-based model checker. In: Proc. of the CAV. 2021. 461−474.

[270] Cimatti A, Clarke E, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, Sebastiani R, Tacchella A. NuSMV 2: An open source tool for symbolic model checking. In: Proc. of the CAV. 2002. 359−364.

[271] Brayton R, Mishchenko A. ABC: an academic industrial-strength verification tool. In: Proc. of the CAV. 2010. 24−40.

[272] Seger CJH, Jones RB, O'Leary JW, Melham T, Aagaard MD, Barrett C, Syme D. An industrially effective environment for formal hardware verification. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2005, 24(9): 1381−1405.

[273] O'Leary JW, Kaivola R, Melham T. Relational STE and theorem proving for formal verification of industrial circuit designs. In: Proc. of the FMCAD. 2013. 97−104.

[274] Deursen TV. Introduction to security protocols. 2011. https://satoss.uni.lu/members/patrick/1stsxxm.pdf

[275] Dolev D, Yao ACC. On the security of public key protocols. In: Proc. of the FOCS. 1981. 350−357.

[276] LaMacchia B, Lauter K, Mityagin A. Stronger security of authenticated key exchange. In: Proc. of the Provable Security. 2007. 1−16.

[277] Lowe G. A hierarchy of authentication specifications. In: Proc. of the CSFW. 1997. 31−44.

[278] Lowe G. Casper: A compiler for the analysis of security protocols. Journal of Computer Security, 1998, 6(1-2): 53−84.

[279] Lowe G. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. Software Concepts Tools, 1996, 17(3): 93−102.

[280] Schmidt B, Meier S, Cremers C, Basin D. Automated analysis of Diffie-Hellman protocols and advanced security properties. In: Proc. of the CSF. 2012. 78−94.

[281] Meier S, Schmidt B, Cremers C, Basin D. The TAMARIN prover for the symbolic analysis of security protocols. In: Proc. of the CAV. 2013. 696−701.

[282] Blanchet B. An efficient cryptographic protocol verifier based on Prolog rules. In: Proc. of the CSFW. 2001. 82−96.

[283] Chadha R, Cheval V, Ciobaca S, Kremer S. Automated verification of equivalence properties of cryptographic protocols. Trans. on Computational Logic, 2016, 17(4): Article 23.

[284] Cremers C, Horvat M, Hoyland J, Scott S, Merwe T. A comprehensive symbolic analysis of TLS 1.3. In: Proc. of the CCS. 2017. 1773−1788.

[285] Basin D, Dreier J, Hirschi L, Radomirovic S, Sasse R, Stettler V. A formal analysis of 5G authentication. In: Proc. of the CCS. 2018. 1383−1396.

[286] Kunnemann R, Steel G. YubiSecure? Formal security analysis results for theYubiKey and YubiHSM. In: Proc. of the STM. 2012. 257−272.

[287] Abadi M, Blanchet B, Fournet C. The applied pi calculus: Mobile values, new names, and secure communication. Journal of the ACM (JACM), 2018, 65(1): Article No.1.

[288] Woo T, Lam S. A semantic model for authentication protocols. In: Proc. of the IEEE Symp. on Security and Privacy. 1993. 178−194.

[289] Bruno B. Automatic verification of correspondences for security protocols. Journal of Computer Security, 2009, 17(4): 363−434.

[290] Bruno B, Martín A, Cédric F. Automated verification of selected equivalences for security protocols. The Journal of Logic and Algebraic Programming, 2008, 75(1): 3−51.

[291] Abadi M, Blanchet B, Fournet C. Just fast keying in the pi calculus. ACM TISSEC, 2007, 10(3): 1−59.

[292] Aiello W, Bellovin SM, Blaze M, Canetti R, Ioannidis J, Keromytis K, Reingold O. Just fast keying: Key agreement in a hostile Internet. ACM TISSEC, 2004, 7(2): 242−273.

[293] Abadi M, Blanchet B. Computer-assisted verification of a protocol for certified email. Science of Computer Programming, 2005, 58(1-2): 3−27.

[294] Delaune S, Kremer S, Ryan M. Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security, 2009, 17(4): 435−487.

[295] Backes M, Hritcu C, Maffei M. Automated verification of remote electronic voting protocols in the applied pi-calculus. In: Proc. of the CSF. 2008. 195−209.

[296] Kremer S, Ryan M. Analysis of an electronic voting protocol in the applied pi calculus. In: Proc. of the ESOP. 2005. 186−200.

[297] Barthe G, Dupressoir F, Grégoire B, Kunz C, Schmidt B, Strub PY. Easycrypt: A tutorial. In: Proc. of the FOSAD. 2013. 146−166.

[298] Barthe G, Grégoire B, Béguelin SZ. Formal certification of code-based cryptographic proofs. In: Proc. of the POPL. 2009. 90−101.

[299] Rushby J. Noninterference, transitivity, and channel-control security policies. Technical Report, CSL-92-02, SRI Int'l, 1992.

[300] Oheimb DV. Information flow control revisited: Noninfluence=Noninterference+nonleakage. In: Proc. of the ESORICS. 2004. 225−243.

[301] Peter Y, Ryan A. Mathematical models of computer security. In: Proc. of the FOSAD. 2000. 1−62.

[302] Focardi R, Gorrieri R. Classification of security properties (part I: Information flow). In: Proc. of the FOSAD. 2000. 331−396.

[303] Roscoe AW, Huang J. Checking noninterference in timed CSP. Formal Aspects of Computing, 2013, 25(1): 3−35.

[304] Volpano DM, Irvine CE, Smith G. A sound type system for secure flow analysis. Journal of Computer Security, 1996, 4(2/3): 167−188.

[305] Zhang DF, Askarov A, Myers AC. Language-based control and mitigation of timing channels. In: Proc. of the PLDI. 2012. 99−110.

[306] Rajani V, Garg D. On the expressiveness and semantics of information flow types. Journal of Computer Security, 2020, 28(1): 129−156.

[307] Zdancewic SA. Programming languages for information security [Ph.D. Thesis]. Cornell University, 2002.

[308] Eggert S, Schnoor H, Wilke T. Dynamic noninterference: Consistent policies, characterizations and verification. CoRR, abs/1208.5580, 2012.

[309] Eggert S, Schnoor H, Wilke T. Noninterference with local policies. In: Proc. of the MFCS. 2013. 337−348.

[310] Hadj-Alouane NB, Lafrance S, Lin F, Mullins J, Yeddes MM. On the verification of intransitive noninterference in multilevel security. IEEE Trans. on Systems, Man and Cybernetics, Part B, 2005, 35(5): 948−958.

[311] Eggert S, Meyden RVD, Schnoor H, Wilke T. The complexity of intransitive noninterference. In: Proc. of the IEEE Symp. on Security and Privacy. 2011. 196−211.

[312] Murray TC, Matichuk D, Brassil M, Gammie P, Bourke T, Seefried S, Lewis C, Gao X, Klein G. sel4: From general purpose to a proof of information flow enforcement. In: Proc. of the IEEE Symp. on Security and Privacy. 2013. 415−429.

[313] Zhao YW, Sanán D, Zhang FY, Liu Y. Refinement-based specification and security analysis of separation kernels. IEEE Trans. on Dependable and Secure Computing, 209, 16(1): 127−141.

[314] Saboori A, Hadjicostis CN. Verification of infinite-step opacity and complexity considerations. IEEE Trans. on Automatic Control, 2012, 57(5): 1265−1269.

[315] Saboori A, Hadjicostis CN. Verification of initial-state opacity in security applications of discrete event systems. Information Sciences, 2013, 246: 115−132.

[316] Saboori A, Hadjicostis CN. Verification of *k*-step opacity and analysis of its complexity. In: Proc. of the CDC. 2009. 205−210.

[317] Saboori A, Hadjicostis CN. Opacity-enforcing supervisory strategies via state estimator constructions. IEEE Trans. on Automatic Control, 2012, 57(5): 1155−1165.

[318] Keroglou C, Hadjicostis C. Initial state opacity in stochastic DES. In: Proc. of the ETFA. 2013. 1−8.

[319] Berard B, Chatterjee K, Sznajder N. Probabilistic opacity for Markov decision processes. Information Processing Letters, 2015, 115(1): 52−59.

[320] Berard B, Mullins J, Sassolas M. Quantifying opacity. In: Proc. of the QEST. 2010. 263−272.

[321] Bryans J, Kounty M, Ryan P. Modelling opacity using Petri nets. Electronic Notes in Theoretical Computer Science, 2005, 121: 101−115.

[322] Tong Y, Li Z, Seatzu C, Giua A. Verification of state-based opacity using petri nets. IEEE Trans. on Automatic Control, 2017, 62(6): 2823−2837.

[323] Cassez F. The dark side of timed opacity. In: Proc. of the ISA. 2009. 21−30.

[324] Wang LT, Zhan NJ, An J. The opacity of real-time automata. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2018, 37(11): 2845−2856.

[325] Deng MN, Wuyts K, Scandariato R, Preneel B, Joosen W. A privacy threat analysis framework: Supporting the elicitation and fulfillment of privacy requirements. Requirements Engineering, 2011, 16(1): 3−32.

[326] Bohli JM, Pashalidis A. Relations among privacy notions. ACM Trans. on Information and System Security, 2011, 14(1): Article No.4.

[327] Dong NP, Jonker H, Pang J. Enforcing privacy in the presence of others: notions, formalisations and relations. In: Proc. of the ESORICS. 2013. 499−516.

[328] Clarkson MR, Schneider F. Hyperproperties. Journal of Computer Security, 2010, 18(6): 1157−1210.

[329] Clarkson MR, Finkbeiner B, Koleini M, Micinski KK, Rabe MN, Sánchez C. Temporal logics for hyperproperties. In: Proc. of the POST. 2014. 265−284.

[330] Finkbeiner B, Hahn C. Deciding hyperproperties. In: Proc. of the CONCUR. 2016. Article No.13.

[331] Finkbeiner B, Hahn C, Stenger M. EAHyper: Satisfiability, implication, and equivalence checking of hyperproperties. In: Proc. of the CAV. 2017. 564−570.

[332] Huang X, Kroening D, Ruan W, Sharp J, Sun Y, Thamo E, Wu M, Yi X. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review, 2020, 37: Article No.100270.

[333] Seshia SA, Desai A, Dreossi T, Fremont D, Ghosh S, Kim E, Shivakumar S, Vazquez-Chanlatte M, Yue X. Formal specification for deep neural networks. In: Proc. of the ATVA. 2018. 20−34.

[334] Dreossi T, Jha S, Seshia SA. Semantic adversarial deep learning. In: Proc. of the CAV. 2018. 3−26.

[335] Katz G, Barrett C, Dill DL, Julian K, Kochenderfer MJ. Reluplex: An efficient SMT solver for verifying deep neural networks. In: Proc. of the CAV. 2017. 97−117.

[336] Ehlers R. Formal verification of piece-wise linear feed-forward neural networks. In: Proc. of the ATVA. 2017. 269−286.

[337] Wang S, Pei K, Whitehouse J, Yang J, Jana S. Formal security analysis of neural networks using symbolic intervals. In: Proc. of the USENIX Security Symp. 2018. 1599−1614.

[338] Singh G, Gehr T, Mirman M, Puschel M, Vechev MT. Fast and effective robustness certification. In: Proc. of the NeurIPS. 2018. 10825−10836.

[339] Singh G, Gehr T, Puschel M, Vechev MT. An abstract domain for certifying neural networks. In: Proc. of the ACM Programming Languages (POPL). 2019. Article No.41.

[340] Weng TW, Zhang H, Chen H, Song Z, Hsieh CJ, Daniel L, Boning DS, Dhillon IS. Towards fast computation of certified robustness for ReLU networks. In: Proc. of the ICML. 2018. 5273−5282.

[341] Zhang H, Weng TW, Chen PY, Hsieh CJ, Daniel L. Efficient neural network robustness certification with general activation functions. In: Proc. of the NeurIPS. 2018. 4944−4953.

[342] Dreossi T, Ghosh S, Sangiovanni-Vincentelli AL, Seshia SA. A formalization of robustness for deep neural networks. In: Proc. of the AAAI Spring Symp. Workshop on Verification of Neural Networks (VNN). 2019.

[343] Dutta S, Jha S, Sankaranarayanan S, Tiwari A. Output range analysis for deep feedforward neural networks. In: Proc. of the NFM. 2018. 121−138.

[344] Ruan W, Huang X, Kwiatkowska M. Reachability analysis of deep neural networks with provable guarantees. In: Proc. of the IJCAI. 2018. 2651−2659.

[345] Li J, Liu J, Yang P, Chen L, Huang X, Zhang L. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In: Proc. of the SAS. 2019. 296−319.

[346] Yang X, Yamaguchi T, Tran HD, Hoxha B, Johnson TT, Prokhorov D. Reachability analysis of deep ReLU neural networks using facet-vertex incidence. In: Proc. of the HSCC. 2021. Article No.18.

[347] Xue B, Liu Y, Ma L, Zhang X, Sun M, Xie X. Safe inputs approximation for black-box systems. In: Proc. of the ICECCS. 2019. 180−189.

[348] Dreossi T, Donzé A, Seshia SA. Compositional falsification of cyber-physical systems with machine learning components. Journal of Automated Reasoning, 2019, 63(4): 1031−1053.

[349] Dreossi T, Fremont DJ, Ghosh S, Kim E, Ravanbakhsh H, Vazquez-Chanlatte M, Seshia SA. VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems. In: Proc. of the CAV. 2019. 432−442.

[350] Fremont DJ, Chiu J, Margineantu DD, Osipychev D, Seshia SA. Formal analysis and redesign of a neural network-based aircraft taxiing system with VerifAI. In: Proc. of the CAV. 2020. 122−134.

[351] Huang C, Fan J, Li W, Chen X, Zhu Q. ReachNN: Reachability analysis of neural-network controlled systems. ACM Trans. on Embedded Computing Systems, 2019, 18(5s): Article No.106.

附中文参考文献:

[12] 王戟, 詹乃军, 冯新宇, 刘志明. 形式化方法概貌. 软件学报, 2019, 30(1): 33−61. http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[13] 卜磊, 陈立前, 陈哲, 等. 形式化方法的研究进展与趋势. 见: 中国计算机学会文集, 2018. 1−68.

[16] 张健, 张超, 玄跻峰, 熊英飞, 王千祥, 梁彬, 李炼, 窦文生, 陈振邦, 陈立前, 蔡彦. 程序分析研究进展. 软件学报, 2019, 30(1): 80−109. http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

王淑灵(1981－), 女, 博士, 副研究员, CCF 专业会员, 主要研究领域为嵌入式系统的建模与验证.

詹博华(1989－), 男, 博士, 副研究员, CCF 专业会员, 主要研究领域为交互式定理证明, 嵌入式系统的建模与验证.

盛欢欢(1996－), 女, 硕士生, 主要研究领域为形式化验证.

吴昊(1997－), 男, 硕士生, 主要研究领域为形式化验证.

易士程(1998－), 男, 硕士生, 主要研究领域为形式化验证.

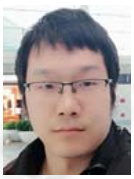
王令泰(1993－), 女, 博士生, 主要研究领域为混成系统安全与隐私的形式化验证.

金翔宇(1996－), 男, 博士生, 主要研究领域为混成系统的模型学习.

薛白(1986－), 男, 博士, 研究员, CCF 专业会员, 主要研究领域为混成系统和 AI 形式验证.

李静辉(1984－), 男, 博士, 主要研究领域为可信理论, 可靠性, AIOps.

向霜晴(1992－), 女, 博士, 主要研究领域为形式化方法, 认知逻辑, 可信理论.



向展(1991－), 男, 硕士, 主要研究领域为形式化验证, 深度学习, 网络安全.

毛碧飞(1970－), 女, 硕士, 主要研究领域为可信基础理论, IPD 可信框架和方法, 可信验证与量化评估框架, 基于声明的论证, 技术伦理前瞻性评估.