智能合约的形式化验证方法研究综述

朱 健,胡 凯,张伯钧
(北京航空航天大学软件开发环境国家重点实验室,北京 100191)

摘 要: 形式化方法是在安全关键软件系统中被广泛采用而有效的基于数学的验证方法,而智能合约属于安全关键代码,采用形式化方法验证智能合约已经成为热点研究领域. 本文对自 2015 年以来的 47 篇典型相关论文进行了研究分析,对技术进行了详细的分类研究和对比分析;对形式化验证智能合约的过程中使用的形式化方法、语言、工具和框架进行综述. 研究表明,其中定理证明技术和符号执行技术适用范围最广,可验证性质最多,很多底层框架均有所涉及,而运行时验证技术属于轻量级的新验证技术,仍处于探索阶段. 由此我们列出了一些关键问题如智能合约的自动化验证问题,转换一致性问题,形式化工具的信任问题和形式化验证的评判标准问题. 本文还展望了未来形式化方法与智能合约结合的研究方向,对领域研究有一定的推动作用.

关键词: 形式化验证; 智能合约; 区块链; 隐私保护; 信息安全; 可信交易
中图分类号: TP301 **文献标识码:** A **文章编号:** 0372-2112 (2021)04-0792-13
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.12263/DZXB.20200723

Review on Formal Verification of Smart Contract

ZHU Jian, HU Kai, ZHANG Bo-jun
(*State Key Laboratory of Software Development Environment, Beijing University of Aeronautics and Astronautics, Beijing* 100191, *China*)

Abstract: Formal methods are widely used in safety-critical software systems and have effective mathematical-based verification methods, while smart contracts belong to safety-critical codes. Using formal methods to verify smart contracts has become a popular research topic. This paper has conducted researches and analysis on 47 typical related papers since 2015 and carried out detailed classification research and comparative analysis on technology, as well as the formal methods, languages, tools and frameworks used in the formal verification of smart contracts overview. Research shows that theorem proving technique and symbolic execution technique have the widest scope of application, can verify the most properties and are involved in many basic frameworks. And the runtime verification is a new lightweight verification technology, still in the exploratory stage. From this, we have listed some key issues such as automatic verification of smart contracts, conversion consistency, trust in formal tools, and evaluation criteria for formal verification. This paper also looks forward for the future research direction on the combination of formal methods and smart contracts. For attracting more valuable ideas and promote the research in the field.

Key words: formal verification; smart contract; blockchain; privacy protection; information security; trusted transaction

1 引言

区块链技术已在金融、云计算、物联网等众多工业领域中取得了广泛的应用. 智能合约作为第二代区块链技术的创新发展以及其丰富的可发展内涵,已经越来越引起业界的重视和宽广的应用预期,成为区块链技术链上最具有发展的核心技术,在数字化的发展进程中起着越来越重要的作用. 智能合约是一段脚本代码,它允许在没有第三方的情况下进行可信交易,这些交易可追踪且不可逆转. 智能合约和区块链技术结合以来已经有广泛的应用,承载了巨大的价值和交易量,但同时,由智能合约引发的区块链安全问题也十分突

收稿日期:2020-07-16;修回日期:2020-10-07;责任编辑:覃怀银
基金项目:国家重点研发项目(No. 2018YFB1402702);国家自然科学基金(No. 61672074, No. 61672075);教育部中国移动基金(No. MCM20180104);软件开发重点实验室基金(No. SKLSDE-2020ZX-21)

出. 据研究人员统计[1],超过 34000 个智能合约都有可被利用的安全隐患,有的造成非常重大的损失,随着应用深入,总体呈上升趋势.

频频爆出的智能合约安全事件,使得越来越多来自学术界和工业界的研究人员开始将注意力集中在使用形式化方法来验证智能合约,以保证其功能的正确性和运行时的安全性. 形式化方法(formal methods)是基于数学的技术,用于系统或计算机软件的规范,开发和验证,在逻辑科学中是指分析、研究思维形式结构的方法,形式化方法设计包括描述语言,建模,模型验证,模型转换和自动代码生成等系列阶段形成的自顶向下的设计方法,已经获得业界的认可,目前在诸如航空航天等安全关键实时的系统中应用较多. 智能合约可以看作一种特殊的安全关键软件,不同之处在于它要求价值转移的安全与准确. 并且它作为合同,要求多方达成一致,更加需要形式化验证的过程来保证多方认可.

当前对形式化方法验证智能合约领域的综述类文章较少,经过调研,国内外有 2 篇相关著作. 2019 年,Singh 等人[2]对形式化方法解决智能合约存在的七个问题进行综述,包括区块链协议和扩展性问题等. 同年,Liu 等人[3]对智能合约的安全性研究进行了综述,主要包括智能合约的安全性保证和安全性验证,后者涉及了对形式化验证技术的讨论.

本文广泛收集了基于形式化方法验证智能合约领域的相关研究,选取标准为该文献对智能合约的形式化验证提出了新技术,新理论或该文献对智能合约形式化验证的相关理论和技术提供实证研究支持. 本文收录的对象为 2015 年以来符合定义的文献选取标准的相关研究,经过大量的筛选,从中挑出 47 篇相关的论文. 我们将每篇论文聚焦的问题可分为智能合约的功能性问题,隐私性问题,安全性问题和语义一致性问题:

合约功能性问题是指智能合约的行为是否符合预期,合约状态是否满足规约条件;

合约隐私性问题主要涉及智能合约的数据加密算法,协议各方的隐私保护措施;

合约的安全性通常包括有界性,可达性和无状态二义性. 当这些性质不被满足时,通常会引发安全漏洞,例如重入,整数溢出,变量未初始化等;

语义一致性问题主要讨论源代码与转换后的目标代码之间是否一致.

2　智能合约的形式化验证方法

2.1　智能合约的形式化描述

智能合约的运行可以看作从一个状态转移到另一个状态. 2019 年,Wang 等人[4]提出多个有限状态机组合来形式化描述面向合同的智能合约. 这里,我们使用五元组状态机来描述智能合约的通用形式化定义.

合约状态 M^* 是一个五元组,

$$M^* = (Q, \sum, \delta^*, s^*, F^*) \tag{1}$$

其中:

$Q = \{q_1^*, q_2^*, \cdots, q_m^*\}$,$Q$ 合约状态机所有状态的集合,$q_i^* \in q_i, (i = 1, \cdots m)$,$q_i^*$ 包含在合约参与方的所有状态中;

$\sum$ 是所有输入事件的集合;

δ^* 状态转移函数,$\delta^*: Q \times \sum \rightarrow Q$;

s^* 是初始状态,$s^* \in Q$;

F^* 是终止状态,$F^* \subset Q$.

以 Solidity 手册的安全远程购买合约(https://solidity-cn.readthedocs.io/zh/develop/solidity-by-example.html)为例,其中:

$Q = \{$合约创立,合约上锁,合约闲置$\}$;

$\sum = \{$终止交易,确认购买,确认收货$\}$;

δ^* 状态转移函数,$\delta^*: Q \times \sum \rightarrow Q$;

s^* 是初始状态,$s^* = \{$合约创立$\}$;

F^* 是终止状态,$F^* = \{$合约闲置$\}$.

图形描述如图 1 所示.

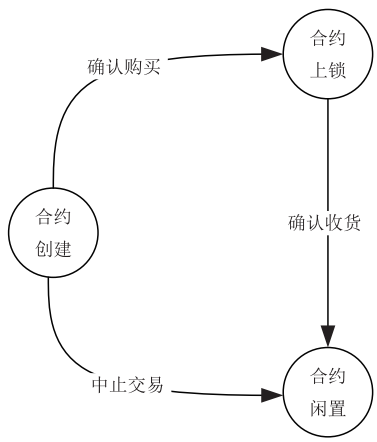

图1　智能合约通用状态机图

2.2　形式化验证的方法分类

形式化验证方法是智能合约进行确定性验证的有效手段,通过形式化语言把智能合约中的概念,判断和推理转化成智能合约模型,可以消除自然语言的歧义性和不通用性,进而采用形式化工具对智能合约建模、分析和验证,进行语义一致性测试,最后自动生成验证过的合约代码. 如图 2 所示,本节将近年来基于形式化验证智能合约的方法进行归类研究,认为大致可分为 8 种技术,并列出每种技术对应的论文数量. 基于该分类方法,对每个技术进行对比研究.

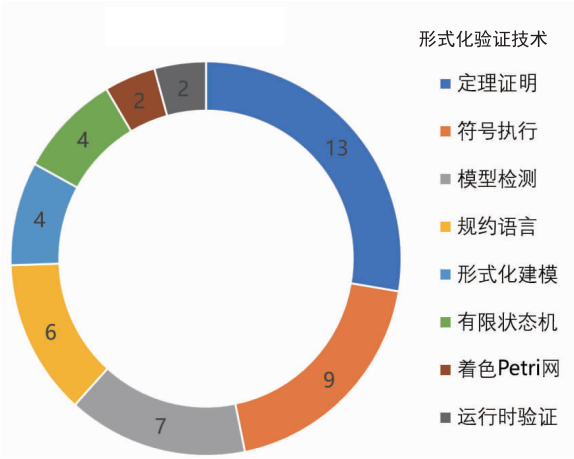

图2 形式化验证技术分类

2.2.1 基于定理证明的形式化方法

定理证明是一种利用演绎推理在符号逻辑中提供证明的形式化方法. 在这种方法中,证明的每个步骤都会引入一个公理或一个前提,并提供一个陈述,使用谓词逻辑将其进行推导,最终得到想要验证的结果[5]. 在本文调研的论文中,有 13 篇使用了定理证明技术来验证智能合约的正确性. 这是形式化验证智能合约最有效且最典型的方法,基于该方法衍生出许多有效的形式化验证工具或框架,如 F * 框架[6],Town Crier 工具[7]等.

我们把智能合约看作一段存储在分布式账本上可自动执行的代码,基于定理证明的形式化验证就是将“代码满足其规约”作为逻辑命题,通过推理规则来证明该命题. 通常需要对以下三个地方进行形式化规约:

智能合约代码. 2016 年,Bhargavan 等人[6]通过定义从 Solidity(https://solidity. readthedocs. io/en/v0. 6. 11/)智能合约到 F * 函数式编程语言的转换规则,从而使用 F * 来规约 Solidity 智能合约代码;2019 年,Nielsen 等人[8]通过 Coq 证明助手定义了智能合约的规约,该规约支持合约之间的调用功能;2020 年,Zhu 等人[9]使用基于一阶逻辑的 Event-B 语言(http://www. Event-B. org/)对 Solidity 智能合约代码进行建模.

智能合约属性. 胡等人[10]提出智能合约应当满足十条基本性质,包括合法性,一致性等,这些性质可以保证智能合约的隐私性,安全性,功能性和语义一致性. 需要将其转换为形式化规约,通过精确的规范来减少所要求属性的二义性和误解释的可能性. 2018 年,Grishchenko 等人[11]使用 F * 语言定义了许多智能合约的安全属性,如合约调用的完整性与原子性等. 作者还提出了 EVM 字节码的初始语义,并在 F * 证明助手中使用定理证明技术进行验证,发现了现有以太坊智能合约的语义和工具存在一些错误,证明了严格的形式语义基础对形式化验证的重要性. 2020 年,Sun 等人[12]总结了五种智能合约的安全性问题,分别为整数溢出问题,函数功能模糊问题,常量改变问题,智能合约权限控制问题以及智能合约中特定功能的行为改变问题,并用 Coq 证明助手对这些性质分别进行了证明.

智能合约运行环境. 虚拟机和容器可作为智能合约的运行环境,他们都是提供一个沙盒来执行智能合约,对其使用的资源进行隔离和限制. 很多智能合约的属性涉及到运行环境的限制,所以对运行环境进行形式化验证可以有效保证合约的正确性. 2017 年,Hirai[13]首次使用 Lem 语言[14]定义了 EVM,并在 Isabelle/HOL 证明助手(https://isabelle. in. tum. de)中验证了以太坊智能合约的几种安全属性,为进一步分析和扩展以太坊智能合约提供了指导. 同年,Hildenbrandt 等人[15]使用 K 框架(http://www. kframework. org/index. php/Main_Page)构建了 EVM 的验证器 KEVM,用于形式化推理和验证 EVM 的行为和性质. 基于 KEVM,作者验证了一些典型智能合约如 ERC20 合约的性质. 2018 年,Amani 等人[16]在字节码级别上进一步扩展了定理证明工具 Isabelle/HOL 中的 EVM 形式化逻辑. 作者将字节码序列构造为直线代码块,并创建一个程序逻辑来对此进行推理,以降低以太坊智能合约形式化验证的复杂性与成本. 同年,Park 等人[17]提出了一个针对以太坊虚拟机(EVM)字节码的形式化验证工具. 为了精确地推断出 EVM 字节码的所有可能行为,他们采用了 Hildenbrandt 等人[15]定义的 KEVM,并实例化了 K 框架的可达性逻辑定理证明,以生成针对 EVM 的演绎验证器. 通过引入特定于 EVM 的抽象和引理来改善验证程序,以提高可扩展性. 如表 1 所示,我们将上述基于定理证明的技术进行对比,列出了每篇文献使用的形式化工具或语言,大致对比了其主要工作的优势、劣势或不足.

除了对以上三个层次的形式化规约,还有一些其他的相关工作使用了定理证明技术来提高智能合约的安全性. 2016 年,Zhang 等人[7]以定理证明的形式化技术为主要方法,引入了 Town Crier,该工具可以充当现有受信的非区块链网站与智能合约之间的数据交互中间层. 它结合了可信硬件的后端和区块链的前端,可从启用 HTTPS 协议的网站向智能合约提供可信和经过身份验证的数据. 同年,Idelberger[18]等人提出了一种基于逻辑的智能合约,逻辑语句可以理解为智能合约的可执行规范,从而降低合约错误的风险. 最后,作者提供了在区块链系统上使用此类基于逻辑的智能合约指南,以提高智能合约编写的正确性. 2017 年,Sergey 等人[19]探讨了共享内存并发的经典问题与以太坊智能合约的多事务行为的相似性. 将以太坊智能合约多事务的特性与共享内存并发的经典问题进行类比,对智能合约潜

在威胁的有了更深刻的理解,最后并使用定理证明加以验证. 2018 年,Sánchez 等人[20]提出一个编程框架 Raziel,它内置有大量的定理推导相关规则,通过形式化方法来验证智能合约的多方计算的安全问题,为智能合约提供安全性保障.

表 1　定理证明技术对比

语言/工具	形式化规约类型	优势	劣势或不足
F * 语言	代码规约	支持对智能合约和虚拟机规约,可验证复杂属性	自动化程度低,建模复杂,不支持智能合约的循环语句结构
Event-B 语言	代码规约	建立智能合约的 Event-B 模型并给出转换规则,可验证合约的功能性	不支持循环语句建模,自动化程度低,验证复杂的性质需要手动插入
Coq 证明助手	代码规约	对 The DAO 合约规约并找到漏洞,优化了代码	应用范围局限,需要手动建模实现,比较复杂
F * 证明助手	属性规约	完整地定义了 EVM 的操作语义并给出智能合约的一些安全属性	实现比较复杂,不能验证具体的智能合约属性
Coq 证明助手	属性规约	针对智能合约的五种安全问题,提出验证智能合约的框架	可验证智能合约属性有限
K 框架	运行环境规约	语义完整,支持同步更新,可扩展性强,可用于衍生分析或验证虚拟机性质的工具	KEVM 验证器处于起步阶段,缺少具体验证智能合约工具
KEVM 验证器	运行环境规约	该验证器基于完整的 EVM 语义,自动化程度较高,可验证一些合约的功能性和安全性	应用范围局限
Isabelle/HOL 工具	运行环境规约	用 Isabelle/HOL 建立 EVM 规约,涵盖智能合约属性,包括 gas 执行情况,可自动生成验证条件	基于 EVM 的部分语义并不完整,可能出现 EVM 字节码临界情况,不利于验证合约的可靠性
Lem 语言 Isabelle/HOL 工具	运行环境规约	定义了 EVM 的形式语义,面向多种证明助手 Coq,Isabelle/HOL	建模复杂,建立 EVM 模型远小于真实环境,没有考虑虚拟机网络环境,对智能合约的验证受限,不支持合约间的调用

2.2.2　基于符号执行的形式化方法

符号执行[21]是一种兴起于 70 年代中期的程序分析方法,它可以通过分析程序来得到让特定代码区域执行的输入,用于测试某些软件是否满足一些特定属性. 近年来,符号执行技术与智能合约相结合用于形式化验证智能合约已成为主流趋势. 本文所调研的文献中,我们发现共有 9 篇使用符号执行技术来验证智能合约的正确性.

首先,该技术主要为验证智能合约的工具设计提供坚实的基础. 2016 年,Luu 等人[22]将以太坊平台的操作语义[23]进行形式化规约[24],以提高智能合约的安全性. 此外,他们开发了一个基于符号执行技术的测试工具 Oyente,它可以用来检测合约代码潜在的安全漏洞;2018 年,Mueller[25]设计了一个基于符号执行的安全分析工具 Mythril,用来分析以太坊的智能合约;2018 年,Tsankov 等人[26]开发了一个针对以太坊平台上智能合约的安全漏洞分析器 Securify. 该工具是完全自动化且可以针对给定的性质来验证智能合约的行为是否安全. 同年,Nikolic 等人[27]实现了一种基于符号分析和程序验证器的工具 MAIAN,该工具通过分析智能合约函数之间调用的字节码序列,来检测可能存在的安全漏洞. 2020 年,Permenev 等人[28]设计了一个可验证以太坊智能合约功能性的自动验证器 VerX,该验证器主要融合三种技术:将时序性简化为可达性检查;基于以太坊虚拟机的符号执行技术以及将谓词抽象和符号执行技术结合的延迟谓词抽象技术.

其次,符号执行技术是一种白盒的静态分析技术[29]. 该技术还可以静态分析和验证智能合约代码的执行路径,并验证其正确性. 2018 年,Le 等人[30]提出基于符号执行技术的静态分析方法,通过静态地证明智能合约的终止(或不终止)情况来确定智能合约终止(或不终止)的输入条件. 通过这种方法,可以保证智能合约在运行时的安全性. 同年,Zhou 等人[31]设计了一种称为 SASC 的静态分析工具,该工具基于符号执行技术,用于智能合约逻辑分析,可以生成函数之间调用的拓扑关系图,以帮助查找智能合约潜在的逻辑漏洞.

最后,符号执行技术还可以与语义结构分析和高阶逻辑定理证明相结合,以此来验证智能合约的安全性. 2020 年,Bang 等人[32]提出一种将语义结构分析和符号执行技术结合的控制流方法,用于审查以太坊智

能合约的漏洞. 同年,Yang 等人[33]建立了一个基于符号执行技术和高阶逻辑定理证明的智能合约验证框架 FSPVM-E,用于验证以太坊智能合约的可靠性和安全性. FSPVM-E 支持 ERC20 合约标准,并且可以自动执行以太坊智能合约代码,扫描其漏洞,并使用 Coq 证明助手的霍尔逻辑[34]验证其可靠性和安全性.

2.2.3 基于模型检测的形式化方法

从形式化验证的角度来看,除了以逻辑推理为基础的定理证明外,还有以穷尽搜索为基础的模型检测. 模型检测技术[35]是一种基于状态迁移系统的自动验证技术. 其基本思想是:检验一个结构是否满足一个公式要比证明公式在所有结构下均被满足容易得多,进而面向并发系统创立了在有穷状态模型上检验公式可满足性的验证新形式[36]. 模型检测方法也已用于形式化验证智能合约的正确性,在本文所调研的文献中,有 7 篇使用了模型检测技术,该技术验证智能合约的主要原理如图 3 所示. 通常情况下,需要迭代验证的过程,才能最终满足验证条件.

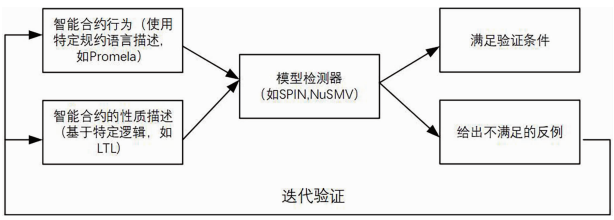

图3 模型检测技术验证智能合约的原理

我们发现,所有使用模型检测技术验证智能合约的论文都使用了自动模型检测工具. 2018 年,Nehaï 等人[37]提出了一种使用模型检测技术形式化验证以太坊智能合约正确性的方法. 该方法的基础是按照三层建模过程即区块链内核层,应用程序层和环境层建立模型. 最后使用 NuSMV 模型检查工具(http://nusmv.fbk.eu/)验证智能合约的功能性. 同年,Abdellatif 等人[38]提出了一种基于强语义的形式化建模方法,用于验证智能合约的功能性. 作者在 BIP 形式化建模框架(http://www-verimag.imag.fr/The-BIP-Framework.html?lang=en)中模拟智能合约的行为,并使用 SMC 模型检测工具[39]分析结果,该结果显示了恶意用户可以使用非法手段操纵智能合约的情况. 同年,Bai 等人[40]使用 SPIN 模型检测工具(http://spinroot.com/spin/whatispin.html)形式化验证智能合约中属性的正确性. 同年,Kalra 等人[41]提出了一种智能合约安全性验证工具 ZEUS,作者使用模型检测和抽象解释进行验证智能合约的安全性. 作者使用 ZEUS 还为 Fabric(https://www.hyperledger.org/use/fabric)和以太坊区块链平台构建了其原型框架. 结果表明,以太坊上的智能合约约有 94.6% 易受到攻击. 2020 年,Antonino 等人[42]将 Solidity 语言进行了抽象,这些抽象是基于以太坊智能合约的行为方式进行了归纳. 基于此进行形式化,创建了 Solidity 的模型检查器 Solidifier. 其过程是将 Solidity 转换为一种中间验证语言 Boogie(https://www.microsoft.com/en-us/research/project/boogie-an-intermediate-verification-language),随后使用 Corral 边界模型检查器[43]对其验证,以此来验证智能合约的功能性.

模型检测技术还可以与博弈论思想结合,来验证智能合约的功能性是否正确. 早在 2015 年,Bigi 等人[44]分析和验证了 DSCP 协议(去中心化智能合约协议),通过博弈论模型分析在各方达成协议和未达成协议不同情况下智能合约的行为,其结果用于指导建立智能合约的形式化模型,随后作者使用模型检测工具 PRISM(https://www.graphpad.com/scientific-software/prism)验证其功能正确性.

以上模型检测技术主要用于验证单一智能合约的功能性和安全性,2020 年 Alqahtani 等人[45]提出一种模型检测的形式化方法,以验证针对不同实体开发和控制的交互式智能合约的安全性. 作者使用 NuSMV 模型检查器和行为交互优先级工具对智能合约的行为及其交互进行建模,以验证智能合约是否满足系统功能的要求. 美国石油学会的案例研究证实了该方法的适用性,并在 Fabric 区块链平台实施了该方法,以验证交互式智能合约的安全性.

2.2.4 基于规约语言的形式化方法

形式化规约语言,是一种由严格的递归语法规则所定义的语言. 一般可分为模型规约语言和性质规约语言. 通常两种规约使用的是同一种语言,方便规约之后进行验证工作,如 dSLAC[46]. 如果在验证过程中涉及到不同的规约语言,需要将不同的规约语言进行转换. 2020 年,Bernardi 等人[47]建立了基于几个通用不变式的规约框架,通过将不变式定义为霍尔三元组,解决了智能合约规约语言转换的差异性问题.

我们统计的文献中,有 6 篇论文应用了该技术,并且不同于定理证明技术中应用形式化语言或框架来验证智能合约,此处基于规约语言的形式化方法特指专门设计出新的规约语言用于描述智能合约模型.

2017 年,Scoca 等人[46]提出了一种形式化规约语言 dSLAC,用于规约智能合约中请求和响应之间的交互. 同年,Biryukov 等人[48]提出一种针对区块链网络的金融领域智能合约的特定形式化规约语言 Findel,该语言可用于规约智能合约条款,可以防止对合约产生歧义,同时生成智能合约标准化模板,验证智能合约的安全性. 2018 年,He 等人[49]引入一种智能合约的规约语言 SPESC. 该语言定义了一些规约,为多人编写智能合

约提供参考. 可以像用自然语言编写现实世界合同一样来定义智能合约,以此来帮助用户明确合约所涉及的权利和义务. 同年,Yang 等人[50]提出 Solidity 智能合约的形式化语言 Lolisa,该语言具有经过验证的形式化语义和语法,基于该语言编写的智能合约代码可以在 Coq 证明助手中检验属性的正确性. 2020 年,Park 等人[51]使用 K 框架形式化验证了以太坊上具有存款功能的智能合约,首先将智能合约编译为字节码,然后对字节码进行形式化规约并验证. 该方法可以保证智能合约的安全性和功能性不依赖于编译器的正确性.

2.2.5　基于形式化建模的形式化方法

形式化建模是使用精确的数学语句或模型组件来设计系统的一种技术,该技术可以很好地定义系统中组件之间的关系,保证了系统行为的无二义性,对基于形式化建模的系统进行仿真的结果是可以复现的,消除了事件发生的偶然性. 在本文所统计的文献中,有 4 篇论文使用到形式化建模的方法. 最典型的是在 2016 年,Kosba 等人[52]使用 UC 框架[53]建立了一套去中心化区块链系统 Hawk,基于该框架,开发人员可以很直观的编写智能合约而无需对其进行加密. 因为其编译器可以自动生成一个高效的加密协议,此协议为 Hawk 内置智能合约生成. 在此协议中,智能合约的交互双方使用零知识证明[54]等加密原语与区块链进行交互,从而保证了智能合约数据的隐私性.

2017 年,Kim 等人[55]提出智能合约形式化建模可以降低价值交换不确定性的观点,此外,该方法在智能合约透明化的同时还可以验证数据的隐私性. 作者通过分析 Intellichain(http://www. intellicha. in)来验证上述观点,该区块链建立了一个形式化,支持用户模拟的智能合约模型.

形式化建模不仅可以提高智能合约的隐私性,还可以用来检测智能合约的漏洞以及对合约功能性验证. 2018 年,Breidenbach 等人[56]建立了 Hydra 框架,该框架基于漏洞赏金的模式和 NNVP 编程,它是 N-version 编程[57]的一种变体,可检测多个程序实例之间差异的一门技术,以鼓励开发者和用户诚实地披露智能合约的错误和漏洞. 作者将 Hydra 框架应用到以太坊 ERC20 合约,找到该合约存在的整数溢出,差一错误等漏洞. 2020 年,Beillahi 等人[58]设计一种自动化方法来形式化验证智能合约的功能性,并提出智能合约行为精化的概念. 作者通过建模分析相关智能合约状态的关系,提出了一种自动归纳证明的方法. 他们证明了对合约行为建模可应用于令牌,拍卖和代管等几种常用的智能合约.

2.2.6　基于有限状态机的形式化方法

有限状态机是描述有限的系统状态以及在这些状态之间转换等动作的技术. 状态机可以根据外部的输入来更改内部的状态. 现已被广泛应用于各个学科领域,是有效且常见的形式化方法之一. 智能合约的执行可以看作从一个状态到下一个状态的变迁. 本文所统计的相关文献中,有 4 篇使用了有限状态机技术,来形式化验证智能合约的正确性.

智能合约在运行时,其运行状态可以抽象为有限状态机,即每发生一次交易,状态均会发生改变. 2018 年,Sergey 等人[59]就是运用这种思想,设计了 Scilla 这个智能合约中间层语言,它具有函数式编程的风格与 Coq 证明助手编程风格十分相近. 它可以作为高级智能合约程序设计语言的转换目标,如 Solidity. Scilla 的计算模型是基于有限状态机,可以将智能合约的运行状态转换为有限状态机模型,然后自动化地进行验证智能合约的安全性和时态性. 验证的过程可以通过 Coq 证明助手来实现,将合约的每个状态进行穷举,检查状态的变更输入输出条件是否符合标准.

智能合约参与方的行为,也会影响智能合约的状态改变. 2019 年,Xu 等人[60]建立了系统化的智能合约安全模型. 为建立模型,首先将智能合约参与方的行为抽象为有限状态机,再对智能合约的函数进行建模,对应有限状态机中不同状态之间的转换. 然后使用时序逻辑,查找合约的漏洞,主要包括未知函数调用,gas 耗尽,重入攻击,合约进入不可预料的状态等安全性问题.

智能合约的语义状态同样可抽象为有限状态机,从而验证智能合约的正确性. 2018 年,Mavridou 等人[61]提出了 FSolidM 框架,该框架允许开发人员将智能合约定义为有限状态机. 作者提供了一种在图形界面上创建有限状态机并自动生成以太坊智能合约. 此外,作者还介绍了一组设计模式,这些模式以插件的形式设计,开发人员可以轻松地将其添加到智能合约中以增强安全性. 2019 年,Wang 等人[62]基于 Azure 区块链智能合约的语义,使用有限状态机技术形式化验证了其智能合约的语义一致性,并提出一种基于程序指令的方法将语义一致性检查简化为对合约安全性验证. 同时,他们基于对 Boogie 语言的转换,开发了一个新的 Solidity 程序验证器 VeriSol,并用其分析 Azure 区块链智能合约,且在已发布的智能合约中找出了一些未知漏洞.

2.2.7　基于着色 Petri 网的形式化方法

Petri 网[63]是 20 世纪 60 年代由 Carl Adam Petri 发明的,适合于描述异步的、并发的系统模型. 由于 Petri 网对事件的行为描述能力不足,这就诞生了着色 Petri 网[64],它在原有 Petri 网的基础上加入了颜色集和模型声明等元素,借此可以表达更复杂的类型信息. 在我们统计的论文中,有 2 篇应用了着色 Petri 网的形式化方

法,来验证智能合约的安全性和功能性.

2019 年,Liu 等人[65]提出了一种基于有色 Petri 网的形式化验证方法来验证智能合约. 基于分层着色 Petri 网建立有可能被攻击的智能合约模型,通过逐步仿真来验证智能合约功能的正确性. 他们利用基于分支时序逻辑的着色 Petri 网工具,进行模型检测,寻找智能合约中的潜在漏洞. 最后作者证明了基于着色 Petri 网建模的验证方法不仅可以检测智能合约逻辑漏洞,还可以检测由用户行为引起的智能合约非逻辑漏洞. 2020 年,Wang 等人[66]提出了一种智能合约多层次建模的解决方案来验证智能合约的安全性. 他们改进了字节码程序逻辑规则,并应用霍尔逻辑创建了基于着色 Petri 网的模型检测工具. 该工具提供的模型检测方法可以显示完整的状态改变和错误的执行路径,从而帮助用户从多个角度分析智能合约的安全性.

2.2.8 基于运行时验证的形式化方法

运行时验证是一个轻量级的新验证技术方向,它并非传统的形式化方法,运行时验证是一种对计算系统分析和执行的方法. 它从正在运行的系统中提取有用信息用来检测系统的行为是否满足或者违背某些属性[67]. 本次调研的论文中,有 2 篇涉及到该方法. 主要是通过编译器设计和分析智能合约执行路径来保证智能合约的正确性.

2018 年,Ellul 等人[68]使用了运行时验证的形式化方法,来保证智能合约运行时的执行路径满足安全性规约,同时他们开发了一个基于概念证明的协议工具 ContractLarva. 当智能合约条件被违反时,该工具可有效阻止智能合约继续执行,以提高智能合约功能的正确性. 2020 年,Li 等人[69]设计了一个称为 Solythesis 的编译器,通过运行时验证的形式化方法来保证智能合约的安全性,向编译器输入 Solidity 源代码和智能合约的属性规约,可以生成一个新的智能合约,它有效地去除了之前违背属性规约的交易,使其更加安全可靠.

2.3 形式化方法总结

有关形式化验证智能合约的方法大致可分为 8 种,本文以每个技术特点为切入点对每一种技术进行对比分析,研究了其各自的优势,劣势或不足,使用的难易程度和可验证智能合约的性质. 如表 2 所示.

表 2 形式化方法总结表

形式化方法	优势	劣势或不足	使用难易程度	可验证合约性质
定理证明	使用数学的方法,通过公理或前提进行推导,保证验证的严谨性	无法保证在源代码与验证代码之间的转换一致性,实现成本高,自动化水平低	高	隐私性,安全性,功能性,语义一致性
符号执行	以符号值作为输入,借助相应工具,可得到具体测试用例,具有很高的代码覆盖率	本质上属于测试,不能 100% 证明智能合约无误	高	安全性,功能性
模型检测	使用市面上现有的模型检测工具,进行自动化验证	无法保证所使用的模型检测工具的完备性与正确性,合约复杂度过高会导致状态空间爆炸	低	安全性,功能性
规约语言	使用数学的思想,提出新的规约语言,保证验证的严谨性	无法保证新规约语言的正确性且实现成本高	高	安全性,功能性
形式化建模	使用精确的数学语句或模型组件来设计系统,仿真结果可以复现	此方法大多使用市面上建模框架,其框架的完备性与正确性无法保证	中等	隐私性,安全性,功能性
有限状态机	思维导向简单,将智能合约抽象转换为状态机的形式,易于操作,且具有图形界面	状态定义的好坏,对智能合约的验证难易程度有很强相关性,合约复杂度过高会导致状态空间爆炸	中等	安全性,语义一致性
着色 Petri 网	基于已有的 Petri 网模型,进行形式化验证,具有良好的语义描述且具有图形界面	当智能合约逻辑较为复杂时,可能会导致可达图生成难度增加,状态空间爆炸	低	安全性,功能性
运行时验证	轻量级的新型验证技术,具有自适应和自我调整	由于运行时验证的特点,要求必须在智能合约运行时检测漏洞,无法在生产环境中使用	高	安全性,功能性

我们可以看出,定理证明方法可检测智能合约的属性最多,适用范围最广,但使用此方法的研究人员要求具有很强的数学功底,所以使用难度较高. 符号执行方法多用于设计相关验证工具,或与其他技术如语义结构分析,高阶逻辑定理等相关知识结合使用,要求知识面要广,使用难度也较高. 模型检测方法不涉及太多的数学相关知识,使用模型检测工具即可自动化验证,故使用难度较低. 规约语言方法涉及开发新的语言,需要对计算机编译原理等相关知识有很深入的理解,故使用难度较高. 形式化建模方法虽然涉及的领域很多,例如区块链加密协议,NNVP 编程模型等,但还是可以使用市面上大多的建模框架进行建模验证,故难度为中等. 有限状态机方法的思维导向单一,均将智能合约的相关状态抽象为自动机的形式并加以验证,但其抽象对象的准确性需要经过大量的实验证明,故难度为中等. 着色 Petri 网方法借助已有的 Petri 网理论进行形式化验证,故其难度较低. 运行时验证方法使用条件相对苛刻,是一种在线的自适应,可自我修复的方法. 此技术与智能合约相结合的验证方法尚不成熟,运行时验证与智能合约一旦部署将不可回滚等特性相矛盾,因此它与此技术的结合仍待考察,使用难度较高.

3　形式化规约语言与框架

3.1　形式化规约语言

在本文研究的相关工作中,有 11 篇著作提出了领域特定语言来规范智能合约. 表 3 中分别列出了这些语言及介绍.

表格中列出的 11 篇论文中有 7 篇关注智能合约功能验证,例如 Scilla,SPESC 等,这些语言验证了几种功能和安全属性,有效杜绝了智能合约相关错误和漏洞.

F * 是本综述收集的所有相关作品中被多次提及的编程语言. 它用于构建程序的形式化验证,用于在源代码和字节代码级别上验证安全属性.

表 3　语言对比汇总表

提出/使用的语言	规约语言简介
SPESC	SPESC 是一种智能合约规约语言,可以用自然语言定义,明确了智能合约中当事人的权利和义务
PROMELA	PROMELA 是一种用于形式验证的建模语言,PROMELA 模型经常使用诸如 SPIN 之类的模型检查器进行分析,以验证系统功能的正确性
F *	F * 是一种函数式编程语言,用于形式化验证程序的正确性
Securify language	Securify 语言是 Securify 工具的领域特定语言,该语言主要用于验证智能合约的安全性
Scilla	Scilla 是高级智能合约程序语言的转换目标,它的计算模型是基于有限状态机
dSLAC	dSLAC 是一种用于云计算领域智能合约的规约语言
Lem	Lem 是一种在工程领域用于语义建模的语言,该语言可以和证明助手有很好的交互,如 Lem 语言可用于 Ocaml 测试,可用于 Coq 证明
Findel	Findel 是一种金融领域特定的语言,该语言可用于规约智能合约条款,使其方便机器识别和确保无二义性
Event-B	Event-B 是一种基于传统的谓词演算和定理证明的形式化语言,它适合用来为周期性行为建模
Boogie	Boogie 是一种中间验证语言,旨在构建其他语言的验证程序的中间层
Lolisa	Lolisa 是 Solidity 语言的子集,该语言有经过验证的形式化语义和语法,编写的程序可以直接在定理证明助手中验证

3.2　形式化工具及框架

本文研究的所有文献中,共涉及 23 个验证工具或形式化框架. 这些工具或框架底层大多数基于定理证明和符号执行技术. 其他方法,例如有限状态机,模型检测以及形式化建模等技术也可用于支持这些工具和框架. 表 4 汇总了这些工具和框架,并给出了各自的描述.

表 4　工具及框架汇总表

工具/框架	使用的形式化方法	工具描述
Solidity * and EVM *	定理证明技术	该框架使用函数式语言 F * 分析验证了 Solidity 智能合约运行时的正确性
Corral	定理证明技术	Corral 是 Boogie 语言的分析工具. 默认情况下,Corral 会进行有界搜索,直到递归深度和固定数量到达一定限度为止
Coq	定理证明技术	Coq 是一个交互式定理证明助手,它提供了一种形式化的语言来编写数学定义,可执行算法和定理
Town Crier	定理证明技术	Town Crier 是现有受信的非区块链网站与智能合约之间的数据交互中间层

续表

工具/框架	使用的形式化方法	工具描述
Raziel	定理证明技术	Raziel 是一个编程框架,用于验证智能合约的多方计算的安全问题,为智能合约的隐私性提供保障
Isabelle/HOL	定理证明技术	Isabelle/HOL 是一个基于高阶逻辑的通用交互式定理证明器
ZEUS	定理证明技术模型检测技术	ZEUS 是一种智能合约安全验证工具,它同时使用符号模型检查和抽象解释进行严格证明
FSPVM-E	定理证明技术符号执行技术	FSPVM-E 是采用了符号执行和高阶逻辑定理证明相结合的形式化验证框架
SASC	符号执行技术	SASC 是一种静态分析工具,它可以发现潜在的逻辑风险并生成调用关系的拓扑图
MAIAN	符号分析技术	MAIAN 是用于智能合约中属性的跟踪工具,该工具利用符号分析和具体验证器来查询漏洞
Securify	符号执行技术	Securify 是一个针对以太坊智能合约的安全漏洞分析器
Mythril	符号执行技术	Mythril 是一个基于符号执行的安全分析工具,用来分析以太坊的智能合约
Oyente	符号执行技术	Oyente 是一个基于符号执行技术的测试工具,可以用来检测合约代码潜在的安全漏洞
VerX	符号执行技术	VerX 是一个可以自动验证以太坊智能合约功能性的验证器
ContractLarva	有限状态机技术	ContractLarva 工具可对以太坊上的 Solidity 智能合约运行时的安全情况进行验证
VeriSol	有限状态机技术	VeriSol 工具使用访问控制策略针对状态机工作流对智能合约语义一致性进行形式化检测
FSolidM	有限状态机技术	FSolidM 工具在图形界面上将智能合约设计为有限状态机的形式并进行验证,它也可以自动生成以太坊智能合约代码
SPIN	有限状态机技术	SPIN 是一种显式模型检测工具,用以检测一个有限状态系统是否满足 PLTL 公式以及其他一些性质,包括是否有循环或可达性
NuSMV	模型检测技术	NuSMV 是 SMV 工具的重新实现和扩展,SMV 是第一个基于 BDD 的模型检查器. NuSMV 被设计为用于模型检查的开放架构,可以可靠地用于工业设计验证
BIP	模型检测技术	BIP 框架是一个通用的系统级形式化建模框架,支持层次化结构,并包含一套支持建模、模型转换、仿真、验证和代码生成的工具集
PRISM	模型检测技术	PRISM 是一个概率模型检查器,它是对表现出随机或概率行为的系统进行形式化建模和分析的工具
SMC	模型检测技术	SMC 是一个模型检测器,可用于检查在不同公平性假设下并发程序的安全性和活性
Hydra	形式化建模技术	该框架基于漏洞赏金的模式和 NNVP 编程,以鼓励开发者和用户诚实地披露智能合约中的错误和漏洞

4 存在的问题和未来的方向

形式化验证智能合约的领域仍处于起步阶段,虽然已经取得了一些突破性进展,但是现阶段发展并不成熟,还面临着以下问题:

(1)智能合约的自动化验证问题. 通常自动化程度越高的方法或框架,所支持验证合约的性质越局限,我们调研的大部分研究基于自动化的验证方法都是有条件限制的,同时需要具备专业知识的人员参与调试,并不利于智能合约形式化方法的广泛应用.

(2)转换一致性问题. 形式化验证智能合约往往需要在不同模型之间转换,以便将高级语言转换为形式化工具可以验证的语言,很多论文提出了创新且可行的规约语言或模型工具,但是没有对转换模型进行一致性验证,对源代码和目标语言的语义一致性需要进行严格的证明,以保证经过验证的智能合约满足原始合约的语义及功能.

(3)形式化工具的信任问题. 通常,保证形式化工具和编译器的正确性比验证智能合约的正确性要更复杂,当解决问题的成本超过问题本身时,我们不得不怀疑解决该问题是否有意义.

(4)形式化验证的评判标准问题. 当前形式化验证智能合约的方法不断增加,但是缺少一个衡量形式化

验证好坏程度的标准. 如何判定形式化方法验证智能合约的效率,验证的准确度. 现阶段主要靠开发人员凭借经验的主观定义,不具备权威性.

随着智能合约的不断发展,会面临着诸多挑战,但是新的挑战也产生了新的机遇. 形式化方法验证智能合约就显得尤为重要. 我们认为未来的重点研究方向将围绕以下几点:

(1)形式化方法在智能合约全周期生产和运行中的适应性和可应用问题,尽量提高形式化技术的自动化程度. 形式化方法不仅可应用于验证,它更是一套自顶向下的系统软件设计方法,可以进行自动代码生成和快速反馈迭代设计等,这方面的研究将对智能合约更可靠和快速的生产和更新具有重要应用价值.

(2)加快建立一套公认的形式化转换规则,此规则包括但不限于如语义转换规则,逻辑转换规则,行为转换规则等. 应用此转换规则可以保证形式化转换的正确性.

(3)建立形式化验证智能合约的评判标准,用于规范与管理不同的形式化方法,指导新技术的产生. 形式化方法已经成为安全关键系统验证评估的重要标准,如国际 CC 软件认证标准. 在未来智能合约评测和认证中,也需要一个标准,研究相关的标准和自动测试平台也是具有重要意义的.

5　结论

在本文中,我们综述了有关形式化验证智能合约的最新研究成果并总结了当前阶段尚存的一些问题,包括智能合约的自动化验证问题,转换一致性问题,形式化工具的信任问题和形式化验证的评判标准问题,并展望未来的研究方向,以期对领域研究者具有参考意义. 我们相信,随着智能和合约契约化、法律化、规模应用化的发展,形式化方法在智能合约的设计、生产和运行过程中,将会起到越来越重要的作用,得到更普遍的应用.

参考文献

[1] Nikolic I, et al. Finding the Greedy, Prodigal, and Suicidal Contracts at Scale[EB/OL]. https://arxiv.org/abs/1802.06038, 2018-05-14/2020-08-20.

[2] Amritarj Singh, Reza M Parizi, Zhang Qi, et al. Blockchain smart contracts formalization: approaches and challenges to address vulnerabilities[J]. Computers & Security, 2020, (88): 1 – 10.

[3] Liu J, Liu Z. A survey on security verification of blockchain smart contracts[J]. IEEE Access, 2019, (7): 77894 – 77904.

[4] 王璞巍,杨航天,孟佶,等. 面向合同的智能合约的形式化定义及参考实现[J]. 软件学报, 2019, 30(9): 2608 – 2619.
Wang P W, Yang H T, Meng J, et al. Formal definition for classical smart contracts and reference implementation[J]. Journal of Software, 2019, 30(9): 2608 – 2619. (in Chinese)

[5] Leroy X. A formally verified compiler back-end[J]. Journal of Automated Reasoning, 2009, 43(4): 363 – 446.

[6] Bhargavan K, Delignat-Lavaud A, et al. Formal verification of smart contracts: short paper[A]. The 2016 ACM Workshop on Programming Languages and Analysis for Security[C]. USA: ACM, 2016. 91 – 96.

[7] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, Elaine Shi. Town Crier: an authenticated data feed for smart contracts[A]. The 2016 ACM SIGSAC Conference on Computer and Communications Security[C]. USA: ACM, 2016. 270 – 282.

[8] Nielsen B J, Spitters B. Smart Contract Interactions in Coq [EB/OL]. https://arxiv.org/abs/1911.04732, 2019-12-2/2020-8-20.

[9] Zhu J, Hu K, Filali M, et al. Formal Verification of Solidity Vontracts in Event-B [EB/OL]. https://arxiv.org/abs/2005.01261, 2020-5-4/2020-8-20.

[10] 胡凯,白晓敏,等. 智能合约的形式化验证方法[J]. 信息安全研究, 2016, 2(12): 1080 – 1089.
Hu K, Bai X M, et al. Formal verification method of smart contract[J]. Journal of Information Security Research, 2016, 2(12): 1080 – 1089. (in Chinese)

[11] Grishchenko I, Maffei M, Schneidewind C. A semantic framework for the security analysis of ethereum smart contracts[A]. International Conference on Principles of Security and Trust[C]. Germany: Springer, 2018. 243 – 269.

[12] Sun T, Yu W. A formal verification framework for security issues of blockchain smart contracts[J]. Electronics, 2020, 9(2): 254 – 255.

[13] Hirai Y. Defining the ethereum virtual machine for interactive theorem provers[A]. International Conference on Financial Cryptography and Data Security[C]. Germany: Springer, 2017. 520 – 535.

[14] Mulligan D P, Owens S, Gray K E, et al. Lem: Reusable engineering of real-world semantics[J]. ACM SIGPLAN Notices, 2014, 49(9): 175 – 188.

[15] Hildenbrandt E, et al. KEVM: A complete formal semantics of the ethereum virtual machine[A]. 2018 IEEE 31st Computer Security Foundations Symposium[C]. USA: IEEE, 2018. 204 – 217.

[16] Sidney A, Myriam B, Maksym Bo, Mark Staples. Towards verifying ethereum smart contract bytecode in Isabelle/HOL[A]. The 7th ACM SIGPLAN International Confer-

ence on Certified Programs and Proofs[C]. USA:ACM, 2018. 66 – 77.

[17] Daejun P, Zhang Y, Manasvi S, et al. A formal verification tool for ethereum VM bytecode[A]. The 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering[C]. USA:ACM, 2018. 912 – 915.

[18] Idelberger F, Governatori G, Riveret R, et al. Evaluation of logic-based smart contracts for blockchain systems[A]. International Symposium on Rules and Rule Markup Languages for the Semantic Web[C]. Germany: Springer, 2016. 167 – 183.

[19] Sergey I, Hobor A. A concurrent perspective on smart contracts[A]. International Conference on Financial Cryptography and Data Security[C]. Germany: Springer, 2017. 478 – 493.

[20] Sánchez D C. Raziel: Private and Verifiable Smart Contracts on Blockchains[OL]. https://arxiv. org/abs/1807. 09484, 2018-7-25/2020-8-20.

[21] Schwartz E J, Avgerinos T, Brumley D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution[A]. 2010 IEEE Symposium on Security and Privacy[C]. USA: IEEE, 2010. 317 – 331.

[22] Luu L, Chu D H, Olickel H, et al. Making smart contracts smarter[A]. The 2016 ACM SIGSAC Conference on Computer and Communications Security [C]. USA: ACM, 2016. 254 – 269.

[23] Plotkin G D. A structural approach to operational semantics[J]. Journal of Logic and Algebraic Programming, 2004, 60(61): 121 – 134.

[24] Hierons R M, BogdanovK, Bowen J P, et al. Using formal specifications to support testing [J]. ACM Computing Surveys, 2009, 41(2): 1 – 76.

[25] Mueller B. Smashing ethereum smart contracts for fun and real profit [A]. HITB SECCONF [C]. USA: Hacker Noon, 2018. 1 – 10.

[26] Tsankov P, Dan A, Drachsler-Cohen D, et al. Securify: Practical security analysis of smart contracts [A]. The 2018 ACM SIGSAC Conference on Computer and Communications Security[C]. USA: ACM, 2018. 67 – 82.

[27] Nikolić I, Kolluri A, Sergey I, et al. Finding the greedy, prodigal, and suicidal contracts at scale[A]. The 34th Annual Computer Security Applications Conference [C]. USA: ACM, 2018. 653 – 663.

[28] Permenev A, Dimitrov D, Tsankov P, et al. Verx: safety verification of smart contracts[A]. IEEE Symposium on Security and Privacy[C]. USA: IEEE, 2020. 18 – 20.

[29] Wichmann B A, Canning A A, Clutterbuck D L, et al. Industrial perspective on static analysis[J]. Software Engineering Journal, 1995, 10(2): 69 – 75.

[30] Le T C, Xu L, Chen L, et al. Proving conditional termination for smart contracts[A]. The 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts [C]. USA: ACM, 2018. 57 – 59.

[31] Zhou E, Hua S, Pi B, et al. Security assurance for smart contract[A]. International Conference on New Technologies, Mobility and Security[C]. USA: IEEE, 2018. 1 – 5.

[32] Bang T, Nguyen H H, Nguyen D, et al. Verification of ethereum smart contracts: a model checking approach[J]. International Journal of Machine Learning and Computing, 2020, 10(4): 588 – 593.

[33] Z Yang, H Lei, W Qian. A hybrid formal verification system in Coq for ensuring the reliability and security of ethereum-based service smart contracts[J]. IEEE Access, 2020, (8): 21411 – 21436.

[34] Hoare C A R. An axiomatic basis for computer programming[J]. Communications of the ACM, 1969, 12(10): 576 – 580.

[35] Christel B, Joost-Pieter K. Principles of Model Checking [M]. MIT Press, 2008.

[36] Clarke EM. The birth of model checking[A]. Proc of the 25 Years of Model Checking [C]. Germany: Springer, 2008. 1 – 26.

[37] Z Nehaï, P Piriou, F Daumas. Model-checking of smart contracts[A]. 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)[C]. USA: IEEE, 2018. 980 – 987.

[38] T Abdellatif, K Brousmiche. Formal verification of smart contracts based on users and blockchain behaviors models [A]. 2018 9th IFIP International Conference on New Technologies, Mobility and Security [C]. USA: IEEE, 2018. 1 – 5.

[39] Sistla A P, Gyuris V, Emerson E A. SMC: a symmetry-based model checker for verification of safety and liveness properties[J]. ACM Transactions on Software Engineering and Methodology, 2000, 9(2): 133 – 166.

[40] Bai X, Cheng Z, Duan Z, Hu K. Formal modeling and verification of smart contracts [A]. 2018 7th International Conference on Software and Computer Applications[C]. USA: ACM, 2018. 322 – 326.

[41] Kalra S, Goel S, Dhawan M, Sharma S. ZEUS: analyzing safety of smart contracts [A]. NDSS [C]. USA: ISOC, 2018. 1 – 8.

[42] Antonino P, Roscoe A W. Formalizing and Verifying Smart Contracts with Solidifier: A Bounded Model Checker for Solidity[OL]. https://arxiv. org/abs/2002. 02710, 2020-2-

7/2020-8-20.

[43] Akash L, Shaz Q, Shuvendu K L. A solver for reachability modulo theories[A]. International Conference on Computer Aided Verification[C]. Germany: Springer, 2012. 427 – 443.

[44] Bigi G, Bracciali A, Meacci G, Tuosto E. Validation of decentralised smart contracts through game theory and formal methods[J]. Programming Languages with Applications to Biology and Security, 2015, 9465: 142 – 161.

[45] Alqahtani S, He X, Gamble R, et al. Formal verification of functional requirements for smart contract compositions in supply chain management systems[A]. Hawaii International Conference on System Sciences[C]. USA: University of Hawaii, 2020. 1 – 10.

[46] Scoca V, Uriarte R B, Nicola R D. Smart contract negotiation in cloud computing[A]. 2017 IEEE 10th International Conference on Cloud Computing[C]. USA: IEEE, 2017. 592 – 599.

[47] Bernardi T, Dor N, Fedotov A, et al. WIP: Finding Bugs Automatically in Smart Contracts with Parameterized Invariants[EB/OL]. https://www. certora. com/pubs/sbc2020. pdf, 2020-9-5/2020-10-5.

[48] Biryukov A, Khovratovich D, Tikhomirov S. Findel: Secure derivative contracts forethereum[A]. International Conference on Financial Cryptography and Data Security[C]. Germany: Springer, 2017. 453 – 467.

[49] He X, Qin B, Zhu Y, et al. SPESC: A specification language for smart contracts[A]. 2018 IEEE 42nd Annual Computer Software and Applications Conference[C]. USA: IEEE, 2018. 132 – 137.

[50] Yang Z, Lei H. Lolisa: Formal Syntax and Semantics for A Subset of the Solidity Programming Language[EB/OL]. https://arxiv. org/abs/1803. 09885, 2018-5-27/2020-8-20.

[51] Park D, Zhang Y, Rosu G. End-to-end formal verification of ethereum 2. 0 deposit smart contract[A]. Computer Aided Verification[C]. Germany: Springer, 2020. 151 – 164.

[52] Kosba A, Miller A, Shi E, Wen Z, Papamanthou C. Hawk: the blockchain model of cryptography and privacy-preserving smart contracts[A]. 2016 IEEE Symposium on Security and Privacy[C]. USA: IEEE, 2016. 839 – 858.

[53] Canetti R. Universally composable security: a new paradigm for cryptographic protocols[A]. Proceedings of 42nd IEEE Symposium on Foundations of Computer Science[C]. USA: IEEE, 2001. 136 – 145.

[54] Goldwasser S, Micali S, Rackoff C. The knowledge complexity of interactive proof-systems[A]. The 17th Annual ACM Symposium on Theory of Computing[C]. USA: ACM, 1985. 291 – 304.

[55] Kim H, Laskowski M. A perspective on blockchain smart contracts: reducing uncertainty and complexity in value exchange[A]. The 26th International Conference on Computer Communication and Networks[C]. USA: IEEE, 2017. 1 – 6.

[56] Breidenbach L, Daian P, Tramèr F, Juels A. Enter the hydra: towards principled bug bounties and exploit-resistant smart contracts[A]. The 27th USENIX Conference on Security Symposium[C]. USA: ACM, 2018. 1335 – 1352.

[57] Chen L, Avizienis A. N-version programming: a fault-tolerance approach to reliability of software operation[A]. The 25th International Symposium on Fault-Tolerant Computing[C]. USA: IEEE, 1995. 113 – 120.

[58] Beillahi S M, Ciocarlie G, et al. Behavioral simulation for smart contracts[A]. The 41st ACM SIGPLAN Conference on Programming Language Design and Implementation[C]. USA: ACM, 2020. 470 – 486.

[59] Serge, I, Amrit K, Aquinas H. Scilla: A Smart Contract Intermediate-Level Language[EB/OL]. https://arxiv. org/abs/1801. 00687, 2018-1-2/2020-8-20.

[60] Xu W, Glenn A F. Building Executable Secure Design Models for Smart Contracts with Formal Methods[OL]. https://arxiv. org/abs/1912. 04051, 2019-12-9/2020-8-20.

[61] Mavridou A, Laszka A. Designing secure ethereum smart contracts: a finite state machine-based approach[A]. International Conference on Financial Cryptography and Data Security[C]. Germany: Springer, 2018. 523 – 540.

[62] Wang Y, Lahiri S K, Chen S, et al. Formal verification of workflow policies for smart contracts in Azure blockchain[A]. Working Conference on Verified Software: Theories, Tools, and Experiments[C]. Germany: Springer, 2019. 87 – 106.

[63] Petri C A. Communication with Automata[M]. USA: Rome Laboratory, 1966.

[64] Kurt J. Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use[M]. USA: ACM, 2010.

[65] Liu Z, Liu J. Formal verification of blockchain smart contract based on colored Petri net models[A]. The 43rd Annual Computer Software and Applications Conference[C]. USA: IEEE, 2019. 555 – 560.

[66] Wang D, Huang X, Ma X. Formal analysis of smart contract based on colored Petri nets[J]. IEEE Intelligent Systems. 2020, 35(3): 19 – 30.

[67] Bartocci E, Falcone Y. Lectures on Runtime Verification[M]. Germany: Springer, 2018.

[68] J Ellul, G J Pace. Runtime verification of ethereum smart contracts[A]. 2018 14th European Dependable Computing Conference[C]. USA: IEEE, 2018. 158 – 163.

[69] Ao L, Jemin A C, Fan L. Securing smart contract with

runtime validation[A]. The 41st ACM SIGPLAN Conference on Programming Language Design and Implementation[C]. USA:ACM,2020. 438 -453.

作者简介

朱 健 男,1997 年生于安徽铜陵. 现为北京航空航天大学硕士研究生. 主要研究方向为区块链技术和形式化验证技术.
E-mail:zhujian@ buaa. edu. cn

胡 凯 男,1963 年生于湖南长沙,现为北京航空航天大学计算机学院教授、博士,主要研究方向和关注领域:分布式计算、区块链与数字社会技术.
Email:hukai@ buaa. edu. cn

张伯钧(通讯作者) 男,1997 年生于北京. 现为北京航空航天大学硕士研究生. 主要研究方向为区块链技术和形式化验证技术.
E-mail:zhangbojun@ buaa. edu. cn