以太坊智能合约安全形式化验证方法研究进展

王赫彬[1],郑长友[2],黄 松[2],孙金磊[1],丁一先[3]
(1.中国人民解放军陆军工程大学 指挥控制工程学院,江苏 南京 210007;
2.中国人民解放军陆军工程大学 全军军事训练软件测评中心,江苏 南京 210007;
3.辽宁工程技术大学,辽宁 阜新 123000)

摘 要:以太坊智能合约是区块链技术的典型应用和实现。由于智能合约一旦部署就难以修改,智能合约在上链之前的正确性显得至关重要。虽然很多当前工作都对于智能合约漏洞的检测与预防进行了一系列研究,但是对于检查合约属性和范式的形式化验证方法还没有比较全面的总结。首先介绍了智能合约基本的表示和编译方式、常用的编程语言、出现的漏洞类型和常见的形式化验证方法,并分析了形式化验证方法在智能合约方面的研究现状;然后,通过实验结果分析验证了两种现有的有界模型检测工具的检测好坏,对于某些具体的合约攻击,漏报率达到30%以上,现有形式化验证方法漏报率高、应用范围受限以及验证语言安全性未知等一系列缺陷亟待解决;最后,针对现有形式化验证方法存在的不足之处展望了未来的研究方向。

关键词:以太坊;智能合约;形式化验证;定理证明;模型检测
中图分类号:TP309;TP311 **文献标识码**:A **文章编号**:1673-629X(2021)09-0104-08
doi:10.3969/j.issn.1673-629X.2021.09.018

Review:Secure Formal Verification Methods for Ethereum Smart Contracts

WANG He-bin[1],ZHENG Chang-you[2],HUANG Song[2],SUN Jin-lei[1],DING Yi-xian[3]
(1.School of Command and Control Engineering,Army Engineering University of PLA,Nanjing 210007,China;
2.Military Software Training and Evaluation Center,Army Engineering University of PLA,Nanjing 210007,China;
3.Liaoning Technical University,Fuxin 123000,China)

Abstract:Ethereum smart contract is a typical application and implementation of blockchain technology. Since a smart contract is difficult to be modified once it is deployed,thus it is important for coders to check smart contracts' correctness before putting on the chains. Although there are many current researches of the vulnerability detection and prevention of smart contracts,no complete summary of the correctness and feasibility of concepts and properties of smart contracts,such as user interactions and specific technical concepts,are carried out. Firstly,basic representation and compilation,frequently-used programming languages,vulnerabilities of smart contracts and common formal verification methods were introduced. Secondly,the effectiveness of the two existing bounded model checking tools is checked and reproduced through the analysis of experiment results,false negative rate reaches 30%,and it is urgent to correct weaknesses of the existing formal verification methods,such as high rate of false positives,limited application scopes,unknown security of the verification language and so on. Finally,the future research directions are prospected according to the limitations of these existing researches of formal verification methods.

Key words:Ethereum;smart contract;formal verification;theorem prove;model checking

0 引 言

自2009年Satoshi Nakamoto的比特币发布以来,出现了各种各样的区块链实现。而自从2013年底Vitalik Buterin发布的以太坊白皮书将智能合约的概念引入到区块链之中,区块链技术的应用变得更加广泛,并使得比特币等加密货币的可编程能力更进一步。智能合约提供了一个框架,允许以可信任、去中心化的方式执行一个应用程序的业务逻辑。智能合约用户通常

收稿日期:2020-08-15 **修回日期**:2020-12-16
基金项目:国家重点研发计划重点专项项目(2018YFB1403400)
作者简介:王赫彬(1996-),男,硕士研究生,研究方向为软件测试、区块链安全;通讯作者:郑长友(1986-),男,讲师,研究方向为软件测试、区块链;通讯作者:黄 松(1970-),男,博士,教授,博导,研究方向为软件测试和质量评估。

携带大量的价值数以亿计的货币,但是智能合约作为一种计算机代码可能具有未发现的和未知的漏洞,具有很大的安全隐患。2016 年 6 月 17 日的 The DAO 攻击事件中,一名攻击者利用去中心化自治组织(decentralized autonomous organization,DAO)部署的智能合约代码中的多个漏洞,控制了当时价值约为 5 000 万美元的以太坊网络,并从合约中盗走了 350 多万以太币;2018 年美链 BEC 上百亿项目归零、BAI(blockchain of artificial intelligence and internet of things)任意账户无限转账都是因为 ERC20 代币标准中出现的一系列漏洞,导致资金恶意发送转移,正常的矿工节点受到了危害。对于难以避免的智能合约安全性问题来说,在部署链上前发现并修复漏洞至关重要。

虽然现有的一些工作通过基于测试的检测方法来检测漏洞,但不能完备地发现智能合约源码和字节码中所有的漏洞,而且对于智能合约在一些概念和属性的正确表达缺乏全面的检查。而现有的一些代码及合约审计工作则依赖于审计专家的个人经验和主观判断,缺乏严谨性和说服力。形式化验证方法则通过证明序列、逻辑语句检查数学模型的逻辑,以证明某些规范的正确性并发现运行时错误,理论上可以发现更多的漏洞。智能合约对于安全性要求高,代码量相对较小,这些特点很适用于进行小型的形式化验证。文中对以太坊智能合约的相关概念进行了系统的介绍,分析了智能合约常见的漏洞类型,总结了常见的形式化验证方法,进行了实验结果分析并讨论了目前形式化验证工作存在的问题和挑战,提出了将来的改进方法及意见建议。

1　背景及相关概念介绍

1.1　以太坊智能合约

以太坊平台于 2015 年提出,是使用区块链平台执行智能合约的最流行的底层系统之一[1]。其市场容量达到了 1 000 亿美元,每天处理超过 40 万笔资金交易。以太坊包含以太坊虚拟机、执行环境(账户管理、默克树等)和共识算法三个部件,适用于公有链、私有链和联盟链三种区块链环境。不同的区块链环境可以通过扩展包的形式,将以太坊智能合约部署到链上。以太坊智能合约是部署在其平台上的一种计算机协议,具有可编程、可维护和安全可信等特点[2]。特别地,智能合约的执行过程是从一个状态到另一个状态[3],可以用有限状态机将智能合约形式化地表示为一个五元组自动机 Q^*:

$$Q^* = (Q_0, \Omega, \zeta, t^*, G^*)$$

其中,Q_0 是合约执行自动机的所有状态的集合,Ω 为所有输入事件的集合,$\zeta: Q_0 \times \Omega \rightarrow Q_0$ 则是转移函数的集合,$t^* \in Q_0$ 为初始状态,$G^* \subset Q_0$ 为终止状态集合。

```
pragma solidity ^0.4.24;
contract HashForEther {
    function withdrawWinnings() {
        // Winner if the last 8 hex characters
of the address are 0.
        require(uint32(msg.sender) == 0);
        _sendWinnings();
    }
    function _sendWinnings() {
        msg.sender.transfer(this.balance);
    }
}
```

PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH2 0x11D DUP1 PUSH2 0x20 PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN STOP PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH1 0x49 JUMPI PUSH1 0x0 CALLDATALOAD PUSH29 0x100000000000000000000000000000000000000000000000000000000 SWAP1 DIV PUSH4 0xFFFFFFFF AND DUP1 PUSH4 0x83AC4AE1 EQ PUSH1 0x4E JUMPI DUP1 PUSH4 0xCC42E83A EQ PUSH1 0x62 JUMPI JUMPDEST PUSH1 0x0 DUP1 REVERT JUMPDEST CALLVALUE DUP1 ISZERO PUSH1 0x59 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x60 PUSH1 0x76 JUMP JUMPDEST STOP JUMPDEST CALLVALUE DUP1 ISZERO PUSH1 0x6D JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x74 PUSH1 0xD5 JUMP JUMPDEST STOP JUMPDEST CALLER PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH2 0x8FC ADDRESS PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND BALANCE SWAP1 DUP2 ISZERO MUL SWAP1 PUSH1 0x40 MLOAD PUSH1 0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB DUP2 DUP6 DUP9 DUP9 CALL SWAP4 POP POP POP POP ISZERO DUP1 ISZERO PUSH1 0xD2 JUMPI RETURNDATASIZE PUSH1 0x0 DUP1 RETURNDATACOPY RETURNDATASIZE PUSH1 0x0 REVERT JUMPDEST POP JUMP JUMPDEST PUSH1 0x0 CALLER PUSH4 0xFFFFFFFF AND EQ ISZERO ISZERO PUSH1 0xE9 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST PUSH1 0xEF PUSH1 0x76 JUMP JUMPDEST JUMP STOP LOG1 PUSH6 0x627A7A723058 KECCAK256 0xdc PUSH29 0xA0FFAD7FFC8EA3A8F83AADC01BF3EF0B967698443B7ED820607ABE3B8A 0xda STOP 0x29

图 1　智能合约代码编译过程示例

1.1.1　以太坊虚拟机

智能合约通常用有别于现有编程语言的高级语言编写,开发人员在编写完智能合约代码后,代码被编译成以太坊虚拟(Ethereum virtual machine,EVM)字节码在计算机上执行。EVM 是一种完全基于堆栈的虚拟机架构,支持一个包含 134 个操作码的指令集,运行在 256 比特的基本数据类型之上。与 Java 虚拟机(Java virtual machine,JVM)相比,EVM 相对更加简单,执行效果更加具有确定性。图 1 是一个合约代码被 EVM 编译成字节码的示例。

1.1.2　智能合约语言

智能合约通常使用专门的智能合约语言进行编写[4]。智能合约语言包含高级语言、中间级语言(intermediate representation,IR)和低级语言三种类型。开发人员通常使用高级语言编写智能合约,因为高级语言能够提供期望的表达方式。IR 语言介于高级语言和低级语言之间,可以用于编写程序以推导属性和检查规范,也可以用于优化代码的结构。而低级语言则

提供了一种在分布式虚拟机上执行的实现方式。基于合约语言不同的安全特性,可以采用不同的方法对合约进行形式化验证。其中 Solidity、Vyper 是以太坊平台常用的智能合约高级语言,EVM 是其常用的低级语言,而 EthIR 和 Yul 是其常用的中间语言。

1.2 常用的形式化验证方法

形式化验证方法[5]是指在形式化建模的基础上,对具有精确语法和语义的形式化语言的系统建立事务及性质等的关系,从而验证系统所应具有的功能正确性和其他关键性质。

1.2.1 定理证明

定理证明[6]适用于很多系统,方法流程是对其进行形式化规约,并将要证明的命题写为“待验证系统满足形式化规约”,从而将验证问题转化为证明一个可满足性问题,使用预先定义的公理和推理规则作为输入来完成证明过程。定理证明分为交互式定理证明、自动化定理证明和混合定理证明三种[7],而不同类型系统的复杂度决定了不同定理证明方式的可行性。复杂度较小的可以使用自动化定理证明,复杂度较大的则可以使用交互式定理证明。

1.2.2 模型检测

模型检测[8]是指对于一个协议框架利用逻辑公式进行检查,观察该协议描述的需求或系统是否具有某些期望的性质。通常需要验证的性质包括安全性(safety)、活跃性(liveness)、无阻性(non-blocking)和非严格顺序性(no strict sequencing)等。该方法适用于可以用有限状态模型表示的系统。模型检测器可以自动化完成精准模型和规范的检测,给出可用的判断结果。

1.3 常见的漏洞类型

随着智能合约语言的迭代更新,不断有新的漏洞被修复,但同时新的版本语言的语法也会带来新的安全漏洞隐患。截至 2020 年 8 月,Solidity 语言从 0.3.x 版本更新到了 0.7.0 版本。按照漏洞来源依赖,以太坊智能合约漏洞可以分为区块链依赖漏洞、软件安全依赖漏洞和以太坊及合约语言依赖漏洞[9]。图 2 是一些以太坊智能合约的常见漏洞。

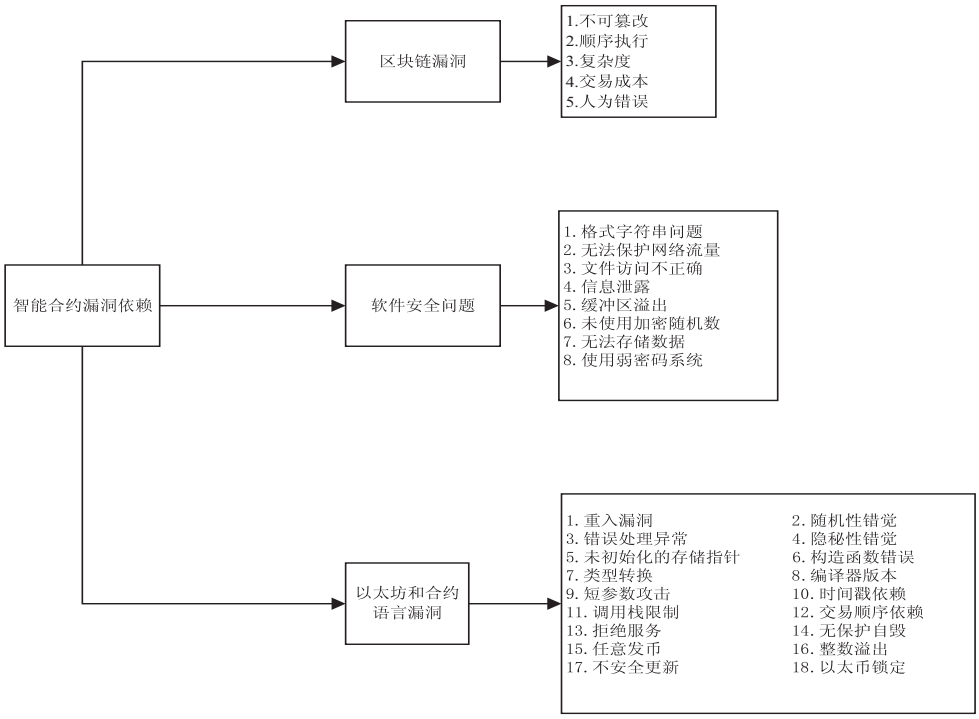

图 2 智能合约漏洞依赖分类

2 相关研究进展

本节将介绍智能合约形式化验证的一些最新技术研究进展。暂时没有一个固定的标准对于这些形式化验证方法进行分类,但是已经有很多研究对于智能合约的正确设计和漏洞挖掘产生了效果。借鉴了 1.2 节传统的形式化验证方法,可以采用以下几种技术手段对智能合约进行形式化验证。

2.1 类型推断

基于类型推断的形式化验证使用数十行或数百行的核心类型理论来替换智能合约语言的类型或类型子集,能够用来快速证明数学定理。这种方法使用函数式程序语言,更多地关注类型检查,而不是运行程序,具有相对简单的语法。这种方法的一般流程为:首先将智能合约高级语言翻译成函数式程序语言,指定源语言中的数据在验证语言中的数据格式和数据类型,

并转换包含这些数据的源语言中的函数;然后提出并证明关于这些函数的定理,可以完成运行时程序正确性的验证。

微软研究院和哈佛大学研究团队的 Bhargavan 等人[10]提出了一个用于形式化分析的 F* 框架,试图通过将合约代码翻译成 F* 语言后,通过类型推断来验证智能合约。该框架包括 Solidity* 和 EVM* 两个工具原型,Solidity* 支持 Solidity 的部分语法类型转换,将智能合约从 Solidity 语言翻译成 F* 语言,EVM* 再从 EVM 字节码形式反编译成 F* 语言。转换过程中支持递归,不支持循环。Solidity* 运用类型推导发现运行时函数和变元的错误和违反的规范,EVM* 则可以分析低层次的属性,估算 gas 消耗的上界。验证了 396 份合约,其中 46 份成功地进行了翻译和类型检查,发现了一些可重入漏洞和未经检查的 send 语句返回调用问题。

Coblenz 等人[11]研发的 Obsidian 是一种面向类型的安全区块链编程语言,该语言以用户为中心,用户实验反馈与语言开发同时进行,类型状态依赖系统保证了程序员编写正确的合约。针对可用操作依赖的高级状态、代码错误导致的资金损失或挪用和不一致状态导致的重入攻击等三种严重的 bug 源,类型状态会将动态信息记录到类型之中,之后进行静态的类型推断,同时,线性类型系统计算货币的数量,完成对于交易过程中的财务信息追踪溯源。

Schrans 等人[12]开发了一种专门用于编写鲁棒性智能合约的静态类型编程语言 Flint,引入保护块(protection blocks)根据类型状态限制运行代码的时间和用户,对于状态的写入进行了限制,简化了合约的推理。具有调用保护、默认不变量、资产类型推理和安全语义等安全特性,实现了安全的内部及外部资金转移(asset transfer)、事件监控和安全的算术运算等代码保护功能,通过限制交易过程中的类型状态,将 Solidity 语言的安全性进行了提升,避免了智能合约重入漏洞和整数溢出等问题的出现。目前,智能合约交互语言 Rholang[13]和 Plutus[14]分别使用 ρ 演算和 λ 演算等抽象演算方法,完成了一些验证工作。

2.2　交互式定理证明

智能合约的交互式定理证明是传统的定理证明方法在智能合约验证中的应用。智能合约高级语言如 Solidity 和低级语言如 EVM 上的字节码需要被定义成可以为定理证明器编译的语言,然后再利用交互式定理证明验证以太坊智能合约的安全性质。定理证明器也可以选择用一些自动化 SMT 定理证明器,SMT 求解器使用一阶逻辑语言来表示命题,然后通过枚举赋值的方式进行证明。可满足命题的赋值为程序提供正确的输入,可以证明在执行路径中给定的状态是可达的。

Amani 等[15]研发的 eth-isabelle 是一种作用于 EVM 字节码的 Isabelle/HOL 定理证明器。首先将 EVM 字节码序列语义用 Lem 语言形式化构造成线性代码块,将 EVM 指令划分为以 JUMPDEST 为开始的基本块、JUMP、JUMPI 等跳转指令以及其他指令基本块三种,然后再翻译到 Isabelle/HOL 或 Coq 中进行定理证明。基于以太坊的 gas 机制给出了一种 out-of-gas 异常处理方法,使用分离逻辑(separation logic)验证以太坊虚拟内存中的字节码程序,支持使用程序规则、基本块规则和指令规则自动生成验证条件和验证规范,降低了 EVM 操作码形式智能合约形式化验证的时间复杂度。

Lei 等[16]提出的 Lolisa 是 Solidity 语言的一个大型子集的形式化语法和语义,采用了更鲁棒的静态类型系统,不仅包含了 Solidity 中映射符、修改器、合约类型和地址类型等语法组件,而且还包含了诸如多个返回值、指针算法以及结构和字段访问等通用编程语言特性。基于 Solidity 语言的可扩展性,提出将 Lolisa 扩展到其他合约编程语言的方案。接着开发了一种支持不同验证范式的可扩展、可重用的形式化内存框架 GREM,模拟了物理内存的硬件结构,并且使用 Coq 提供了一套防干扰的应用程序编程接口(application programming interfaces,API),实现了一个独立可定制的框架。之后开发的执行时验证同构体(execution verification isomorphism,EVI)将符号执行与定理证明结合,提高了高阶逻辑定理证明辅助工具的自动化程度,并且证明了 Lolisa 可满足于 EVI。

Yang 等人[17]开发的 FEther 编译器是一种基于 Coq 的混合定理证明工具,基于 Solidity 语言的形式化语法和语义子集 Lolisa,结合符号执行和高阶逻辑定理证明,提高了执行和验证合约的自动化程度,同时 FEther 具有很好的可重用性,可以将验证过的代码片段复用检查其他代码段的属性,其运行效率远远超过 Coq 标准教程中的解释器。是第一个在 Coq 中对于 Solidity 语言进行语法和语义定义的证明引擎。Annenkov 等人[18]提出了一个基于 Coq 的框架 ConCert,可以对于智能合约进行形式化证明鉴定。

Sergey 等[19]研发的中间语言 Scilla 提供了一个编程组件,为智能合约间的通信提供了一个计算与交互的分离、效率和纯计算的分离及调用动作和连续过程的分离等,允许丰富的交互模式,具有规则形式写成的语义,将 Scilla 嵌入 Coq 之中,易于进行形式化验证。描述了基于自动机[20]的 Scilla 模型,展示了自动机关于安全和时间等属性的简化验证过程。除了 Coq 和

Isabelle/HOL 等工具外，Why3[21] 也实现了 EVM 的语义表示，可以基于 EVM 字节码进行一些智能合约的定理证明。

2.3 有界模型检测

智能合约的模型检测中常用的方法为有界模型检测（bounded model check，BMC），有界模型检查器系统地、象征性地研究系统的行为，直到某个边界 k，寻找对给定属性的违反，其中边界 k 表示允许从某个初始状态进行的转换的数量。使用单个逻辑语句检查智能合约的性质及规范，若合约状态转移符合规范，则证明智能合约满足所验证的安全特性；否则构造模型检测器会构造一个状态转移的反例，证明合约违反了该条规范，存在安全性问题。

牛津大学的 Antonino 等[22] 开发了一种基于 Solid 语言的有界模型检测工具 Solidifier，Solid 语言对于 Solidity 语言和以太坊区块链进行形式化描述，之后将 Solid 转换成中间语言 Boogie[23]，使用 Corral[24] 对于 Boogie 语言形式的合约进行有界模型检测。提出合约验证和函数验证两种工具，定义 main 过程和捕获 Solid 调用结构的 callP 过程，探索区块链的行为，寻找合约错误。通过特定代码模式匹配或未预料到的行为方式，查找不符合开发人员意图的程序错误及状态。

微软 Azure 区块链的 Wang 等[25] 研发的 VeriSol 可以用来分析企业级 Azure 区块链服务工作台上的所有合约，对于 BaaS 产品工作台上的应用程序所用的规定语言进行了形式化建模，给定应用策略 $\pi = (\dot{M}, \dot{N})$ 和工作流 $\eta = < T, t_0, \dot{M}_\eta, G, G_0, bd_0, \gamma > \in \dot{N}$，使用霍尔三元组完成初始化和跃迁。利用 Solidity 语言中的 modifier 修改器包装函数调用，构造进行检测，定义了具体的规定和合约的语义一致性检查问题（semantic conformance verification，SCV）。然后对于包括被受信任的 PoA 合约在内的所有 Solidity 0.5.x 版本语言写成的应用程序进行了验证，报告了一些之前未知的错误。

Joel 等[26] 开发了一种基于符号执行的有界模型检测工具 ETHBMC，设定了 Keccak256 函数、Memcopy-like 指令和 Inter-Contract 交互等测试断言，建立了以太坊的攻击模型、内存模型和数据调用模型等，然后符号执行检测模块中的漏洞，对于每个具有易受攻击的路径生成交易过程实例。通过对于当前 go-ethereum 区块链工具套件中活跃的大约 220 万个用户大规模分析，自动生成 5 905 个触发漏洞的有效输入，与之前的分析方法相比较，具有 22.8%的更多输入。

Zhang 等[27] 开发了一个用 4 800 多行 Python 和 C++写成的有界模型检测工具 NPCHECKER，处理 Solidity 及 Vyper 语言的不确定性对于合约付款等交易过程的影响。发现由于不可预测的事务调度和外部未知的被调用者行为而读写更改的合约全局变量，公开了各种不确定性因素和问题。NPCHECKER 的工作流程为：首先将 EVM 字节码反编译，然后利用内联汇编将汇编指令转换成 LLVM，恢复控制流图，最后使用 SMACK[28] 和 Boogie 等模型检测工具检查恢复的 IR 代码中是否含有 NP_{I} 和 NP_{II} 两种付款 bug。在以太坊主网上收集到 3 万多份在线合约并对其进行了评估。NPCHECKER 在 1 111 份合约中检测到不确定性支付，识别了六种未知的新漏洞类型。目前，NuSMV[29]、SPIN[30] 等传统的模型检测工具被用于能源交易市场、网上购物等智能合约，完成需求规范开发阶段和最终系统验证集成阶段的交互验证。

2.4 其他验证方法

除了以上研究深入广泛的以太坊智能合约形式化验证方法外，根据以太坊智能合约框架的性质用途和工作流程，还有很多结合紧密的验证方法产生更加具有领域自适应性的应用。本节介绍几种在主流的智能合约形式化验证之上有所改进的方法。

2.4.1 用户和区块链行为建模

传统的模型检测方法可以对智能合约及其用户进行建模，但是不能对底层的区块链进行建模。Abdellatif 等[31] 提出了一种对于区块链及其用户进行建模的方法，按照行为交互优先级（behavior interaction priorities，BIP）使用统计型模型检测器（statistical model checking，SMC）进行验证。对于给定的系统 B 和性质 δ，SMC 模型检测器使用概率有界线性时序逻辑（probabilistic bounded linear time logic，PB-LTL）来判断 B 是否满足 δ。令 B 满足 δ 记为 $B \models \delta$，首先判断公式：

$$P(\{B \models \delta\}) > \theta$$

是否成立，其中 $P(.)$ 为某个事件发生的概率，θ 为某个选定的阈值；然后估算 $P(\{B \models \delta\})$ 的值 p，使得估计值 p_0 满足：

$$P(|p - p_0| \leqslant \alpha) \geqslant 1 - \beta$$

其中，α 为估计精度，β 为攻击风险水平。运用该方法能够发现传统模型检测方法难以发现的易受攻击用户行为和交易路径。

2.4.2 博弈论

Giancarlo 等[32] 提出了一种利用博弈论（game theory）来对涉及物理行为的智能合约进行验证的方法。首先通过博弈论分析交易过程中顾客和商家的行为，观察两方行为的相互影响，然后使用概率模型检测器 PRISM 通过分析顾客和商家不确定的选择行为对于选择事件对应的概率的影响，将行为建模成离散时间马尔可夫决策过程（discrete time Markov decision

process,DTMDP),来解决不确定性问题。根据顾客的诚实度和商家的信誉使用了六种存款与出售价格比率,对于一组诚实的顾客与不诚实的商家,商家损失30%的可能性不大于2%,顾客损失30%的可能性却可以达到50%。

2.4.3　着色 Petri 网

Wang 等[33]改进了字节码的程序逻辑规则,应用霍尔条件(Hoare condition)建立了着色 Petri 网(colored Petri Nets,CPN)模型。首先,对于 Solidity 源代码形式的智能合约,利用 ANTLR 语法分析器建立抽象语法树(abstract syntax tree,AST)模型并解析抽象语法树中的元素,生成等价的 CPN 模型;然后,对于 EVM 字节码形式的智能合约,对于字节码序列对应的操作语义进行静态分析,划分出具有不同特征的基本块,构建控制流图(control flow graph,CFG),生成等价的 CPN 模型。最后,将之前建立的 CPN 模型放入 CPN 工具中完成动态仿真和模型检测。该方法设计了一种自动化的建模方法,引入了一个自定义调用库和一种基于回溯的路径推导算法,提高了传统模型检测方法的针对性和有效性。

2.4.4　K 框架

K 框架[34]是一种可执行的程序框架语义集,可以用于形式化智能合约,对事务进行状态和网络演化进行建模,并使用转换规则来详细阐述状态的修改和网络的演化。K 框架在逻辑上可以分成三个模块:data.k,language.k 和 ethereum.k,目的是与以太坊的依赖结构保持一致。Hildenbrandt 等[35]在 K 框架中定义了一个 EVM 的语义—KEVM,将 EVM 表示成 K 框架范式中的 ENBF 样式提供的语言语法、状态配置描述和驱动程序执行的转换规则三个主要组件,深入解释了 KEVM 的结构定义描述,然后给出了几种语义扩展,根据 ERC20 标准验证了标准代币的两种不同语言实现,使用 K 框架的可达性逻辑证明器,表现了 KEVM 语义框架的可用性。此外,该框架还可用于 gas 消耗分析和解析以太坊白皮书中 EVM 的语言规范等。KVyper[36]由 Vyper 到 LLL、LLL 到 EVM 两个翻译器组成,可用于发现 Vyper 编译器中存在的错误。KSolidity[37]是 K 框架中 Solidity 语义的定义,可用于 Solidity 语言合约测试。

除此之外,ZEUS[38]和 Oyente[39]也使用形式化验证的方法分别在源码级和操作码级进行了自动化的漏洞检测,并发现了一些以太坊智能合约存在的安全漏洞。

3　总结与展望

3.1　现有研究存在的不足

如第二节所述,智能合约形式化验证工具发展迅速,但是仍有一些缺陷,主要问题如下:

(1)合约语言不完整。不同的形式化验证方法通常能够验证合约语言的一个子集,而不能完整地验证所有智能合约可能出现的问题。总结一些常用形式化验证方法对于智能合约的处理,如表 1 所示。

表 1　智能合约形式化验证框架/工具对比分析

验证框架/工具	合约语言类型	验证方法	验证范围		验证机制		漏洞类型
			属性	语法	逻辑范式	Gas 机制	
F*	源码、操作码	类型推断	部分属性	部分语法	不支持	支持	重入漏洞、未处理的异常
Flint	源码	类型推断	部分属性	部分语法	不支持	支持	整数溢出
eth-isabelle	操作码	定理证明	全属性	部分语法	支持	支持	未检查的返回值
FEther	源码	符号执行、定理证明	全属性	部分语法	支持	不支持	委托调用滥用
KEVM	操作码	定理证明	全属性	所有语法	支持	支持	
Oyente	字节码	符号执行、模型检测	部分属性	所有语法	不支持	不支持	时间戳依赖、交易顺序依赖等

可以看出,很多语言只能处理合约语言的一个语义语法子集,如 FEther 基于 Solidity 语言的 Lolisa 子集进行证明,不能对整个 Solidity 语言进行验证,查找漏洞的手段较少,发现漏洞的类型不足。有些待验证的规范可能包含类型特性子集之外的语义语法元素,导致难以验证智能合约满足的性质及规范。扩展可验证

的合约语言子集,建立更多的验证范式,发现更多的漏洞类型是一个需要解决的技术问题。

(2)应用范围受限。由于智能合约仍是一个较新的领域,因此形式化验证缺乏一个统计评价标准和最佳实践,方法应用没有突破高昂成本、合约规模等技术局限。文中选取了 106 个 Solidity 源码合约样本结合偷以太币(Steal Ether)、黑杰克(Hijack)和自毁(Suicidal)三种攻击行为,对 Solidifier 和 ETHBMC 两种有界模型检测工具检查效果进行对比,其编译通过率和漏报率结果如表 2 所示。三种工具都存在较多的问题漏报的情况。此外,形式化验证还只适用于简化的合约模型,采用的启发式方法难以扩展到复杂的合约中去。自动化程度不高,且缺乏对于高阶业务逻辑范式的证明。

表 2 模型检测结果比较

攻击类型	工具	编译通过率	漏报率
Steal	ETHBMC	0.80	0.18
	Solidifier	0.82	0.14
Hijack	ETHBMC	0.85	0.17
	Solidifier	0.84	0.03
Suicidal	ETHBMC	0.92	0.32
	Solidifier	0.90	0.16

(3)验证语言安全性未知。在验证的过程中,通常需要将源代码形式或字节码形式的合约翻译成另一种语言,例如 F* 方法,但是验证语言的安全性没有得到保证,具有很大的安全隐患。翻译过程的正确性也是未知的,很有可能出现转换前后的类型有很大差别。因此证明或提高验证语言的安全性及翻译过程的正确性是形式化验证面临的一些重难点工作。

3.2 未来研究方向和改进思路

针对现有工具存在的问题,提出如下未来改进思路:

(1)扩展形式化验证方法的应用范围。由于缺乏测试判定准则(test oracles),验证好坏通常难以衡量。所以需要建立标准的测试判定用例集,对于验证方法的有效性进行定性定量评估。通过建立综合评价指标体系,找出实践性好的验证方法及工具,有助于在此基础上对智能合约的形式化验证进行改进和提高。通过与符号执行、污点分析等方法的结合,提高形式化验证方法的自动化程度,创建可重用的验证模式库,减少验证所需时间,增强合约验证方法的可用性。

(2)提高形式化验证方法的验证效果。针对智能合约形式化验证过程不直观的问题,可以通过类型标记、着色 Petri 网等增强合约验证的可视化效果,利用逆向工程重新构造合约执行的控制流图。根据以太坊智能合约具有的一些性质和用途,建立具有严谨可靠数学基础的区块链模型和智能合约机制模型,提高以太坊平台合约验证的效果。

4 结束语

以太坊为代表的智能合约平台的出现扩展了区块链的资金转移能力,促进了区块链 2.0 时代的到来。文中梳理了以太坊智能合约常用的语言和常见的漏洞类型,讨论了基于类型推断的形式化验证、智能合约的交互式定理证明和智能合约的有界模型检测等主流的智能合约形式化验证方法,通过对几种现有形式化验证工具的实验结果分析研究发现,并针对现有方法的不足之处,提出了未来研究的改进思路。下一步工作的重点和难点在于根据以太坊智能合约的实现机制、环境状态和网络演化等内部性质用途,提出具有自适应性的理论着力提高验证的准确性,减少时间和空间成本,扩展验证的适用范围,进一步提高以太坊智能合约的安全开发部署能力。

参考文献:

[1] 毕晓冰,马兆丰,徐明昆.区块链智能合约安全开发技术研究与实现[J].信息安全与通信保密,2018(12):63-73.

[2] 马春光,安 婧,毕 伟,等.区块链中的智能合约[J].信息网络安全,2018(11):8-17.

[3] BAI X,CHENG Z,DUAN Z,et al.Formal modeling and verification of smart contracts[C]//International conference software and computer applications. New York: Association for Computer Machinery,2018:322-326.

[4] DWIVEDI V,DEVAL V,DIXIT A,et al.Formal-verification of smart-contract languages:a survey[C]//International conference on advances in computing and data sciences. Singapore:Springer International Publishing,2019:738-747.

[5] 胡 凯,白晓敏,高灵超,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.

[6] HASAN O,TAHAR S.Formal verification methods[M].3rd ed.Hershey:IGI Global,2015:7162-7170.

[7] 方 言.基于以太坊智能合约的形式化验证技术研究与实现[D].成都:电子科技大学,2019.

[8] MILLER A,CAI Z,JHA S,et al.Smart contracts and opportunities for formal methods[C]//Leveraging applications of formal methods, verification and validation. Limassol, Cyprus:Springer International Publishing,2018:280-299.

[9] PRAITHEESHAN P,PAN L,YU J,et al.Security analysis methods on Ethereum smart contract vulnerabilities:a survey[J].arXiv:1908.08605,2019.

[10] BHARGAVAN K,DELIGNATLAVAUD A,FOURNET C,et al. Formal verification of smart contracts: short paper[C]//ACM workshop on programming languages and analysis for security. New York: Association for Computer Ma-

chinery.2016:91-96.

[11] COBLENZ M. Obsidian: a safer blockchain programming language[C]//International conference on software engineering.Washington DC:Institute of Electric and Electronics Engineers Computer Society,2017:97-99.

[12] SCHRANS F,HAILS D,HARKNESS A,et al.Flint for safer smart contracts[J].arXiv:1904:06534,2019.

[13] MEREDITH L G,PETTERSSON J,STEPHENSON G,et al. Contracts,composition,and scaling: the rholang specification [EB/OL].(2018-02-27)[2020-06-22].https://developer.rchain.coop/assets/rholang-spec-0.2.pdf.

[14] PEYTON M,CHAPMAN J,JENKINS K,et al.input-output-hk/plutus[EB/OL].(2016-09-13)[2020-03-31].https://github. com/input-output-hk/plutus.

[15] AMANI S,BEGEL M,BORTIN M,et al.Towards verifying ethereum smart contract bytecode in Isabelle/HOL[C]//7th ACM SIGPLAN international conference.[s. l.]: ACM, 2018:66-77.

[16] YANG Z,LEI H.Lolisa:formal syntax and semantics for a subset of the solidity programming language[J].arXiv:1803: 09885,2018.

[17] YANG Z,LEI H.FEther:an extensible definitional interpreter for smart-contract verifications in Coq[J].IEEE Access, 2019,7:37770-37791.

[18] ANNENKOV D,NIELSEN J B,SPITTERS B,et al. ConCert:a smart contract certification framework in Coq[J]. arXiv:1907:10674,2020.

[19] SERGEY I,KUMAR A,HOBOR A.Scilla:a smart contract intermediate-level language[J].arXiv:1801:00687,2018.

[20] LYNCH N,TUTTLE M R.An introduction to input/output automata[J].CWI Quarterly,1989,2(3):219-246.

[21] MARCHE C,MELQUIOND G,FILLIATREJ C,et al.AdaCore/why3[EB/OL].(2009-11-29)[2018-12-18].https://github.com/Ada Core /why3.

[22] ANTONINO P,ROSCOE A W.Formalising and verifying smart contracts with Solidifier:a bounded model checker for Solidity[J].arXiv:2002:02710,2020.

[23] BARNETT M,CHANG B Y E,DELINE R,et al.Boogie:a modular reusable verifier for object-oriented programs [C]//International symposium on formal methods for components and objects.[s.l.]:Springer International Publishing, 2005:364-387.

[24] HARAN A,CARTER M,EMMI M,et al.SMACK+Corral:a modular verifier[C]//International conference on tools and algorithms for the construction and analysis of systems.[s. l.]:Springer International Publishing,2015:451-454.

[25] FRANK J,ASCHERMANN C,HOLZ T.{ETHBMC}:a bounded model checker for smart contracts[C]//29th {USENIX} security symposium ({USENIX} security 20). Alameda:Berkeley,2020:2757-2774.

[26] WANG Y,LAHIRI S K,CHEN S,et al.Formal specification and verification of smart contracts for Azure blockchain[J]. arXiv:1812:08829,2018.

[27] WANG S,ZHANG C,SU Z,et al.Detecting nondeterministic payment bugs in Ethereum smart contracts[C]//Conference on object-oriented programming systems,languages,and applications.[s. l.]: Association for Computer Machinery, 2019:1-29.

[28] RAKAMARIC Z,EMMI M.SMACK:decoupling source language details from verifier implementations[C]//Computer aided verification.Washington DC:Institute of Electric and Electronics Engineers Computer Society,2014:106-113.

[29] NEHAI Z,PIRIOU P,DAUMAS F,et al.Model-checking of smart contracts[C]//Green computing and communications. Washington DC: Institute of Electric and Electronics Engineers Computer Society,2018:980-987.

[30] OSTERLAND T,ROSE T.Model checking smart contracts for Ethereum[J].Pervasive and Mobile Computing,2020, 63:101129.

[31] ABDELLATIF T,BROUSMICHE K.Formal verification of smart contracts based on users and blockchain behaviors models[C]//New technologies mobility and security.[s.l.]: Springer International Publishing.2018:1-5.

[32] BIGI G,BRACCIALI A,MEACCI G,et al.Validation of decentralised smart contracts through game theory and formal methods[M].[s.l.]:Springer International Publishing,2015: 142-161.

[33] DUO W,XIN H,XIAOFENG M.Formal analysis of smart contract based on colored petri nets[J].IEEE Intelligent Systems,2020,35:19-30.

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

[35] HILDENBRANDT E,SAXENA M,RODRIGUES N,et al. KEVM:a complete formal semantics of the ethereum virtual machine[C]//IEEE computer security foundations sym- posium.Oxford,UK:IEEE,2018:204-217.

[36] ZHANG Y,BOGDANAS D,PARK D,et al.kframework/vyper-semantics[EB/OL].(2017-10-22)[2020-08-10]. https://github.com/kframework/vyper-semantics.

[37] FILARETTI D,HILDENBRANDT E,LIN S,et al.kframework/solidity-semantics[EB/OL].(2018-05-27)[2020-05-22].https://github. com/kframework/solidity-semantics.

[38] KALRA S,GOEL S,DHAWAN M,et al.ZEUS:analyzing safety of smart contracts[C]//Network and distributed system security symposium.Washington DC: Internet Society, 2018:1-15.

[39] LUU L,CHU D,OLICKEL H,et al.Making smart contracts smarter[C]//Computer and communications security.[s. l.]:Association for Computer Machinery,2016:254-269.