信息安全研究 第2卷 第12期 2016年12月
Journal of Information Security Research Vol.2 No.12 Dec. 2016

智能合约的形式化验证方法

胡 凯[1] 白晓敏[1] 高灵超[2] 董爱强[2]
[1](北京航空航天大学计算机学院 北京 100191)
[2](北京中电普华信息技术有限公司 北京 100192)
(hukai@buaa.edu.cn)

Formal Verification Method of Smart Contract

Hu Kai[1], Bai Xiaomin[1], Gao Lingchao[2], and Dong Aiqiang[2]
[1](*School of Computer Science and Engineering, Beihang University, Beijing* 100191)
[2](*Beijing China Power Puhua Information Technology Co. Ltd, Beijing* 100192)

Abstract Smart contract is a code contract and algorithm contract and will become the basis of future agreements in digital society. Smart Contract utilizes protocols and user interfaces to facilitate all steps of the contracting process. This paper summarized the main technical characteristics of smart contract and existing problems such as trustworthiness and security and proposed that formal method is applied to the smart contract modeling, model checking and model verification to support the large-scale generation of smart contract. In this paper, a formal verification framework and verification method for smart contract in the whole life circle of smart contract has been proposed. The paper presented a smart shopping scene, in which Promela language is used for modeling a SSC(smart shopping contract) and SPIN is used to simulate and model checking to verify the effect of formal method on smart contract.

Key words smart contract; formal method; modeling; verification; SPIN

摘 要 智能合约是一种代码合约和算法合同,将成为未来数字社会的基础技术,它利用协议和用户接口,完成合约过程的所有步骤.总结了智能合约主要技术特点和现存的可信、安全等问题,提出将形式化方法应用于智能合约的建模、模型检测和模型验证过程,以支持规模化智能合约的生成.研究提出了一个应用于智能合约生命周期的形式化验证框架和验证方法,针对一个智能购物场景,采用 Promela 建模语言对智能购物合约进行建模,用 SPIN 进行了模型检测,验证了形式化方法对智能合约的作用.

关键词 智能合约;形式化方法;建模;验证;SPIN 模型检测工具

中图法分类号 TP301

收稿日期:2016-10-31
基金项目:国家自然科学基金项目(91538202)
通信作者:白晓敏(baixiaomin@buaa.edu.cn)

2016年6月17日，运行在以太坊公有链上的The DAO[1-2]智能合约遭遇攻击，该合约筹集的公众款项不断被一个函数的递归调用转向它的子合约，涉及总额300多万以太币，这是一起严重的智能合约被攻击事件. The DAO本质上是个VC(风险投资基金)，通过以太坊筹集到的资金会锁定在智能合约中，没有哪个人能够单独动用这笔钱. 事件是The DAO智能合约本身的脚本漏洞被利用所引发. 因此，智能合约的安全、可信问题引起大家的关注，如何编写具有高可靠性、高安全性的智能合约成为目前亟待解决的问题.

“智能合约”(smart contract)[3]这个术语至少可以追溯到1995年，是由密码学家尼克萨博首次提出的. 智能合约是能够自动执行合约条款的计算机程序，他创造性地提出“智能合约就是执行合约条款的可计算交易协议”.

尼克萨博指出计算机代码可以代替机械设备，进行更复杂的数字财产交易，未来的某一天，这些程序甚至可能取代处理某些特定金融交易的律师和银行，“智能财产可以将智能合约内置到物理实体的方式，被创造出来”. 比如，房屋出租商将发现智能合约这种用途很有吸引力，一所房屋的门锁，能由被连接到物联网上的智能合约打开，所有门锁都是连接互联网的. 当你为租房进行了一笔交易时，存储在智能手机中的钥匙可以为你打开房屋，并进行自动资金转移. 虽然智能合约仍然处于初始阶段，但是其潜力显而易见，因为它把人与法律协议以及网络之间复杂的关系程序化了.

智能合约有许多非形式化的定义，尼克萨博给出了一个简短的概念，即“智能合约通过使用协议和用户接口来促进合约的执行”；Miller[4]认为智能合约就是用程序代码编写的合约，它的条款由程序来执行；Ethereum的智能合约就是基于区块链的可直接控制数字资产的程序[5].

总的来说，一个智能合约是一套以数字形式定义的承诺，包括合约参与方可以在上面执行这些承诺的协议. 它是能够自动执行合约条款的计算机程序，被部署在分享的、复制的账本上，它可以维持自己的状态，控制自己的资产和对接收到的外界信息或者资产进行回应. 承诺指的是合约参与方同意的权利与义务，这些承诺定义了合约的本质和目的. 数字形式意味着合约不得不写入计算机可读的代码中. 只要参与方达成协议，智能合约建立的权利和义务，是由一台计算机或者计算机网络执行完成的. 协议是技术实现，在此基础上，合约承诺被实现，或者合约承诺实现被记录下来.

尼克萨博提出的智能合约理论几乎与互联网(world wide web)同时出现，但应用实践却一直严重地落后于理论. 主要面临2个方面问题：一是智能合约如何来控制实物资产保证有效地执行合约，计算机程序很难控制现实世界的现金、股份等资产；二是计算机很难保证执行这些条款以获得合约方的信任，合约方需要可靠地解释和执行代码的计算机，它无法亲自检查有问题的计算机，也无法直接观察与验证其他合约方的执行动作，只有让第三方审核各方合约执行的记录. 而区块链技术的出现解决了这些问题，从而触发了智能合约的应用. 如今很多区块链系统，如Ethereum[6]，有可编程的合约语言与可执行的基础设施，以实现智能合约. 在Ethereum中，智能合约是存储在区块链上的脚本，通过区块链节点以分布式的形式执行，相当于商业交易、监督管理过程中法律、法规的执行者. 智能合约可以按序、安全、可验证的方式实施特定的流程.

然而，智能合约的生成和执行还存在一些问题，包括：1)智能合约对保证资产的安全性提出了更高的要求，合约需要验证合约逻辑属性的正确性，重要的是能够自动生成可信的可执行合约代码以节省成本并提高效率；2)智能合约最终会取代合约文本，因此我们必须保证合约文本与合约代码的一致性. 基于上述问题，本文提出将形式化方法应用于智能合约的整个生命周期.

形式化方法[7]是描述系统性质的基于数学的技术，用于计算机软件的规范、开发和验证. 将形式化方法用于软件设计，是期望能够像其他工程学科一样，使用适当的数学分析以提高设计的可靠性和鲁棒性. 其中，形式化方法中很重要的一步就是形式化验证，形式化验证可以以更正式的方式产生程序. 例如，可以进行从规范到程序的属性或细化的证明.

近年来，模型驱动(model-driven)[8]的设计与开发方法逐渐受到重视，并被工业界认为是切实可行的重要方法. 该方法将模型作为整个系统开发过程的核心元素，在设计阶段就建立系统的体

系结构模型，尽早进行验证和分析. 同时，模型的重用以及基于模型转换的自动或半自动的逐步求精过程，都有助于降低系统开发时间和成本. 然而，模型驱动的设计与开发方法的真正有效使用，需要多方面的支持. 首先，需要合适的体系结构建模语言，并要求建模语言对系统的软/硬件结构、运行时环境、功能行为以及非功能属性可表达；其次，为满足系统的需求，形式验证与分析方法是重要的手段；最后，基于经过验证和分析的模型，研究自动代码生成技术，有助于避免手工编码带来的错误，可以进一步提高系统的质量属性.

将形式化方法应用于智能合约整个生命周期的流程包括合约设计、自然语言描述、形式化描述、模型验证、自动代码生成和一致性测试. 本文重点探讨形式化描述和形式化验证可以更好地生成和执行智能合约. 其中，形式化描述可以克服自然语言描述的缺点，例如二义性. 形式化验证可以检查合约中是否存在逻辑错误，可以做可达性分析、不变性分析、等价性分析等.

1 智能合约的性质

目前，智能合约处在逐渐的发展中. 2015 年 3 月 20 日，以太坊基金会发布了 Ethereum 项目，它是一个开源数字货币和区块链平台，为开发者提供在区块链上搭建和发布智能合约的平台，可用来担保和交易任何事物. 2015 年，Linux 基金会发起一个推进区块链数字技术和交易验证的开源项目 hyperledger[9]，这是一个区块链和智能合约结合的开源平台，允许任何人发行个人货币，通过权限控制保证了用户的隐私和交易的安全性. 目前有很多机构和学者都是基于以上 2 个原型系统对智能合约作更深入的研究.

研究智能合约通过协议与用户界面来促进合约过程的所有步骤，我们提出了一个良好的智能合约，它具备以下 6 个基本特征：一致性、可定制性、可观察性、可验证性、自强制性和接入控制.

1）一致性. 智能合约应与现有法律一致，必须经过具有专业法律知识的专业人士制定审核，不与现有法律冲突，具有法律效应.

2）可定制性. 智能合约是可定制的. 多个合约可以合并成一个复合或复杂的合约.

3）可观察性. 合约方能够通过用户界面去观察关于合约的所有状态，包括合约本身及合约执行过程的记录等.

4）可验证性. 合约方执行合约的过程是可验证的.

5）自强制性. 对于违反合约行为的制裁必须是强制性的，这需要把资产变得数字化可控，并且由密码协议保证其安全.

6）接入控制. 就是指具有相关合约利益的人才能接触相应的合约信息，即与合约相关的知识、控制、执行都应该作为资产保护起来，只有发生争执时，才把内容提供给第三方检验.

智能合约作为计算法律学的一种新技术，有一个很重要的特性是当条件满足时可以自动执行相应动作. 但这一特性在其他应用领域已经有类似的技术. 例如，20 世纪 80 年代的基于知识系统都有这一特性. 一个是基于规则的系统，当满足某个条件时，相应的规则就会被触发. 如果有多个规则同时被触发，会有相应的解决机制协调这些规则的执行. 一个是黑板架构系统，在这个系统中，有多个代理同时监控，当某个条件满足时，相应的代理会激活自身的规则并执行.

在 Open-Transactions[10] 中这样描述智能合约：一个智能合约涉及多个当事人，一旦被激活就能自主运行，它有可执行的代码；只有被选定的功能可以被激活执行；可操纵那些被明确声明的合法资产；可暂时存储资金；合约的状态值被不断地更新、记录.

下面给出一个互联网智能购物合约的例子. 基本合约包括商品订货、分销和售后服务. 组合的合约用于连接客户和商家.

基础合约：

合约 1. 货物订购与分配.

合约方：客户和商家.

```
Contract goods_ordering_and_distribution
(goods_information, payment, distribution)
BEGIN
IF goods are available and pay is completed
    and distribution is available
  Inform merchant to send the goods to
    customer, set the terminator=
    timestamp+one week
```

```
ELSE
  The transaction failed and returned the
      money to customer
Wait one week for the acknowledgment
    message from customer
IF acknowledgment message received
  Pay to merchant and quit
IF timeout
  The transaction failed and returned the
    money to customer
END
```

合约 2. 售后服务合约.

合约方:客户和商家.

```
Contract sale_after_service (item_information,
payment, terminator)
BEGIN
WHILE (timestamp<terminator) {
  Wait for the feedback message from
    customer
  IF merchant received message from
    customer
    SWITCH(message){
      Case goods_return message:
        IF timestamp<terminator
          merchant wait for the goods
        IF merchant received the goods
          merchant return the money
            back to customer
        ELSE
          merchant reject and quit
      Case goods_exchange message:
        IF timestamp<terminator
          merchant wait for the goods
        IF merchant received the goods
          merchant send the new goods to
            customer
      Other:
        NULL
    }
  }
  QUIT
END
```

然而,当前的智能合约研究还处于初级阶段,应用还十分简单,甚至是不智能的,智能合约面临许多可信与安全问题. 作为一种特殊的程序代码,除面临一般软件面临的可信和安全问题外,智能合约还面临以下可信和验证方面问题:

1) 如何编制合约双方认可的模板框架,谁来编程实现合约代码,并保证代码的正确性及获得双方认可.

2) 合约验证问题. 程序都是有 bug 的,如明显地有利于合约的一方,该怎样进行修复,如何验证合约的逻辑正确并杜绝漏洞.

3) 合约的定制问题. 如何制定好的智能合约模板,根据不同场景定制不同合约,组合多个合约形成复合合约.

4) 一致性问题. 智能合约代码执行与文本合约具有一致性吗?不一致的合约是不可信的.

5) 智能合约执行过程中的可控性和可调度性,要确保执行过程中的可信性和安全性.

下面本文提出采用形式化方法来研究解决这些问题.

2 形式化方法的引入

为解决智能合约上述所提到的问题,本文引入形式化方法,将形式化方法应用于智能合约生成和执行的整个生命周期.

形式化方法[11]是指用数学方法描述和推理基于计算机的系统,直观地说,就是规范语言+形式推理,在技术上通过精确的数学手段和强大的分析工具得到支持,其表现形式通常有逻辑、离散数学、状态机等. 规范语言包括语法、语义以及满足关系等,可以分为 4 类:抽象模型规范法、代数规范法、状态迁移规范法和公理规范法.

形式化方法主要包括形式规约和形式化验证. 形式规约使用具有精确语法和语义的形式语言刻画系统的行为和性质,是设计系统的约束和验证系统是否正确的依据;形式化验证则是在形式规约的基础上,建立系统行为及其性质的关系,从而验证系统是否满足期望的关键性质,主要包括模型检测和定理证明.

模型检测[11]是一种重要的自动验证和分析技术,通过显式状态搜索或隐式不动点计算来验证

有穷状态系统是否满足预期的性质. 模型检测方法目前已经涵盖了并发系统模型检测、实时系统模型检测、混成系统模型检测以及概率模型检测等多个方面.

模型检测[12]详细探讨了模型中的相关属性. 这不仅对于有限模型是可以实现的,而且对于一些无限模型也是可以实现的,其中无限状态集合可以通过使用抽象或利用对称性的优点来表示. 通常包括:探索模型中的所有的状态和转换,通过使用智能的和领域特定的抽象技术在单个操作中考虑整个状态组,并减少计算时间.

基于有限状态机的模型检验可以全自动验证,它可以检测模型的不同状态. 定理证明需要在一些关键路径进行手动控制,它可以证明程序的正确性. 等价验证主要是验证设计的一致性,也就是说,设计是否满足需要,或者程序和合约是否相同.

定理证明技术将系统行为和性质都用逻辑方法来刻画,基于公理和推理规则组成的形式系统,证明系统是否满足期望的关键性质. 而使用定理证明器来辅助证明,相当于将手工证明变成一系列能够在计算机上自动进行的符号演算,能够对证明过程进行正确性检查,从而提高证明的可信度. 常用的定理证明器主要有 Coq,PVS,Isabelle 等.

作为形式化方法的工程实践,模型驱动工程(MDE)旨在提高程序规范中的抽象级别,并增加程序开发的自动化. MDE[8]的思想是使用不同抽象级别的模型用于开发系统,从而提高程序规范中的抽象级别. 通过使用可执行模型转换来增加程序开发中的自动化. 较高级别的模型被转换为较低级别的模型,直到该模型可以使用代码生成或模型解释来执行. MDE 可以支持智能合约的整个生命周期,从建模、验证、代码生成到一致性测试,应用于智能合约的形式化方法流程如图 1 所示.

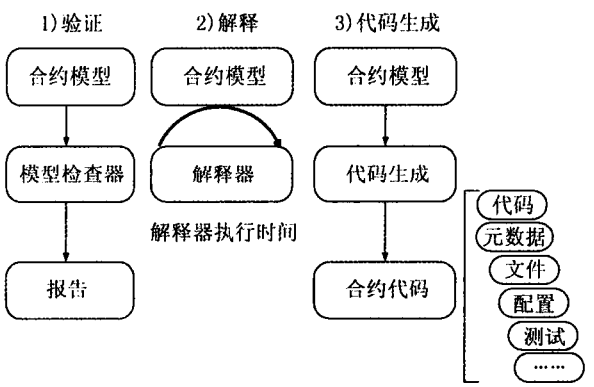

图 1 智能合约的 MDE 流程图

在模型驱动框架下,由于体系结构模型包含的系统特征和信息较多,一般不直接对体系结构模型进行验证和分析,而主要采用模型转换的方式,即将体系结构模型(或子集)转换到另一个形式模型,或者直接转换到模型检测工具或定理证明器,目的是为了重用这些已有的验证和分析能力.

基于模型的代码自动生成[13]概念源于模型驱动架构(model driven architecture, MDA). 模型到代码的生成和模型到模型的转换都是 MDA 模型转换的子集. MDA 能够尽早对系统进行分析与验证,并将验证后符合需求的模型生成代码,不仅有助于保证系统的质量属性,同时能够促进应用开发的标准化和工业化,并有效控制开发时间与成本.

一致性测试[14]指的是被测系统与标准的一致性,它是一种黑盒测试. 通过一致性测试,可以给用户提供 2 个信息:通过了一致性测试的合约实现,具有合约所要求的各种能力;在具有代表性的合约实例中,被测试的合约代码实现的外部特性与标准合约文本的要求一致.

综上,我们提出智能合约的形式化方法框架如图 2 所示. 一个合约的生成包括:首先,用户提出需求,根据用户需求制定合约文本;然后对合约

图 2 形式化方法应用框架

文本进行形式化描述，并选择合适的建模语言和建模工具对形式化规格说明文档进行建模和性质验证，其中，模型验证包括理论证明和模型检测，在模型检测中，我们通常使用模型转换来验证更多的性质；最后是一致性测试. 为了证明合约代码与合约文本在性质和执行力是保持一致的，因此，需要对合约文本和合约代码进行一致性测试. 这就是将形式化方法应用于智能合约完整生命周期的框架.

图 3 是将形式化方法应用于智能合约整个生命周期的流程. 当需要新的合约时，使用非正式化规范来设计合约，然后使用形式化规范来描述合约以验证合约. 模型检验工具可以用来检查合约，或使用演绎验证方法来证明合约. 其次，可以通过模型工具，将模型自动化生成合约代码. 最后，一致性测试确保文本和程序代码的一致性.

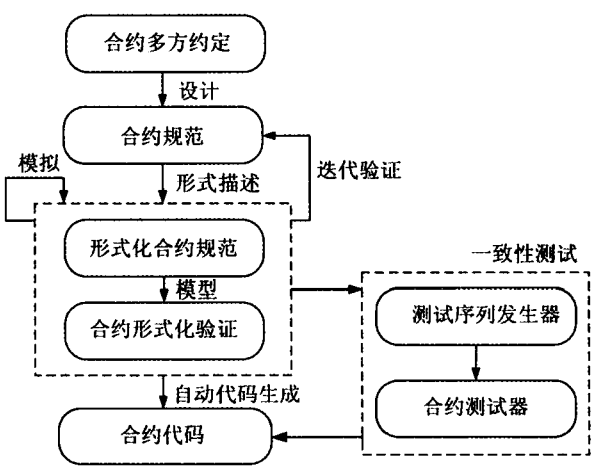

图 3　智能合约的形式化流程

将形式化方法应用于智能合约，使得合约的生成和执行有了规范性约束，保证了合约的可信性，使人们可以信任智能合约的生产过程和执行效力. 合约的形式化验证保证了合约的正确属性，自动化代码生成提高了合约的生成效率，合约的一致性测试保证了合约代码与合约文本的一致性.

3　智能合约的形式化验证方法

对智能合约的形式化验证包括形式化描述、形式化验证、自动代码生成和一致性测试. 本文中，我们重点探讨智能合约的形式化描述和形式化验证方法.

形式化方法的另一个重要研究内容是形式化验证. 形式化验证与形式化规约之间具有紧密的联系，形式化验证就是验证已有的程序（系统）P 是否满足其规约(ϕ,ψ)的要求（即 $P(\phi,\psi)$），它也是形式化方法所要解决的核心问题[15].

传统的验证方法包括模拟和测试，它们都是通过实验的方法对系统进行查错. 模拟和测试分别在系统抽象模型和实际系统上进行，一般的方法是在系统的某点给予输入，观察在另一点的输出，这些方法花费很大，而且由于实验所能涵盖的系统行为有限，很难找出所有潜在的错误. 基于此，早期的形式验证主要研究如何使用数学方法，严格证明一个程序的正确性（即程序验证）.

Von Neumann[16]（冯. 诺伊曼）早在 1948 年发表的论文“Planing and Coding Problems for an Electronic Computer Instrument”中就提到了程序正确性证明；Floyd[17] 在 1967 年发表论文“Assigning Meanings to Programs”中提出了验证流程图程序正确性的归纳断言方法，这是程序验证方面的开创性工作；1969 年，Hoare[18] 在“An axiomatic Basis for Computer Programming”一文对 Floyd 归纳断言法形式化，首次提出程序验证的公理系统，称为 Hoare 逻辑公理化方法，1970 年以来还出现能辅助用户正确编制程序的实用的半自动程序验证系统. 1976 年，S. Owicki，D. Gries 提出并发程序的验证方法；1977 年，A. Pnueli 提出反映系统验证的时序逻辑方法；1981 年，E. M. Clarke，E. A. Emerson 提出有穷状态并发系统的模型检测方法；80 年代后期主要研究解决模型检测“状态爆炸问题”；90 年代起主要研究实时与混成系统的形式验证问题.

目前常见的形式化验证方法主要可分为 2 类：演绎验证和模型检测. 其中，早期（20 世纪 60—70 年代）的形式化技术主要采用演绎法证明顺序和并发程序正确性，而近期（20 世纪 80—90 年代）则多采用模型检测方法验证实时和混成系统.

1）演绎验证. 演绎验证是早期采用的主要验证技术，它基于定理证明的基本思想，采用逻辑公式描述系统及其性质，通过一些公理或推理规则来证明系统具有某些性质.

演绎验证的优点是可以使用归纳的方法来处理无限状态的问题，并且证明的中间步骤使用户对系统和被证明性质有更多的了解. 缺点是现有的方法不能做到完全自动化，还需与用户交互，要求用户能提供验证中创造性最强部分的工作. 因而演绎证明方法的效率较低，很难用于大系统的验证.

目前主要演绎验证工具有：基于 Manna-Pnueli 证明系统的 STeP(stanford theorem prover)、TLV、机器定理证明器(ACL2, Coq, HOL, Isabelle, Larch, Nuprl, PVS, TPS)等[19].

2) 模型检测(算法验证). 模型检测是对有穷状态系统的一种形式化确认方法，它基于状态搜索的基本思想，是模拟和测试方法的自然延伸，搜索的可穷尽性有赖于为合约建立有穷状态的模型，这为建模造成一定的难度，但能保证搜索过程终止.

模型检测方法的基本思想是通过状态空间搜索来确认合约是否具有某些性质. 即给定一个合约(程序)P 和规约 φ，生成对应的合约模型 M，然后证明 $M \models \varphi$，即规约公式 φ 在合约模型 M 中成立，这样就证明了合约(程序)P 满足规约 φ.

模型检测方法通常采用 Dolev-Yao 模型、模态逻辑、有限状态机和进程代数等理论作为合约分析的理论基础，其基本思想是用状态迁移系统 S 表示系统的行为，用模态/时序逻辑公式 F 描述系统的性质. 一般地，一个模型检测方法主要由特定的形式模型、形式逻辑和相应的模型检测算法 3 个方面构成，不同的模型检测方法具有不同的应用领域.

将模型检测应用于智能合约以解决合约的可信问题，一般包括以下步骤：

1) 建模. 通过选择合适的建模语言和建模工具，使用模型检测工具能够接受的形式语言来描述合约.

2) 描述. 阐明所要验证的合约性质，包括合约的状态可达性、死锁、活锁、有界性等.

3) 验证. 对合约的状态空间进行搜索，发现合约存在的问题并及时修改，对合约进行迭代验证.

目前形式化描述技术主要分为 2 种类型：形式化描述模型和形式化描述语言. 通过形式化描述模型，可以获得抽象的合约模型. 形式化描述语言总是基于一种或多种形式化描述模型. 形式化描述技术已经有几十年的发展，目前有多种形式化描述模型和形式化描述语言，如图 4 所示. 形式化描述语言主要有 3 种标准：CCITT 国际电报电话咨询委员会(International Telephone and Telegraph Consultative Committee)组织制定的 SDL，ISO 组织制定的 LOTOS 和 ESTELLE.

形式化描述方法	分类	内容
形式化描述模型	状态变迁模型	FSM, EFSM, Petri 网模型
	进程代数	通信系统演算(CCS) 通信顺序进程(CSP)
	其他	时序逻辑(或时态逻辑)(TL)
形式化描述语言	CCITT 组织	SDL
	ISO 组织	LOTOS, ESTELLE
	其他	Promela 语言：SPIN(著名模型检测工具)的输入语言

图 4 各种形式化描述技术

目前的模型检测工具主要有：COSPAN/FORMAL CHECK(Bell), MURPHY(Stanford), SPIN(Bell), SMV(CMU), VIS(Berkeley)等，常用的有 Telelogic Tau, SPIN, UPPAL 等[19].

SPIN 是美国贝尔实验室的形式化方法与验证小组开发的模型检测工具. 它所关心的主要问题是进程之间的信息能否正确地交互，而不是进程内部的具体细节. 由于其良好的性能、完善的文档、开源及不断的维护更新服务使得 SPIN 被广泛应用于工业界和学术界. 它采用的描述语言为 Promela[20].

模型检测主要适用于有穷状态系统，早期主要用于硬件和协议的验证. 模型检测的优点是完全自动化并且验证速度快，即便是只给出了部分描述的合约，通过搜索也可以提供关于已知部分正确性的有用信息. 尤其重要的是，在性质未被满足时，搜索终止可以给出反例，这种信息常常反映出合约设计中的细微失误，因而对于合约排错有极大的帮助.

形式化验证方法可以检查智能合约的很多属性，例如，合约的公平性、可达性、有界性、活锁、死锁、不可达以及无状态二义性等.

4 智能合约的验证实例

在本节中，使用 Promela 建模语言和检测工具 SPIN 来建立和验证智能购物合约 SSC(smart shopping contract)的模型.

SPIN[20]是一个通用的工具，以严格的和大多数自动化的方式验证分布式软件模型的正确性. 它是从 1980 年开始，由 Gerard J. Holzmann 和贝尔实验室计算科学研究中心的原始 Unix 组的研究人员编写而成. 该软件自 1991 年以来一直可用，并继续遵循该领域的新发展而发展. SPIN 是一种著名的分析验证并发系统逻辑一致性的工具，以其简洁明了和自动化程度高而备受注目. SPIN 已成功应用在安全协议验证、控制系统验证、软件验证及最优化规划等领域.

作为一种形式化自动验证工具，SPIN 的目的是提供：1)系统建模语言 Promela(process meta language)，用于直观、明确地描述系统 Promela 模型规约，而不考虑具体实现细节；2)功能强大而简明的描述系统应满足性质(属性要求)的逻辑表示法(L T)L；3)提供一套验证系统建模逻辑一致性及系统是否满足所要验证性质的方法. 除模型检测之外，SPIN 还可以作为模拟器操作，遵循系统的一个可能的执行路径并且向用户呈现所产生的执行轨迹.

SSC 应在用户订单之后触发，并且它具有 2 个参与者，即用户和商店.

智能购物合约的描述如下：当用户下订单时需要将购物所需的资金提交给智能购物合约，智能购物合约暂时持有资金. 同时 SSC 启动 2 个子进程：用户进程和商店进程. 用户进程：如果商家在 7 天内没有交货，用户将取消交易，SSC 会将资金退还给用户，用户进程定时、周期地检测交货状态；商店进程：商家收到订单后，首先系统判断订单是否结束，如果订单没有超时，则商店发货. SSC 需要确保资金的安全性和交易过程中各个状态的可达性.

SSC 的 Promela 模型如图 5 所示.

使用模型检测工具 SPIN 检测 SSC 模型，模型的模拟结果如图 6 所示.

```
byte money=100
byte user_money=0
byte shop_money=0
byte day=0
bool isSend=false
ltl {<>isSend }
active proctype user() {
  do
  :: isSend -> atomic {
    shop_money=100;
    money=0;
    break; }
  :: (day>6)-> {
    user_money=100;
    money=0;
    break; }
  :: else -> day=day+1;
  od
  :: }
active proctype shop() {
  do
  ::(day<7)->isSend=true;
  ::break;
  od
  ::}
init {
  atomic{run user();run shop(); }
}
```

图 5 SSC 的 Promela 模型

从图 6(a)的模型仿真结果可以看出，一旦超时(day= 8)，SSC 将钱退还给用户.

如图 6(b)结果所示，当商店在第 2 天送货时，SSC 将钱转给商店. 实验结果与预期结果一致.

由于 SSC 是有限状态模型，通过对 SSC 的建模和验证，SPIN 可以随机生成合约的所有状态，实验结果与合约的预期结果一致.

5 结 论

本文针对智能合约存在的可信与安全问题，将形式化方法应用于智能合约的生命周期验证，从形式化描述和形式化验证方面进行了详细的阐述. 一个好的模型检测工具有助于检查和验证智能合约的各项属性. 通过一个验证实例可以看出，智能合约可以在合约的不同阶段获得不同的状态，当智能合约验证时，SPIN 可以随机产生若干种不同的结果. 形式化方法可以在智能合约的建立、验证和代码生成中得到重要应用，是智能合约可信和安全性的发展方向. 因此，后续将会对合约的自动化代码生成和一致性测试作更深入的研究.

```
  0:    proc  - (:root:) creates proc  0 (user)
  0:    proc  - (:root:) creates proc  1 (shop)
  0:    proc  - (:root:) creates proc  2 (:init:)
0 user  8   else
0 user  17  day = (day+1)
Starting user with pid 3
  4:    proc  2 (:init:) creates proc  3 (user)
Process Statement          day
2 :init ini run user()     1
Starting shop with pid 4
  5:    proc  2 (:init:) creates proc  4 (shop)
2 :init ini run shop()     1
  7:    proc  4 (shop) terminates
3 user  8   else           1
3 user  17  day = (day+1)  1
0 user  8   else           2
0 user  17  day = (day+1)  2
0 user  8   else           3
0 user  17  day = (day+1)  3
3 user  8   else           4
0 user  8   else           4
3 user  17  day = (day+1)  4
3 user  8   else           5
0 user  17  day = (day+1)  5
3 user  17  day = (day+1)  6
3 user  8   else           7
3 user  17  day = (day+1)  7
3 user  8   day>7          8
3 user  16  user_money = 1 8
Process Statement          day         user_money
3 user  15  money = 0      8           100
Process Statement          day         money       user_money
0 user  8   day>7          8           0           100
 33:    proc  3 (user) terminates
0 user  16  user_money = 1 8           0           100
0 user  15  money = 0      8           0           100
 36:    proc  2 (:init:) terminates
 36:    proc  1 (shop) terminates
 36:    proc  0 (user) terminates
5 processes created
```

(a) 超时退款

```
  0:    proc  - (:root:) creates proc  0 (user)
  0:    proc  - (:root:) creates proc  1 (shop)
  0:    proc  - (:root:) creates proc  2 (:init:)
Starting user with pid 3
  1:    proc  2 (:init:) creates proc  3 (user)
2 :init ini run user()
Starting shop with pid 4
  2:    proc  2 (:init:) creates proc  4 (shop)
2 :init ini run shop()
1 shop  22  day<6
1 shop  23  isSend = 1
Process Statement          isSend
1 shop  22  day<6          1
3 user  8   isSend         1
1 shop  23  isSend = 1     1
0 user  8   isSend         1
0 user  9   shop_money = 1 1
Process Statement          isSend      shop_money
0 user  11  money = 0      1           100
Process Statement          isSend      money       shop_money
0 user  8   break          1           0           100
3 user  9   shop_money = 1 1           0           100
3 user  11  money = 0      1           0           100
1 shop  22  day<6          1           0           100
3 user  8   break          1           0           100
 18:    proc  4 (shop) terminates
1 shop  23  isSend = 1     1           0           100
 19:    proc  3 (user) terminates
 19:    proc  2 (:init:) terminates
 21:    proc  1 (shop) terminates
 21:    proc  0 (user) terminates
5 processes created
```

(b) 交易完成

图 6 模型模拟结果

参 考 文 献

[1] Decentralized autonomous organization: The DAO [EB/OL]. (2016-06-17) [2016-10-10]. https://en.wikipedia.org/wiki/Decentralized_autonomous_organization

[2] Castillo M. DAO Attack [EB/OL]. 2016 [2016-10-10]. http://www.coindesk.com/the-dao-just-raised-50-million-but-what-is-it/

[3] Szabo N. Formalizing and securing relationships on public networks [J/OL]. First Monday, 1997, 2(9) [2016-10-10]. http://ojphi.org/ojs/index.php/fm/article/view/548

[4] Miller M S. The digital path: Smart contracts and the third world [EB/OL]. 2003[2016-10-10]. http://www.erights.org/talks/pisa/paper/index.html

[5] Ethereum: A next-generation smart contract and decentralized application platform [OL]. 2014 [2016-10-10]. https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-White-Paper

[6] Ethereum [EB/OL]. 2014 [2016-10-10]. http://www.ethereum.org/

[7] Formal methods [EB/OL]. 2002[2016-10-10]. http://en.wikipedia.orgi/Formal_methods

[8] Vidar S, Peter H, Kraemer F A. Model-driven engineering of reliable fault-tolerant systems—A state-of-the-art survey [J]. Advances in Computers, 2013, 91: 119-205

[9] Hyperledger: Blockchain technologies for business [EB/OL]. 2016 [2016-10-10]. https://www.hyperledger.org/

[10] OpenTransactions: Smart contracts from open yransactions [EB/OL]. [2016-10-10]. http://opentransactions.org/wiki/index.php?title=Smart_contracts

[11] Kenneth M. Symmetry and model checking [J]. Formal Methods in System Design, 1996, 9(1/2): 105-131

[12] Baier, Christel, Katoen, et al. Principles of Model Checking [M]. Cambridge: MIT Press, 2008

[13] 朱江. 基于AADL架构模型的代码自动生成技术研究与实现[D]. 北京: 北京航空航天大学, 2011

[14] 张颖蓓. LDP协议一致性测试研究与实现[D]. 长沙: 国防科技大学, 2003

[15] Schamann J M. Automated Theorem Proving in Software Engineering [M]. Berlin: Springer, 2001: 546-555

[16] Von Neumann J, Goldstine H H. Planning and coding of problems for an electronic computing instrument [OL]. Institute for Advanced Study, Princeton, New Jersey, 1948 [2016-10-10]. http://publications.ias.edu/sites/default/files/u:13_p:214____Planning_Coding_Problems_v2p3r.pdf

[17] Floyd R W. Assigning meanings to programs [M] // Program Verification. Berlin: Springer, 1967: 19-32

[18] Hoare C A R. An axiomatic basis for computer programming [M] //Pioneers and Their Contributions to Software Engineering. Berlin: Springer, 1969: 576-580

[19] Clarke E M, Grumber J O, Peled D A. Model Checking [M]. Cambridge: MIT Press, 1999

[20] Prigent A, Cassez F, Dhaussy P, et al. Extending the translation from SDL to promela [C] //Proc of the 9th Int SPIN Workshop on Model Checking of Software. Berlin: Springer, 2002: 79-94

胡 凯

博士，教授，主要研究方向为分布式计算、区块链与数字社会技术.

hukai@buaa.edu.cn

白晓敏

硕士研究生，主要研究方向为形式化方法、智能合约和区块链技术.

baixiaomin@buaa.edu.cn

高灵超

高级工程师，主要研究方向为智能电网、大数据、企业云计算和区块链技术.

gaolingchao@sgitg.sgcc.com.cn

董爱强

硕士研究生，高级工程师，主要研究方向为国网信息化、大数据多维度分析、云环境下软建构建平台和区块链技术.

dongaiqiang@sgitg.sgcc.com.cn