基于纳什均衡的智能合约缺陷检测

陈晋川[1),2)] 夏华辉[1)] 王璞巍[1),2)] 马纳波[1)] 王 琪[1)] 李浩然[1)] 杜小勇[1),2)]

[1)](中国人民大学信息学院 北京 100872)

[2)](数据工程与知识工程教育部重点实验室(中国人民大学) 北京 100872)

摘 要 本文研究区块链智能合约的缺陷检测问题,即检测合约中是否存在部分合约方无论选择什么动作,均无法避免损失的状态.将智能合约问题转换成合约状态迁移图上的博弈策略选择问题,提出了基于纳什均衡理论的合约缺陷自动检测方法,以及为了提高检测效率,提出了针对合约状态变迁图和博弈策略形式的一系列化简方法.最后,实现了一个智能合约建模、分析和部署工具.针对一组真实合同的实验表明,该工具能自动发现部分合同缺陷,化简方法提升缺陷检测效率的效果明显.

关键词 智能合约;纳什均衡;缺陷检测;区块链;博弈论

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2021.00147

Detecting Smart Contract Loopholes Based on Nash Equilibrium

CHEN Jin-Chuan[1),2)] XIA Hua-Hui[1)] WANG Pu-Wei[1),2)] MA Na-Bo[1)] WANG Qi[1)]
LI Hao-Ran[1)] DU Xiao-Yong[1),2)]

[1)](*School of Information, Renmin University of China, Beijing* 100872)

[2)](*Key Lab of Data Engineering and Knowledge Engineering, Ministry of Education, Renmin University of China, Beijing* 100872)

Abstract In recent years, blockchain has attracted many interests form both academic and research communities, and been applied in cryptocurrency, financial techniques, logistics, copyright protection etc. Blockchain is basically a distributed storage system, consisting of a set of servers or nodes. In blockchain systems, smart contracts are widely adopted as the code laws to be obeyed by all participants. Before storing the data item into blockchain, each node needs to verify the validity of this operation by executing its corresponding smart contracts, i.e. computer programs. Noted that, in this paper we focus on classic smart contracts, i.e. computer codes for verifying, executing, or disseminating contracts in an informational manner. Smart contract has become one of the core techniques of blockchain, and it is very important to study how to detect loopholes existing in smart contracts. Here, the term loophole does not mean bugs in the contract codes, but some bad outputs of executing the contracts. These outputs are unfavorable to some contract partners. If all partners are rational and have complete information about the contracts, these outputs cannot be avoided no matter what choices the partners choose during executing the contracts. Therefore these contracts will not become repeated games since some partners cannot benefit from joining the contracts. These kind of contracts will be one-shot games and are harmful to business zoology. We propose a method to detect the loopholes of formalized smart

收稿日期:2019-12-02;在线发布日期:2020-05-23.本课题得到国家自然科学基金(U1911203)、贵州财经大学与商务部国际贸易经济合作研究院联合基金(2017SWBZD08)资助.**陈晋川**,博士,副教授,主要研究方向为分布式数据管理和区块链.E-mail:jcchen@ruc.edu.cn.**夏华辉**,硕士研究生,主要研究方向为区块链和智能合约.**王璞巍**(通信作者),博士,副教授,主要研究方向为服务计算和区块链.E-mail:wangpuwei@ruc.edu.cn.**马纳波**,硕士研究生,主要研究方向为分布式计算和区块链.**王 琪**,硕士研究生,主要研究方向为区块链.**李浩然**,硕士研究生,主要研究方向为服务计算和区块链.**杜小勇**,博士,教授,主要研究领域为数据管理技术、语义网技术、智能信息检索技术.

contracts based on the Nash equilibrium theory. Instead of hiring lawyers to check the contract terms one by one, we can now automatically find out the loopholes of contracts. We propose a data structure, called state-transition-graph, to describe the outputs for each partner of choosing different strategies of executing a contract. Based on the state-transition-graph, we further define the problem of smart contract game, and define the loopholes as a special kind of Nash equilibriums, where some partners would obtain zero or negative utilities and they cannot improve their utilities even if they change their own strategies. In this way, we are able to automatically find out the loopholes by computing the Nash equilibriums of smart contract game. However, the strategy forms obtained from the state-transition-graph may be too large due to state explosion. There could be millions of and even larger number of states in a strategy form, which causes very expensive time costs of computing Nash equilibriums. We then propose two rules, i. e. Strategy equivalence and Nash equivalence rules, for simplifying the state-transition-graph and strategy forms, which can greatly cut down the time costs. In order to further cut down the time costs of computing Nash equilibrium, we also propose a method to find out the redundant strategy combinations and prove that we can obtain the same Nash equilibriums when these strategy combinations are removed. Based on these methods, we design and implement a tool for modelling, analyzing and deploying smart contracts. We analyze a set of real contracts crawled from two professional lawyer web sites. These contracts cover the main contract categories listed in Contract Law of People's Republic of China. We discuss some contract loopholes detected by this tool, and evaluate the ability of our proposed methods to improve the performance. As illustrated by the experimental results, our methods can greatly reduce the strategy forms and the time costs.

Keywords smart contract; Nash equilibrium; loophole detection; blockchain; game theory

《计算机学报》

1 引 言

智能合约(Smart Contract)的概念最早可以追溯到 1994 年,Szabo 将智能合约定义为"一种可以实现合同条款的计算机程序[1]".但是,一直到 2008 年区块链(Blockchain)技术出现之前,智能合约都没能得到广泛关注.原因可能是很难找到一种合适的技术可以在非完全信任环境中保证计算机程序正确执行和不会被篡改.区块链源自于"比特币①"的底层技术,是一种去中心化、不可篡改、多方共同维护的分布式计算和存储系统.区块链网络中每个节点都是独立维护的,通过一种分布式共识机制,在没有权威中心节点的情况下,确保各个节点数据保持完全一致.研究者认为区块链系统非常适合为智能合约提供运行平台.2013 年后出现的以太坊②和 Hyperledger Fabric③都提供了图灵完备的编程语言用于编写智能合约,可以支持各种复杂的业务逻辑.至此,智能合约开始得到了越来越多的应用,其技术也有了迅速发展.

随着研究者对智能合约理解角度的不同,智能合约的内涵逐渐泛化,一些研究者开始把智能合约广义地理解成"运行在区块链系统之上的一段计算机代码,由共识机制确保代码的正确执行[2]".在本文中,我们关注的是 Szabo 所定义的经典智能合约:"以信息化方式验证、执行或传播合同的计算机代码".随着区块链技术的发展,智能合约变得越来越重要,现有的区块链系统都使用智能合约描述参与各方需要共同遵守的规则(合同条款).例如,在医疗数据共享领域,通过智能合约来规范数据的产生者(病人)、数据的保管者(医院)以及数据的使用者(数据分析公司)三方的权责利;Kantara Initiative④则试图将区块链和智能合约变成一种法律条款的数字撰写平台;2016 年工信部发布的《中国区块链技术和运用

① Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf
② Ethereum. https://www.ethereum.org
③ Hyperledger Fabric. https://hyperledger-fabric.readthedocs.io
④ Kantara Initiative. https://kantarainitiative.org

发展白皮书》[3]，将智能合约解释为一种用计算机语言取代法律语言去记录条款的合约.我们期待智能合约可以逐渐替代部分传统合同.但是，智能合约并没有统一的编程语言和运行模型.这就使得程序员之外的管理人员难以理解和编写智能合约.例如，以太坊使用 Sodility 编写智能合约，将其实现为一种状态机①；Hyperledger Fabric 智能合约被称为链码(Chaincode)，使用 Go 语言编写，没有全局状态，对区块链的读写都需要经过链码来执行.

接下来，我们用一个非常简单的买卖合同例子分析智能合约编写面临的问题："卖方发货，买方将钱转账给卖方".

编写这样的智能合约，我们首先遇到的问题是：如何构建一种与编程语言和平台无关的、易于理解的中间模型？文本合同是由参与各方的管理人员(律师)用自然语言撰写的，然而智能合约需要使用计算机编程语言实现，因此又要由专业的程序员编写.由于自然语言和程序语言之间的差别，双方可能无法准确理解彼此的意图.例如，上述例子中没有明确说明"卖方什么时候发货"，这给智能合约代码的编写带来困难，他们之间缺少一种双方都能准确理解的中间模型(类似数据库领域的 ER 模型)，这就是智能合约建模问题；

此外，我们还会面临第二个问题：如何发现智能合约中存在的逻辑缺陷？智能合约是由计算机代码构成的，现有的一些方法提出使用检测工具对智能合约代码进行检测以找出其中存在的漏洞[2].然而，智能合约的一些重要缺陷，其实不是代码漏洞，而是逻辑缺陷.例如，前面例子中的买卖合同存在着一个对卖方不利的执行路径："卖方发货，买方不转账".我们需要对智能合约进行分析以判断是否存在这样的逻辑缺陷，这就是智能合约的缺陷检测问题.

对于第一个问题，我们之前提出了一种智能合约形式化模型[4].首先，站在管理人员的角度，采用多 Agent 领域的承诺(commitment)模型[5]，将智能合约看作是一组"承诺"的集合.承诺描述了参与各方要执行的动作；其次，站在程序员的角度，将智能合约看作是一种基于承诺生成的状态机，描述智能合约的执行过程.在之前工作的基础上，本文提出从承诺到状态机的自动生成算法.但在这个生成过程中，往往会遇到"状态爆炸"问题，也就是会产生指数级的数量巨大的状态，导致生成算法无法高效完成.为了应对这个问题，本文又提出了一系列的化简算法，可以极大地提高状态机的生成效率.

对于第二个问题，根据智能合约参与各方选择的动作不同，智能合约将进入到不同的状态，形成不同的执行路径.我们提出把这种执行过程看作是一个多方博弈问题[6]：各方将会从智能合约中获取或损失收益，他们是理性的，会以收益最大化为目的来选择如何执行动作.智能合约中的逻辑缺陷实际上是指部分参与人会陷入到一种局面，无论他们选择什么动作，都无法避免收益损失.我们提出用"纳什均衡(Nash Equilibrium)"思想来捕捉这样的缺陷.纳什均衡是博弈论中的重要理论，对于参与博弈各方所能选择的策略，存在这样的策略组合：各参与方均不能通过单方面改变自己的策略来获取更好的收益.例如，考虑一个简单买卖合同，卖方有两种可以选择的策略{发货，不发货}，买方也有两种策略{转账，不转账}.对于策略组合{卖方不发货，买方不转账}，卖方和买方单方面更改策略都会面临收益损失：如果卖方将策略改为发货，买方还是不转账，这会使得卖方出现收益损失；同样如果买方将策略改为转账，卖方还是不发货，也会使得买方收益损失.这样的策略组合就称为纳什均衡点，它体现了博弈的均衡状态，在此状态下各个参与人都倾向于采取理性的行动.

我们定义合同的逻辑缺陷就是特殊的纳什均衡.如果合同的纳什均衡点中，各参与人的收益全部为零，或者收益有正有负，那么这个合同就可能有逻辑缺陷，因为部分参与人无法取得正收益，他们将不再倾向于参与合同.上述简单买卖合同例子中，唯一的纳什均衡点就是{卖方不发货，买方不转账}，此时双方收益均为零.原因在于该合同中没有对买方违约行为(不转账)进行约束.那么理性的卖方在权衡得失之后，将选择不执行合同.这样的合同对双方都没有价值.

对于简单的合同，执行路径较少，容易以人工的方式发现逻辑缺陷.但是对于复杂的合同，其执行路径将非常复杂.即使受过专业训练的律师也可能难以发现合同中所有的逻辑缺陷.当合同以智能合约的形式存在，人工检测逻辑缺陷将更不可行.

本文的贡献如下：

(1)提出一种基于纳什均衡的智能合约缺陷分

① Ethereum. https://solidity.readthedocs.io/en/v0.4.21/common-patterns.html#state-machine

《计算机学报》

析方法.形式化定义了智能合约的博弈问题、策略形式和纳什均衡;将智能合约的缺陷检测问题转换成了纳什均衡计算问题,通过对智能合约逻辑缺陷的分析,可以将逻辑缺陷以及相应的收益情况反馈给用户,避免因为智能合约的逻辑缺陷给用户带来损失;

(2)提出了一种基于智能合约形式化模型的纳什均衡检测算法.将形式化定义的智能合约转换为状态变迁图;基于策略等价原则和纳什均衡等价原则,提出了一系列状态变迁图的化简方法,极大地避免了状态爆炸;然后根据简化的状态变迁图,提出了冗余策略的化简方法,进一步提高计算纳什均衡的效率;

(3)依据《中华人民共和国合同法》(1999年3月15日颁布),常见的合同共有15类.我们参照华律网(66law.cn)和合同帮(hetongb.com)的合同范本,对其中14类合同进行了分析,找出了其中真实存在的合同缺陷,说明方法的有效性;并通过实验分析了所提出的化简方法的效率.

本文第2节介绍智能合约建模和分析的相关工作;第3节介绍如何为智能合约建模,以及智能合约的状态机生成算法;第4节介绍智能合约的博弈问题;第5节介绍状态变迁图的化简方法,以及冗余策略组合的化简方法;第6节通过实验说明所提出方法的有效性和效率.最后,第7节进行总结和对未来工作的展望,本文工作并非提出新的智能合约,而是将合同建模为状态机,并基于此建模方法对合同进行分析.

2 相关工作

智能合约漏洞暴露出来的安全性事件一直层出不穷,例如,2017年发生的DAO① 事件,黑客利用智能合约的漏洞将面向公众筹集的大量以太坊代币转向自己的地址,造成了巨大的经济损失.因此,智能合约安全性一直受到研究者广泛关注,目前很多工作采用软件工程领域的代码检测方法来做智能合约代码漏洞检测.2016年,Hiari② 提出将智能合约代码转换为AVL树,然后利用Isabelle定理证明器对AVL树进行验证,找出智能合约的代码漏洞.Yang等人[7]提出一种符号模型(Symbolic Process Virtual Machine,FSPVM)用于刻画智能合约的执行流程,然后基于Hoare逻辑验证智能合约代码的可靠性和安全性.Bigi等人[8]认为智能合约中参与人之间存在非合作博弈,在假设他们是理性的前提下分析他们可能采取的策略,在此基础上生成一种概率模型,再检测智能合约的安全性.上述这些工作目的是为了检测智能合约代码的安全漏洞.Bigi等人虽然提到了博弈论的概念,但也只是为了分析合约参与人可能的行为,并没有考虑合约公平性的问题.

不同区块链系统有不同的智能合约运行模型.例如,以太坊是公有链,任意节点都可以加入参与交易.它的智能合约用Sodility编写,以字节码形式存储于区块中,可以被各个节点执行.智能合约在发布之后可以被多次调用,每次调用均需从之前的区块中读取合约的状态和字节码,而执行的结果也会存入新的区块中.因此,智能合约在以太坊上的执行可以看作是一个状态机,每次执行将会从一个状态转移到另一个状态[9].Hyperledger Fabric是使用最广泛的许可链,节点需要经过授权才能参与交易.与以太坊不同的是,它的智能合约(链码)并不存储于区块,而是一直运行在联盟节点的docker容器之中.采用Go语言编写,对区块链数据的读写都要通过链码执行(但链码本身不记录执行后的状态).需要注意的是,智能合约是一个概念,指一种可以实现合同条款的计算机程序,本质上智能合约与区块链是独立的.链码是Hyperledger Fabric系统中实现智能合约的方式,在发展过程中超越了传统智能合约的概念.本文讨论的智能合约是一个状态机,是程序化的合同条款,我们将基于承诺模型来描述合同并自动生成相应的状态机.

在多Agent领域中,承诺(commitment)模型通常用来研究多个主体之间的交互协议.Singh等人[5,10]形式化定义了承诺模型,认为承诺是商业关系的基本抽象,是商务合同的本质.承诺是有状态的,例如,完成、违约和过期等.对于每个承诺,有操作(动作)可以触发其状态发生变化.不同承诺之间存在相互影响关系,多个承诺就构成了多个主体之间的商业协议[11].在承诺模型的基础上,Singh等人[12-13]又提出了商业合同公平性和健壮性等属性的检测方法,基本思路也是判断各方是否都从合同中收益或者合同执行是否会进入非法状态.这些工作没有考虑商业合同运行在区块链之上的情况,本文

① The DAO. https://github.com/slockit/DAO
② Hirai Y. Formal Verification of Deed Contract in Ethereum Name Service, 2019. https://yoichihirai.com/deed.pdf

《计算机学报》

提出在承诺模型的基础上构建智能合约(商业合同)，再设计算法从一组承诺自动生成运行模型(也就是状态机模型)，然后在状态机模型的基础上自动分析智能合约，检测出可能存在的逻辑缺陷. 在文献[11]中，也提出采用纳什均衡来分析合同的正确性(correctness)，但其方法需要穷举博弈树中所有可能的路径，没有做任何化简. 在效率上与我们所实现的 baseline 方法(第 6 节中的 SE 方法)相同.

综上所述，现有的智能合约分析工作主要还是进行代码漏洞分析，缺少对合约中逻辑缺陷的分析. 此外，现有区块链系统的智能合约运行模型各有不同，缺少一种统一的、无关于编程语言和平台的智能合约模型，用于支持这种分析. 本文提出了一种易于理解的、编程语言无关的智能合约模型，以及基于纳什均衡理论提出了一种合约缺陷检测方法.

3　智能合约的建模

承诺模型在多 Agent 领域被广泛用于对跨组织协议的建模[5,9-12]. 本文提出智能合约建模首先由用户将合同条款以“承诺”的形式描述，然后再基于用户书写的承诺自动构建智能合约. 本节介绍如何定义“承诺”，以及如何基于一组承诺自动构建智能合约的状态机.

3.1　承诺

定义 1.　承诺(commitment). 一个承诺是一个五元组 $C(x,y,p,r,tc)$，表示 x 向 y 承诺，只要前提条件 p(premise)满足，就会在 tc 规定的时间期限内完成 r(result).

定义 2.　承诺的生命周期. 一个承诺可能具有五种不同的状态，其生命周期可以用一个状态机表示，如图 1 所示.

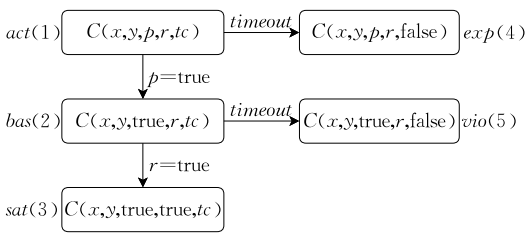


图 1　承诺生命周期

如图 1 所示，一个承诺的生命周期内可能有五种不同的状态，其中激活(act)是初始状态，终止状态集是{过期(exp)，违约(vio)，满足(sat)}：

(1) 激活(act). 当前提 p 和结果 r 的值都未知，并且 tc_{act} 处于有效期. 本文用数字 1 表示.

(2) 就绪(bas). 前提 p 为 true，结果 r 未知，并且 tc_{bas} 并未超出有效期. 此时则表示前提 p 已经达成，等待结果的完成. 本文用数字 2 表示.

(3) 满足(sat). 当一个承诺处于就绪之后，若责任方 x 完成了结果 r，该承诺进入满足(sat)状态，表示承诺完成. 本文用数字 3 表示.

(4) 过期(exp). 若承诺原本处于激活态，但如果超出了时间约束，则承诺进入过期(exp)状态. 表示承诺已经失效. 本文用数字 4 表示.

(5) 违约(vio). 承诺进入就绪态，前提 p 已经满足，但责任方 x 没有在规定的时间内完成结果 r，则承诺进入违约状态. 本文用数字 5 表示.

承诺的前提条件 p 是一个逻辑表达式，组成表达式的每个文字(literal)可以是一个布尔条件，表示某个承诺处于某个状态，例如若承诺 C_2 的前提 $p=C_1.sat$，表示必须 C_1 的状态为 sat 之后，C_2 才可以就绪. 前提的文字也可以是一个布尔函数形式的仲裁结果，表示由第三方判定某个条件是否成立. 例如，在某运输合同中有一个承诺，若出现自然灾害，则承运方无须赔偿，那么承诺的前提条件就应包括 $judge$(自然灾害). 本文使用 $judge$ 函数来表示仲裁，仲裁可以由智能合约预言机[14](监控区块链事件并能将监控结果发回智能合约)或合同双方共同指定的第三方完成. 需要指出，预言机或第三方只需要对 $judge$ 所规定的事项作出仲裁，而不是仲裁整个合同的执行状况.

r 称为承诺的结果，是一组动作的合取式.

时间约束 tc 包含两个部分，即 $tc:=(tc_{act},tc_{bas})$，分别表示当承诺进入 act 或 bas 之后，承诺应该在 tc_{act} 或 tc_{bas} 的期限内完成.

例 1.　一个简单的购书合同. 甲承诺在 3 天内付款 100 元，乙承诺在收到书款之后 1 天内发货.

C_1(甲，乙，$\varnothing$，pay(乙，100)，(∞，3))，

C_2(乙，甲，$C_1.sat$，$deliver$($book$，甲)，(3，1)).

承诺 C_1 的含义是甲向乙承诺，在 3 天内付款 100 元. 注意此承诺的前提条件为空，即不设前提. 并且此承诺在前提满足之后 3 天内必须完成动作 pay(乙，100).

承诺 C_2 的含义是，当承诺 C_1 进入 sat 状态(即：甲在 3 天内付款)，则乙承诺在 1 天内发货. 注意该承诺的时间约束的第一部分为 3 天，含义是在合约签署 3 天之后，该承诺过期. 也就是说即使甲在 3 天

之后完成承诺 C_1，乙也没有义务发货.

3.2 智能合约

定义 3. 智能合约(smart-contract). 一个智能合约是一个确定性有穷状态机(DFA)，即 $SC:=(CC,\Sigma,S,s_0,\Delta,\mathcal{F})$：

(1) $CC=\{C_1,\cdots,C_n\}$，表示一组承诺；

(2) Σ 是字母表，包括所有可触发 CC 中一个或多个承诺改变状态的事件. 一个事件可以是一个动作，某个仲裁的结果，或是某个承诺状态发生改变(比如过期)；

(3) $S=\{s_0,\cdots,s_m\}$，是有限个状态的集合. $\boldsymbol{s}_i(i=0,\cdots,m)$是一个 n 维向量，即 $\boldsymbol{s}_i=\langle C_1.state,\cdots,C_n.state\rangle(state\in\{act,bas,sat,exp,vio\})$；

(4) $s_0\in S$ 是初始状态，若一个承诺的前提为空，则其在 s_0 中状态为 bas；否则其状态为 act；

(5) $\Delta:S\times\Sigma\to S$ 是状态转移函数；一个状态转移 $(\boldsymbol{s}_i,e)\to\boldsymbol{s}_j\in\Delta$ iff

① $\boldsymbol{s}_i$ 和 $\boldsymbol{s}_j$ 两个向量只在第 λ 维上的取值不同($\lambda=1,\cdots,n$)，且

② 依据 C_λ 的定义，当事件 $e\in\Sigma$ 发生，C_λ 的状态应从 $\boldsymbol{s}_i[\lambda]$ 转变为 $\boldsymbol{s}_j[\lambda]$

(6) $\mathcal{F}\subseteq S$ 是一组接受的状态(终态)，$\forall\boldsymbol{s}_i\in\mathcal{F}$，$\boldsymbol{s}_i[k]\in\{sat,exp,vio\}(k=1,\cdots,n)$.

图 2 中展示的是根据例 1 中的承诺得到的智能合约. 如图 2 所示，合约的初始状态(根结点)为(2,1)，因为承诺 C_1 前提为空，因此其初始状态为 2(bas). 图中边上给出了改变合约状态的事件，比如在初始状态可以发生两个事件，$timeout\_bas(C_1)$ 和 pay. 前者表示 C_1 在进入 bas 之后超时，也就意味着该承诺违约，因此转变到状态(5,1)，表示承诺 C_1 的状态从 2(bas)改变为 5(vio). 后一个事件，pay，表示甲执行付款动作，使得承诺 C_1 从 2(bas)改变为 3(sat). 因为 C_1 进入到 sat，C_2 的前提得到满足，其状态自动从 1(act)变为 2(bas). 因此我们得到状态(3,2). 注意，当一个承诺的前提变为 true，其状态将自动从 1 变更为 2，无需额外事件触发. 因此本文描述中都略去了从 1 到 2 的状态变化过程.

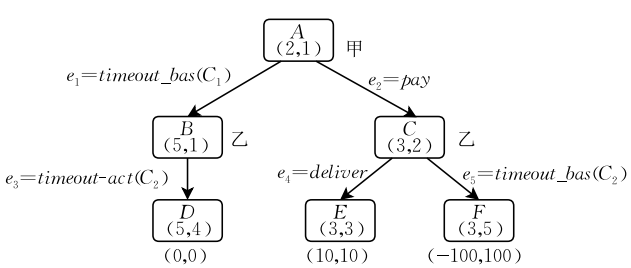


图 2 DFA 形式的智能合约

3.3 由承诺生成状态机

根据一组承诺，我们可以自动构造相应的 DFA. 算法如下.

算法 1. 创建 DFA.

输入：一组承诺 CC

输出：DFA 的根结点

```
1.  n0←initRoot(CC); //n0是DFA根结点
2.  H.push(n0); //H是一个堆栈,存放状态结点
3.  WHILE(H not empty) DO{
4.    n←H.pop();
5.    s←n.state;
6.    FOR (each s′满足 ∃e (s,e)→s′∈Δ) DO{
7.      IF (s′∈H) THEN{ //若该状态曾出现过
8.          chd←find(H,s′);
9.          addChild(n,chd);
10.     }
11.     ELSE{ //新的状态结点
12.         chd←createChild(n,s′);
13.         H.push(s′,chd);}
14.   }
15. }
16. RETURN n0;
```

算法 1 第一行初始化一个根结点 n_0，将其存入到堆栈 H 中. 当堆栈不为空，读出栈顶的结点 n 以及该结点的状态 s. 算法第 6 行遍历所有可以改变当前状态的事件 e，算法 7～13 行将孩子状态加到当前结点的孩子集合中. 第 8 行从 H 中读出已有的一个结点 chd. 第 9 行根据 chd 构造当前结点 n 的孩子. 第 12 行根据 s' 构造 n 的一个新的孩子结点.

回过头来分析例 1，这是“一手交钱，一手交货”的经典交易模式. 然而，显然例 1 是有问题的，卖方完全可以在收到货款之后不发货. 在下一节，我们将从博弈论的角度来对例 1 进行分析，并给出基于纳什均衡检测合同缺陷的方法.

4 智能合约的博弈问题

商业合同是一个典型的博弈过程，合同参与人在合同的不同状态上可能有不同的策略，以获得对自己最有利的结果. 本节将首先定义智能合约的博弈问题，再讨论如何基于博弈论的思想来自动分析合同的优劣.

定义 4. 智能合约博弈. 智能合约博弈由下列

部分组成：

(1)一个参与人的集合 N；

(2)一个状态变迁图 $M=(SC,\alpha)$，其中 SC 是状态机(见定义 3)，α 是参与人函数，它赋给 SC 中每个非终止状态一个参与人，也就是 $\alpha(s)\in N$ 指在非终止状态 s 上只有 $\alpha(s)$ 可以采取行动；

(3)对每个参与人 $i\in N$ 有一个终止状态上的效用函数 $u_i:\mathcal{F}\to\mathbb{R}^+$. $u_i(s)$ 表示参与人 i 在终止状态 $s\in\mathcal{F}$ 上的收益.

图 2 中的例子也是一个状态变迁图. 包含两个参与人 $N=\{$甲,乙$\}$. 每个非终止状态(内部结点)的行动人标在了结点旁边，例如 $\alpha(A)=$甲. 参与人甲的效用函数是 $u_1(D)=0, u_1(E)=10, u_1(F)=-100$；参与人乙的效用函数是 $u_2(D)=0, u_2(E)=10, u_2(F)=100$. 简略起见，我们将两个参与人在一个终止状态的效用值写成一个向量. 在图 2 中，结点 D、E 和 F 下方是两个参与人的效用值，甲在前，乙在后.

定义 5.　智能合约博弈$\langle N,M,\alpha,(u_i)\rangle$中，第 i 个参与人可以采取的策略集合是笛卡尔积 $\Omega_i=\times_{s\in S\setminus\mathcal{F},\alpha(s)=i}\{e\mid(\boldsymbol{s}_i,e)\to\boldsymbol{s}_j\in\Delta\}$. 一个策略组合是 $\gamma=(\omega_1,\cdots,\omega_N)\in\times_{i\in N}\Omega_i$，其中 $\omega_i\in\Omega_i$ 是参与人 $i\in N$ 策略集合的一个策略，即第 i 个参与人在各个内部结点上所选择的动作.

对于每个承诺 $C(x,y,p,r,tc)$，责任方 x 是发出承诺的一方. 那么对于 C，当前提 p 为真时，x 就有两个选项，履行承诺或者违约. 当前提 p 为假时，x 只有一个被动的动作选项，即让承诺过期.

例如，图 2 中，结点 A 对应承诺 C_1(**甲**,**乙**,$\varnothing$, pay(**乙**,100),$(\infty,3)$). 由于前提条件恒为真，甲在结点 A 上可以选择执行的动作为$\{e_1,e_2\}$，即 $timeout_bas(C_1)$和 pay. 第一个动作表示甲在约定期限内不付款，使得该承诺违约. 第二个动作表示甲选择付款，履行承诺. 因此在结点 A，甲的策略集合是 $\Omega_1=\{e_1,e_2\}$. 类似地，乙在结点 B 可以执行的动作有$\{e_3\}$，在结点 C 可以执行的动作为$\{e_4,e_5\}$. 那么乙的策略集合是笛卡尔积 $\Omega_2=\{e_3\}\times\{e_4,e_5\}=\{e_3e_4,e_3e_5\}$. 这里，$e_3e_4$ 是乙的一个策略，含义是他在结点 B 时会做动作 e_3，在结点 C 会做动作 e_4.

参与人策略集合的笛卡尔积就是所有可能的策略组合集合. 例如，根据甲和乙的策略集合，有一个策略组合(e_1,e_3e_4)，其含义是：甲在结点 A 时会做动作 e_1，乙在结点 B 时会做动作 e_3，在结点 C 时会做动作 e_4.

对于每个策略组合 γ，在状态变迁图中都能确定并且唯一确定一条从初始结点到终止结点的动作序列. 例如，对于策略组合(e_1,e_3e_4)，在图 2 中 A 是初始状态，甲做动作 e_1，合约状态将改变为状态 B. 按照策略组合的内容，乙将在状态 B 做动作 e_3，合约将进入到终止状态 D. 注意，没有进入状态 C，因此乙也就没有做动作 e_4. 也就是说，在一个策略组合中并非所有的策略都会真正执行.

对于参与人来说，他们的收益只与终止状态有关，因此会关心动作序列的终止状态. 我们把动作序列表示为甲$\xrightarrow{e_1}$乙$\xrightarrow{e_3}D$，其中 D 是一个终止状态，甲$\xrightarrow{e_1}$ 表示参与人甲执行动作 e_1.

定义 6.　智能合约博弈$\langle N,M,\alpha,(u_i)\rangle$的策略形式表示为$\langle N,M,(\Omega_i),(u_i)\rangle$，对于每个参与人 $\forall i\in N$ 有：

(1)Ω_i是参与人 $i\in N$ 的策略集合；

(2)$u_i:\times_{i\in N}\Omega_i\to\mathbb{R}^+$是参与人 i 在策略组合上的效用函数. 任意一个策略组合 $\forall\gamma\in\times_{i\in N}\Omega_i$，在状态变迁图 M 中确定并唯一确定一条终止状态是 s 的动作序列，因此有 $u_i(\gamma)=u_i(s)$($u_i(s)$是参与人 i 在终止状态 s 上的效用).

由于例 1 中只有两个参与人，其博弈问题的策略形式可以通过效用矩阵表示(甲是行参与人，乙是列参与人)，如表 1 所示. 注意，每个策略组合对应矩阵中一个元素. 例如，策略组合(e_1,e_3e_4)对应左上角的元素. 该策略组合在图 2 中确定了一条唯一的动作序列 甲$\xrightarrow{e_1}$乙$\xrightarrow{e_3}D$. 因为有 $u_1(D)=0$ 和 $u_2(D)=0$，左上角的效用值为(0,0).

表 1　效用矩阵

甲	乙	
	e_3e_4	e_3e_5
e_1	0,0	**0,0**
e_2	10,10	−100,100

定义 7.　纳什均衡. 智能合约博弈$\langle N,M,\alpha,(u_i)\rangle$的纳什均衡是一个策略组合 $\gamma^*=(\omega_1^*,\cdots,\omega_N^*)\in\times_{i\in N}\Omega_i$，使得对每个参与人 $\forall i\in N$ 的策略 $\forall\omega_i\in\Omega_i$ 有

$$u_i(\omega_{-i}^*,\omega_i^*)\geq u_i(\omega_{-i}^*,\omega_i).$$

上式中，ω_{-i}^*表示策略组合 γ^* 中去掉参与人 i 的策略 ω_i^* 之后其他参与人的策略. (ω_{-i}^*,ω_i)表示动作人 i 选择任意策略，其他参与人选择 ω_{-i}^*中规定的策略. 也就是对于任何一个参与者 i 而言，当其他每个参与者都选择均衡策略 ω_{-i}^*时，参与者 i 所有的策略产生的效用都不高于他选择策略 ω_i^* 所产生的效用.

《计算机学报》

例如,从表 1 中可以看到对双方都有利的策略组合是(e_2,e_3e_4),在图 2 中对应着动作序列甲$\xrightarrow{e_2}$乙$\xrightarrow{e_4}E$,含义是甲付款,且乙发货,双方各自可以得到 10 的效用值.然而,这个策略组合不是一个纳什均衡,因为乙可以改变他的策略为 e_3e_5,也就是不发货,使得乙的效用值从 10 提高到 100.理性的乙在这种情况下会选择不发货,因此甲付款且乙发货这样的理想局面无法出现.根据表 1 可以计算出只有一个策略组合是纳什均衡,即右上角的(e_1,e_3e_5).该策略组合在图 2 中对应着一条动作序列甲$\xrightarrow{e_1}$乙$\xrightarrow{e_3}D$,含义是甲不付款,且乙不发货,双方的效用值都是 0.这是一个纳什均衡,因为双方都无法通过单方面改变策略来获得更多的收益:如果甲将他的策略改变为 e_2,他的效用值会从 0 降低到 -100;如果乙将他的策略改变为 e_3e_4,他的效用值保持不变,依然是 0.

根据纳什均衡理论[6,15-16],如果所有参与人都是理性的,完全了解效用矩阵,并且存在纳什均衡,参与人会选择进入到纳什均衡的局面.这是因为参与人在做策略选择时,当预计到可能会进入到非纳什均衡的局面时,一些参与人知道他们可以通过改变策略获得更多的效用值.由于他们都是理性的,他们会改变策略,因此非纳什均衡这个局面并不稳定.由于存在纳什均衡,这些参与人策略最终会收敛到纳什均衡策略,因此会进入纳什均衡局面.例如,在前面的例子中,假设甲乙都是理性的,他们也知道表 1 的信息.如果智能合约预计会进入到非纳什均衡局面(e_2,e_3e_4),乙会将策略从 e_3e_4 改变到 e_3e_5,因为他的效用值会从 10 提高到 100,但甲的效用值会从 10 降低到 -100.但智能合约预计又会进入到另一个非纳什均衡局面(e_2,e_3e_5),甲又会将策略从 e_2 改变到 e_1,因为他的效用值会从 -100 提高到 0,处于纳什均衡局面(e_1,e_3e_5),也就是甲乙都选择不合作.

当然,现实情况中,可能由于参与人的非理性行为或者不了解效用矩阵,会进入到非纳什均衡的局面.但本文考虑的是智能合约本身的问题,因此假设参与人都是理性和完全了解信息的.在前面例子中,甲乙只有选择不合作,显然这样的智能合约是有逻辑缺陷的.

定义 8. 智能合约逻辑缺陷.智能合约逻辑缺陷是一种使得参与人效用小于或等于零的纳什均衡 $\gamma^*\in\times_{i\in N}\Omega_i$,也就是:存在参与人 $\exists i\in N$,他从纳什均衡 γ^* 获得的效用小于或等于零,即 $u_i(\gamma^*)\leq 0$.

基于状态变迁图,我们可以通过计算纳什均衡来识别出智能合约的逻辑缺陷,然后自动分析出智能合约的优劣.然而,状态变迁图存在组合爆炸的问题.图中的结点数量与合同中承诺数量成指数关系.当合同承诺数量较多时,将难以得到分析结果.接下来,我们将讨论如何提高计算纳什均衡的效率,这主要通过简化策略形式来实现.

5 策略形式的化简

给定一个合约,其策略形式本质上是状态变迁图中各个结点的策略集合得到的笛卡尔积.因此,对策略形式的化简,首先考虑的是对状态变迁图进行化简.此外,我们还将讨论,如何在生成策略形式时,避免生成冗余的策略组合.

本节将首先介绍“等价策略”合并原则,在此基础上提出两个基于该原则的化简方法:基于效用值相同叶结点的化简方法,以及基于独立承诺的化简方法.接下来,我们将讨论一种新方法,该方法可以突破等价策略的局限,对状态变迁图进行进一步化简.最后,我们将提出一个发现冗余策略组合的方法,避免一部分冗余策略组合的生成.

5.1 等价策略合并原则

在经典的博弈论理论中[6],等价策略合并原则是最为常用的对策略形式进行化简的方法.

定理 1. 等价策略合并原则.一个策略形式 $\langle N,M,(\Omega_i),(u_i)\rangle$,若对于某参与人 $k\in N$ 的两个策略 ω_k^1 和 ω_k^2,

$$\boldsymbol{u}(\omega_1,\cdots\omega_k^1,\cdots,\omega_N)=\boldsymbol{u}(\omega_1,\cdots\omega_k^2,\cdots,\omega_N),\quad \forall\omega_i\in\Omega_i(i\neq k)$$

即给定其他参与人的任何策略,分别与 ω_k^1 和 ω_k^2 构成的两个策略组合,这两个策略组合的效用值对于每个参与人均相等.注意:对于一个策略组合 $\gamma=(\omega_1,\cdots,\omega_N)$,$\boldsymbol{u}(\gamma)=(u_1(\gamma),\cdots,u_N(\gamma))$ 是一个 N 维向量,表示所有参与人在策略组合 γ 上的效用值.

根据文献[6],若 ω_k^1 和 ω_k^2 满足上述条件,则任何两个策略组合 γ 和 γ',若 $\omega_k^1\in\gamma$ 且 $\omega_k^2\in\gamma'$,则可以将 γ 和 γ' 合并为一个策略,而不影响纳什均衡的计算.

当只有两个参与人时,上述条件成立等价于效用矩阵中有两个相同的行或列,因此也可以将其称为“行(列)相同合并原则”.

接下来,我们讨论利用这一原则对状态变迁图进行化简.

5.2　效用值相同叶结点合并方法

定理 2.　效用值相同叶结点合并规则. 若多个叶结点 $l_1,\cdots,l_m$ 属于同一个非终止状态结点的孩子，且这些叶结点上每个参与人的效用值都相等，则将 $l_1,\cdots,l_m$ 合并将不影响纳什均衡的求解.

证明.　不失一般性，设参与人为甲，设 $e_1,\cdots,e_m$ 分别为从父结点转移到 $l_1,\cdots,l_m$ 的事件（也是甲的动作）. $\forall i,j\in\{1,\cdots,m\}$，假设 γ_i 和 γ_j 为任意两个分别包含 e_i 和 e_j 的策略组合，且 γ_i 和 γ_j 除去 e_i 和 e_j 之外其余部分完全相同，即 $\gamma_i/e_i=\gamma_j/e_j$.

若 e_i 和 e_j 在策略组合中均不生效（没有包含在对应的动作序列中），γ_i 和 γ_j 必然对应相同的执行路径，效用值相等.

若 e_i 和 e_j 有一个生效，说明 γ_i 和 γ_j 对应的动作序列将到达 $l_1,\cdots,l_m$ 的父结点，则 e_i 和 e_j 都会生效. 因此 γ_i 和 γ_j 对应的动作序列必然终止于 e_i 和 e_j 指向的叶结点，效用值也相同. 因此 γ_i 和 γ_j 的效用值必然相等，按等价策略合并原则，将 γ_i 和 γ_j 合并将不影响纳什均衡的计算.　　证毕.

依据定理 2，我们可以将图 3 中左侧的两个叶结点予以合并.

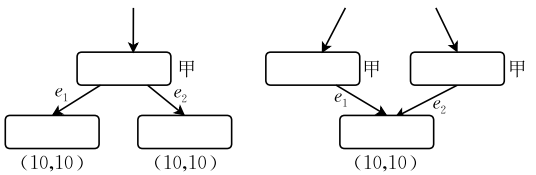


图 3　基于效用值相同叶结点进行化简的例子

定理 3.　内部结点合并规则. 若多个内部结点 $v_1,\cdots,v_m$ 对应同一个参与人，它们均只有一个孩子结点，且它们的孩子结点为同一个叶结点，则 $v_1,\cdots,v_m$ 可以合并，同样不影响纳什均衡的求解.

证明过程类似定理 2，限于篇幅，略去细节.

依据定理 3，我们可将图 3 中右侧图中倒数第二层的两个结点予以合并.

5.3　基于因果独立关系的化简方法

5.3.1　因果关系和因果独立

根据前面的分析，我们知道例 1 的合约是存在漏洞的，不利于建立双方长期合作. 原因在于没有对乙的行为进行约束. 因此，我们现在对例 1 进行完善：仿照支付宝和以太坊的思路，加入资金监管. 购书款先存入到合约地址，当乙履行发货义务，才将购书款转给乙. 修改后的合约承诺如下.

例 2.　第二版购书合同. 甲承诺在 3 天将书款 100 元存入到合约账户；乙承诺在甲付款1天内发货；甲承诺在乙发货 7 天内确认收货；Contract 承诺在甲确认收货 1 天内将书款转账给乙；Contract 承诺在乙违约（不发货）1 天内将书款退回给甲.

注意 Contract 的动作是智能合约代码规定的，一旦前提满足，就必然会执行. 因此 Contract 不会出现违约.

观察例 2 中的承诺，我们可以发现一些承诺之间存在依赖. 比如，承诺 C_2 的前提包含了 C_1，因此必须等 C_1 进入终止态之后才能判断 C_2 是否可以进入 *bas* 状态. 下面，我们引入因果关系的概念来描述这种依赖关系.

定义 9.　因果图（Carsal Graph）. 给定一组承诺 CC，其相应的因果图 G 定义如下：

（1）CC 中每一个承诺 C_i 对应 G 中一个顶点；

（2）如果一个承诺 C_j 的前提中包含 C_i，则在 G 中有一条有向边从 C_i 指向 C_j.

图 4 展示的是例 2 中承诺所对应的因果图.

C_1(甲,乙,∅,*deposit*(*Contract*,100),(∞,3))
C_2(乙,甲,C_1.*sat*,*deliver*(*book*,甲),(3,1))
C_3(甲,乙,C_2.*sat*,*confirm*(receive_book),(7,7))
C_4(*Contract*,乙,C_3.*sat*,*transfer*(乙,100),(30,1))
C_5(*Contract*,甲,C_3.*vio*,*refund*(甲,100),(30,1))

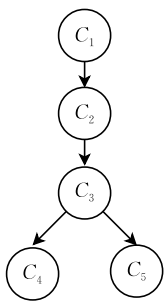


图 4　因果图例子

定义 10.　因果关系（Carsal Relation）. 给定两个承诺 C_i 和 C_j，定义 C_i 和 C_j 存在因果关系当且仅当在因果图 G 中存在至少一条路径连接 C_i 和 C_j. 若 C_i 和 C_j 不存在因果关系，我们称它们是因果独立（carsal independent）的，或简称独立.

注意，随着合约的执行，某些承诺的状态可能发生改变，若一个承诺 C_i 进入某个终止态，我们须将 C_i 对应的顶点和与其相连的边都从 G 中删除. 因此，可能存在 C_i 和 C_j 原本存在因果关系，但当合约执行到某个状态时，它们变成了独立. 如图 4 所示，C_4 和 C_5 本来是存在因果关系的，但如果 C_3 进入了终态并从图中移去，则 C_4 和 C_5 独立.

《计算机学报》

5.3.2 复合状态转移

若在因果图中存在一条从 C_i 到 C_j 的路径，则 C_j 需要等待 C_i 的执行结果才能发生状态改变. 但如果 C_i 和 C_j 是独立的，则它们的状态改变可以以任意顺序发生. 例如，假设当前合约状态为 s，并假设此时 C_i 和 C_j 都可以发生状态改变. 假设事件 e_i 使得承诺 C_i 发生改变，并引起状态转移 $\delta_1:(s,e_i)\rightarrow s_1$. 事件 e_j 使得承诺 C_j 发生改变，并引起状态转移 $\delta_2:(s_1,e_j)\rightarrow s_2$.

也就是说，通过状态转移序列 $\delta_1\delta_2$，可将合约状态从 s 转变到 s_2. 同理，若我们先改变 C_j，再改变 C_i，也就是执行状态转移序列 $\delta_2\delta_1$，一样可以将合约状态从 s 转变到 s_2.

上面的例子可以看出，当相互独立的承诺发生状态改变时，我们可以将多个状态改变合并到一起，避免展开不必要的状态转移序列. 由于状态变迁图中一个结点只有一个动作人，我们在合并状态转移时要求被合并的状态转移由同一个动作人发起.

定义 11. 复合状态转移(Combined Transition). 设合约状态为 s，给定一组两两独立的承诺 $\{C_1,C_2,\cdots,C_m\}(m\geq 2)$，若这些承诺此时均能发生状态改变. 不失一般性，假设这些承诺依次改变状态，产生的状态转移序列为 $\delta_1,\delta_2,\cdots,\delta_m$，并最终得到合约状态 s'，且这些状态转移对应的事件分别为 $\{e_1,e_2,\cdots,e_m\}$. 我们可以将这些状态转移合并为一个复合状态转移 $\delta^{\oplus}:(s,[e_1,e_2,\cdots,e_m])\rightarrow s'$. 这里，$[e_1,e_2,\cdots,e_m]$ 表示这 m 个事件的任意排列.

需要指出，对于 $\{e_1,e_2,\cdots,e_m\}$ 的任意一个序列 φ，$(s,\varphi)\rightarrow s'$. 因为这些事件可以在状态 s 时独立发生，其顺序不会影响结果.

图 5 的左侧展示了根据例 2 承诺生成的状态变迁图的一部分. 可以看到当 C_3 进入 *sat* 状态之后，C_4 和 C_5 可以独立改变自己状态. 当事件 *transfer* 发生，C_4 从 2 改变到 3；当事件 *timeout_act*(C_5) 发生，C_5 从 1 改变到 4. 从图中可以看到，这两个事件产生的两个序列，最终得到相同的结果. 因此，我们可以将这两个状态转移合并，得到右边的图.

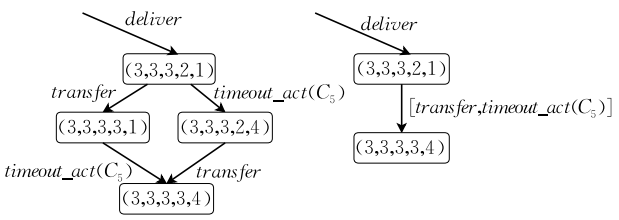


图 5 复合状态转移例子

定理 4. 基于复合状态转移的化简原则. 当状态变迁图的结点 v 存在一组两两独立的承诺 $\{C_1,C_2,\cdots,C_m\}(m\geq 2)$，按定义 11 将 v 出发的多个状态转移路径合并为一个复合状态转移，不会影响纳什均衡的计算结果.

证明. 设化简之前的状态变迁图为 G，通过一个复合状态转移 $e^{\oplus}$ 对 G 进行化简之后的状态变迁图为 G'. 设 γ 为任意一个由 G' 生成且包含 $e^{\oplus}$ 的策略组合. 则 γ 在 G' 中对应唯一一条动作序列 ρ 指向叶子结点 l，其对应的效用值为 $u(l)$.

设 $e^{\oplus}$ 在 G 上对应的事件序列集合为 $[e^{\oplus}]$. 并设 ρ 在 G 上对应的路径集合为 P.

若 $e^{\oplus}$ 不在 ρ 上，则任何属于 $[e^{\oplus}]$ 的事件序列都不会出现在任何一条 P 的路径上.

若 $e^{\oplus}$ 在 ρ 上，则属于 $[e^{\oplus}]$ 的事件序列都在 P 的某条路径 ρ' 上. ρ 和 ρ' 的唯一差别在于复合状态转移部分，它们将指向相同的叶结点. 也就是说，包含这些序列的策略组合，其效用值均为 $u(l)$.

根据等价策略合并原则可证. 证毕.

化简效果分析. 若一个状态 s 包含 k 个独立承诺，那么未化简前的状态变迁图将产生 $k!$ 个状态转移序列，但是化简之后，这 $k!$ 个序列(路径)将被合并为 1 个. 因此，定理 4 可以大大减少状态变迁图结点的数量，也将大幅降低策略形式的数量.

5.4 基于纳什均衡等价的化简方法

之前讨论的化简方法的基础都是等价策略(行列相同)合并原则，但是等价策略合并不是万能的. 图 6(a)的例子中，结点 C、D、E 结构非常类似：有相同的父结点 A；孩子结点都是 F 和 G，且转移到 F 或 G 的事件也相同. 我们希望将 C、D、E 合并为一个结点，如图 6(b)所示.

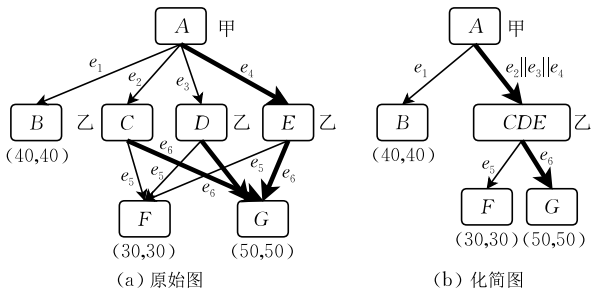


图 6 状态变迁图等价

图 6(a)的效用矩阵如表 2 所示(表中黑体部分为纳什均衡的策略组合). 当我们分析其效用矩阵时，发现并不存在相同的行或列. 无法适用 5.1 小节里的等价策略原则(也就是行列合并原则).

表 2　图 6(a)的效用矩阵

乙	甲			
	e_1	e_2	e_3	e_4
$e_5(C)e_5(D)e_5(E)$	**(40,40)**	(30,30)	(30,30)	(30,30)
$e_5(C)e_5(D)e_6(E)$	(40,40)	(30,30)	(30,30)	**(50,50)**
$e_5(C)e_6(D)e_5(E)$	(40,40)	(30,30)	**(50,50)**	(30,30)
$e_5(C)e_6(D)e_6(E)$	(40,40)	(30,30)	**(50,50)**	**(50,50)**
$e_6(C)e_5(D)e_5(E)$	(40,40)	**(50,50)**	(30,30)	(30,30)
$e_6(C)e_5(D)e_6(E)$	(40,40)	**(50,50)**	(30,30)	**(50,50)**
$e_6(C)e_6(D)e_5(E)$	(40,40)	**(50,50)**	**(50,50)**	(30,30)
$e_6(C)e_6(D)e_6(E)$	(40,40)	**(50,50)**	**(50,50)**	**(50,50)**

但是我们观察化简图的效用矩阵，发现一个有趣的现象.

首先，虽然不能适用行(列)相同合并原则，但是在 CDE 三个结点上，乙采取的策略是完全相同的：选择 e_5 则到达 F，选择 e_6 则到达 G.

其次，在化简图中得到两个策略组合(表中黑体标识). 如表 3 所示，其中左上角的策略组合 $\{e_5,e_1\}$ 与表 2 左上角的策略组合相同，在表 2 中该组合也为纳什均衡点. 右下角的策略组合 $\{e_6,e_2\|e_3\|e_4\}$ 含义是甲在 e_2、e_3 或 e_4 中任选一个，乙始终选择 e_6. 可以逐一验证，这个策略组合与表 2 中标为黑体且值为(50，50)的 12 个策略组合是一致的. 比如 $\{e_6(C)e_6(D)e_6(E),e_4\}$，因为甲选择了 e_4，乙的有效动作就是 $e_6(E)$，也就是说可以简化为 $\{e_6,e_4\}$.

表 3　图 6(b)的效用矩阵

乙	甲	
	e_1	$e_2\|e_3\|e_4$
e_5	**(40,40)**	(30,30)
e_6	(40,40)	**(50,50)**

最后，我们观察 $\{e_6,e_2\|e_3\|e_4\}$ 所对应的执行路径(图中加粗的箭头所示)，该路径在原图中对应三条执行路径(同样是加粗箭头标识)，容易验证，这三条路径对应的策略组合均为纳什均衡点.

基于以上观察，我们提出以下基于纳什均衡等价的化简方法.

定理 5.　基于纳什均衡等价的化简规则. 若结点集合 $v_1,\cdots,v_m(m\geq 1)$ 有同一个父亲结点 v^p，v^p 到 $v_i(i=1,\cdots,m)$ 的动作分别为 e_i；$\forall i\in\{1,\cdots,m\}$，$v_i$ 的孩子结点集合均为 $l_1,\cdots,l_n$(叶结点)，v_i 到 $l_j(j=1,\cdots,n)$ 的动作为 e'_j. 且 $v_1,\cdots,v_m$ 的动作执行人相同. 则可将 $v_1,\cdots,v_m$ 合并为一个结点 v，v^p 到 v 的动作为 $e_1\|\cdots\|e_m$，v 到孩子结点 l_j 的动作为 e'_j.

说明：依据定理 5 进行化简，我们将得到包含可选动作，例如 $e_1\|\cdots\|e_m$ 的执行路径，其在原图中对应 m 个路径，即将 $e_1\|\cdots\|e_m$ 拆开成单独的动作，而其他部分保持不变. 例如化简图中路径甲 $\xrightarrow{e_2\|e_3\|e_4}$ 乙 $\xrightarrow{e_6}$ G，对应原图三条路径，甲 $\xrightarrow{e_2}$ 乙 $\xrightarrow{e_6}$ G，甲 $\xrightarrow{e_3}$ 乙 $\xrightarrow{e_6}$ G 和 甲 $\xrightarrow{e_4}$ 乙 $\xrightarrow{e_6}$ G.

证明.

(⇒) 设基于定理 5 的化简图计算得到纳什均衡点 γ，且 γ 在化简图中对应路径 ρ. 将 ρ 中可选动作集 $e_1\|\cdots\|e_m$ 拆开，可得到原图中的路径集合 P. 不妨假设 ρ 是根到叶结点 l_j，则 $\gamma=\varphi(e_1\|\cdots\|e_m)e'_j$ (φ 表示任意动作序列). 既然 $\varphi(e_1\|\cdots\|e_m)e'_j$ 满足纳什均衡条件，$\forall i\in\{1,\cdots,m\}$，$\gamma_i=\varphi e_i e'_j$ 也必然满足.

(⇐) 反过来，$\forall i1,i2\in\{1,\cdots,m\}$，若 $\gamma_{i1}=\varphi e_{i1}e'_j$ 是纳什均衡点，注意 $\gamma_{i1}=\varphi e_{i1}e'_j$ 和 $\gamma_{i2}=\varphi e_{i2}e'_j$ 是在结点 v^ρ 之后才分开. 不妨假设 $v_1,\cdots,v_m$ 的动作人为甲，对于甲在 γ_{i1} 的策略集，其他人的最优策略就是 $\gamma_{i1\_甲}$. 而 $\gamma_{i1\_甲}$ 和 $\gamma_{i2\_甲}$ 差别仅仅在于 γ_{i1} 在结点 v^ρ 采取了 e_{i1}，而 γ_{i2} 在结点 v^ρ 采取了 e_{i2}. 因为甲的后续动作均为 e'_j，它们的效用值是一样的. 因此，γ_{i2} 也必然是纳什均衡点. 这样我们就得到，$\forall i\in\{1,\cdots,m\}$，$\gamma_i=\varphi e_i e'_j$，这些策略组合要么都是纳什均衡点，要么都不是纳什均衡点.

若策略组合集 $\{\gamma_i=\varphi e_i e'_j\mid i=1,\cdots,m\}$ 中每一个都是纳什均衡点，容易看出 $\gamma=\varphi(e_1\|\cdots\|e_m)e'_j$ 也必然满足纳什均衡条件.

若策略组合集 $\{\gamma_i=\varphi e_i e'_j\mid i=1,\cdots,m\}$ 都不是纳什均衡点，则依据(⇒)的逆否命题，$\gamma=\varphi(e_1\|\cdots\|e_m)e'_j$ 也不可能是纳什均衡点.　　证毕.

讨论：经过上述化简，状态转移图的规模大大缩小，但是在基于状态转移图求纳什均衡的过程中依然遇到了很大问题. 原因在于求策略形式过程中，需要做两次笛卡尔积. 接下来，我们将从另一个角度来简化策略形式，即在构建策略组合时，避免生成冗余的组合.

5.5　冗余策略形式化简

依据策略形式的定义，我们需要计算一个参与人在所有内部结点的动作集合的笛卡尔积，以得到该参与人所有策略. 例如图 7，甲的所有策略为 $\{e_1e_3,e_1e_4,e_2e_3,e_2e_4\}$，即甲在 A 的动作集合 $\{e_1,e_2\}$

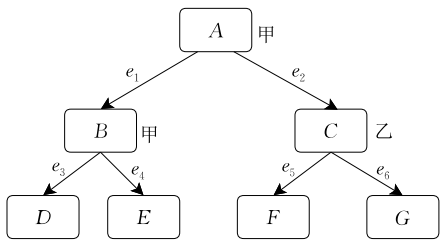


图 7　冗余策略形式例子

《计算机学报》

与他在 B 的动作集合$\{e_3,e_4\}$做笛卡尔积.

从图 7 中我们发现,如果甲在结点 A 选择了 e_2,那么动作 e_3,e_4 必然不会生效,因此 e_2e_3,e_2e_4 可能是冗余的.基于此观察,我们提出下面的化简方法.

注意到一个动作 e 实际对应状态变迁图的一条边,为了便于描述,不妨设 e 的源结点为 $src(e)$,目标结点为 $dst(e)$.

定理 6. 若某动作人在不同结点有两个策略 e_α和e_β,$src(e_\alpha)$是 $src(e_\beta)$的祖先,且 e_β不在以 $dst(e_\alpha)$为根的子树中,则所有同时包含 e_α和 e_β的策略组合都是冗余的.

证明. 假设一个策略组合 γ 同时包含 e_α和 e_β.

若 e_α生效,说明 γ 对应的路径将经过边 e_α,由于 e_β不在以 $dst(e_\alpha)$为根的子树中,则 e_β不可能生效.那么 γ 可以被另一个更短的策略组合 γ/e_β替代.

若 e_β生效,说明 γ 对应的路径将经过边 e_β,则 e_α不可能生效.那么 γ 可以被另一个策略组合 γ/e_α替代.

若 e_α和 e_β均不生效,则 γ 可以被另一个策略组合 $\gamma/\{e_\alpha,e_\beta\}$替代.

综上所述,任何同时包含 e_α和 e_β的策略组合都是冗余的. 证毕.

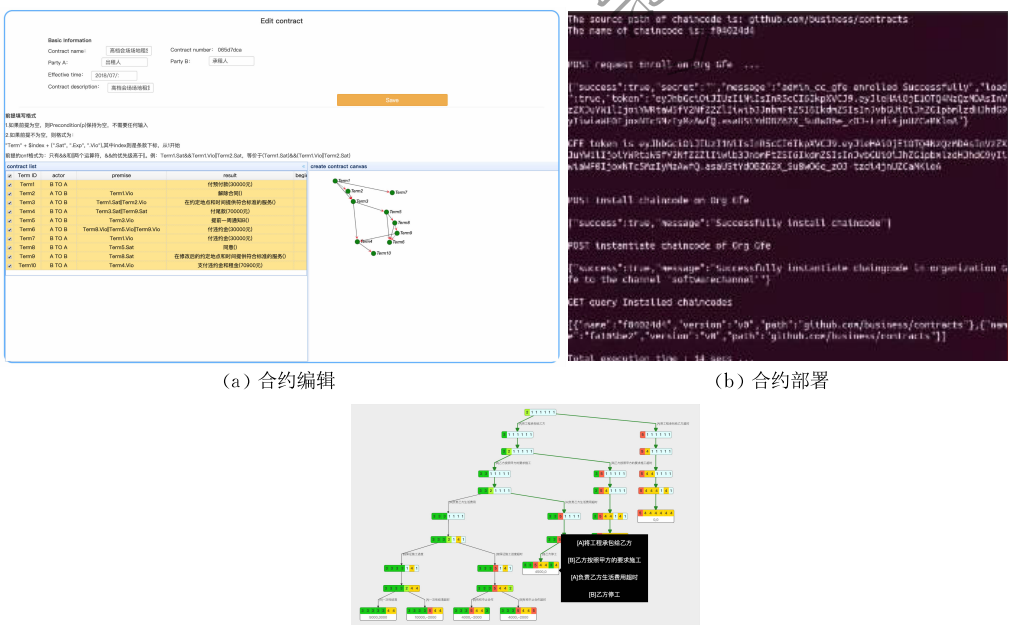
实际应用中,策略组合的数量可能非常大,难以在短时间内全部构造出来.上述定理可以帮助我们避免生成不必要的策略组合,而不是在生成所有策略组合之后再化简.

6 系统实现和实验分析

本节将首先介绍我们开发的一个用于智能合约建模、分析以及部署到区块链平台的工具.接下来介绍我们对一组真实合同分析的结果,以及我们所提出方法在性能提升方面的表现.

6.1 系统介绍

我们开发了一个用于智能合约建模、分析及自动部署的工具.图 8 是部分截图.(a)展示了一个合约的编辑界面,用户可以输入合约的承诺,系统会自动展现承诺之间的依赖关系(因果依赖图);(b)展示的是将一个合约自动部署到 Fabric 平台的运行结果;(c)是合约分析界面,用户可以看到根据合约承诺自动构建的状态变迁图.在输入效用值之后,还可以自动分析合约的纳什均衡点,其中黑色文字框展示了合约的一个执行路径.我们的工具还可以自动根据状态变迁图生成智能合约的代码,例如 Hyperledger Fabric 链码.用户将其中部分动作函数补充完整之后,就可以使用我们的工具将合约部署到 Fabric 平台上.



(a) 合约编辑 (b) 合约部署

(c) 合约分析

图 8 智能合约建模、分析及部署工具

我们采用了经典的 Lemke-Howson 算法[15]来求解纳什均衡.

6.2 有效性实验

依据《中华人民共和国合同法》(1999 年 3 月 15 日

颁布），常见的合同共有买卖合同、供用合同、赠与合同等 15 类. 我们参照华律网（66law.cn）和合同帮（hetongb.com）两个网站的合同范本，使用我们开发的工具对其中 14 类合同进行了建模和分析. 只有居间合同因为涉及到多方参与，暂时不支持.

我们总计对 14 大类 50 个合同进行了建模和分析. 并根据分析结果将这些合同分为三个等级.

若一个纳什均衡点，参与各方效用（收益）均为零，我们称为全零；若参与各方效用均为正，称为全正；若有正有负，称为正负；这三类之外的归入其他.

如图 9 所示，在合同范本中，大部分合同质量是合格的，其中有 35% 的合同达到了优秀. 这些合同中，因为只有全零和全正的纳什均衡点，参与各方会优先考虑“双赢”的博弈策略.

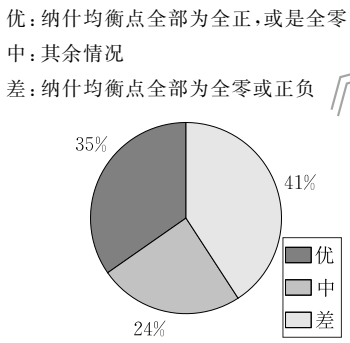


图 9　合同分析结果

但我们也看到，有 41% 的合同存在问题. 这些合同中，大部分是只存在全零的纳什均衡点. 此时，参与各方在综合利弊之后，将很可能选择不参与博弈，也就是不执行合同.

下面我们分析一个评分为“差”的合同.

合同实例：消费寄托合同

C_1（甲，乙，∅，寄物品&&付寄托金，(7,1)）

C_2（乙，甲，$C_1.sat$，受领，(∞,1)）

C_3（乙，甲，$C_2.sat$，使用寄托物，(∞,30)）

C_4（乙，甲，$C_2.sat$&&$judge$（寄托满 30 天），返还，(∞,3)）

消费寄托合同的含义是甲方将货物寄托在乙方处，并支付寄托金. 乙方可以在寄托期间使用部分寄托物品（比如煤炭），但是当寄托期满，乙方必须补足原有物品数量并归还给甲方.

上述合约的含义是：甲方将物品寄托给乙方并且付给乙方 3000 元寄托费（C_1），乙方决定是否受领寄托物（C_2）. 乙方可以自由消费寄托物（C_3），但是必须在寄托时间到期（由 $judge$ 函数调用预言机实现）后将寄托物全额返还给甲方（C_4）.

如果一开始甲方就没有将物品寄托给乙方，双方受益均为 0. 或者甲方想要将物品寄托给乙方，但是乙方不受领寄托物，双方受益也均为 0. 但是当寄托物一旦成功寄托后，乙方可以自由选择是否执行承诺 C_4，如果乙方违约，不将寄托物返还给甲方，这能使乙方自己获得更高的收益.

因此，上述合同只有全零的纳什均衡点. 甲如果选择寄托物品，那么作为理性的博弈参与人，乙将选择使自己收益最大的策略——违约. 那么甲在分析利弊之后，最好的对策是不寄托物品. 也就意味着合同根本不会被执行. 这样的合同不利于双方重复博弈.

需要指出的是，我们通过自动分析，发现很多合同范本都存在上述问题，没有对某参与方的违约行为予以惩罚. 那么合同是否顺利执行可能只有取决于该参与方的道德水平或法律诉讼. 从商务合同的原则来看，这显然是不合理的.

可以看出，我们提出的建模方法可以对现实中大多数合同进行建模；并且，基于博弈论对合同自动分析能够发现潜在有逻辑缺陷（不公平）的合同.

6.3　效率实验

我们按状态结点数量将合同分为四类，结点数量分别为[1,50]，[51,100]，[101,200]以及[201,∞). 状态结点的数量基本反映了合同的复杂程度.

我们对比了四类方法：

（1）DFA 方法. 采用基本的智能合约建模方法，未做任何化简.

（2）SE 方法. 基于策略等价（Strategy Equivalence，SE）原则对状态变迁图进行化简.

（3）SE＋NE 方法. 采用了 SE 方法和纳什均衡等价原则（Nash Equivalence，NE）对状态变迁图化简.

（4）SE＋NE＋RS 方法. 采用以上两类化简方法基础上，还采用了冗余策略组合化简方法（Redundant Strategy，RS）.

下面我们分别从状态结点个数、策略组合个数以及计算纳什均衡时间三个方面验证我们所提出方法的有效性.

图 10 对比了 DFA、SE 和 SE＋NE 这三种方法在四类合同中产生的状态结点数量. 可见随着合同趋于复杂，化简方法效果越明显. 特别地，当合同较为复杂，化简之后的结点数量只有化简前的约 1%. 由于最后一种化简方法（RS）并不会减少状态结点个数，我们没有在图 10 中列出来.

接下来，我们深入分析三种化简方法在计算纳什均衡过程中的效果. 由于 DFA 方法只能计算出

《计算机学报》

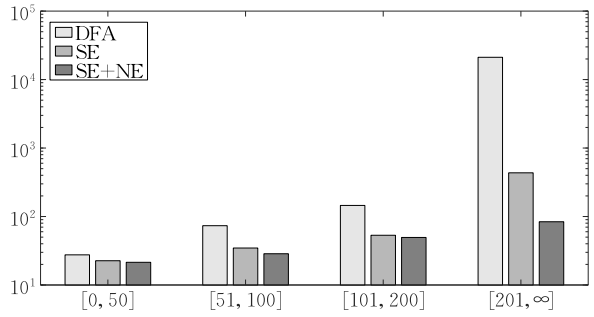


图 10 状态结点个数

极少数非常简单合同的纳什均衡，在后续分析中，不再列出 DFA 方法的结果.

图 11 对比了 SE、SE+NE 和 SE+NE+RS 这三种方法在四类合同中产生的策略组合数量(对数坐标). 可以明显看到，叠加了冗余策略化简之后，策略组合数量明显下降，对于复杂的合同，SE+NE+RS 方法的策略组合数比 SE+NE 方法少了 6 个数量级，而 SE 方法将会产生极多的策略组合，以至于根本不能在有效时间内计算出纳什均衡.

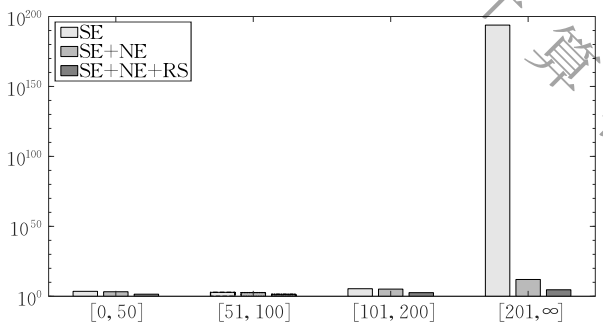


图 11 策略组合个数

图 12 展示了四类合同中 SE、SE+NE 和 SE+NE+RS 这三种方法在计算纳什均衡耗时的对比(对数坐标). 可以看到，叠加了冗余策略化简之后，计算纳什均衡的效率同样大幅提升. 对于一些非常复杂的合同，SE+NE 方法需要超过一小时来求纳什均衡，而 SE+NE+RS 方法只需要几秒. 我们在该实验中设置的最长时间为 1h，因此对于 SE 方法，其实际需要的时间远超过图中的数据.

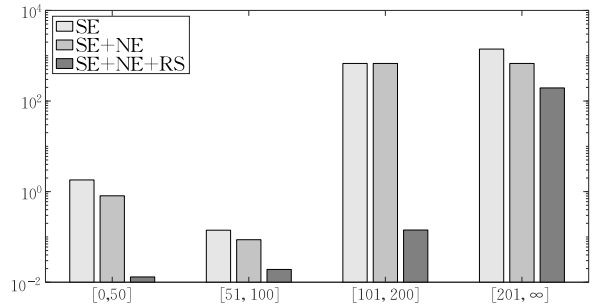


图 12 计算纳什均衡所需时间(单位：s)

以上结果表明，我们所提出的化简方法可大幅度缩减计算纳什均衡所需的时间.

7 总结和展望

区块链使用智能合约描述参与各方需要共同遵守的规则. 一旦合约中存在漏洞问题，往往就会带来巨大的损失，因此智能合约缺陷检测成为了受到广泛关注的研究问题. 然而，现在区块链的智能合约并没有统一的编程语言和运行模型，一般需要由不懂计算机编程语言的管理人员负责智能合约的创建和检测. 针对这个问题，我们之前的工作提出了一种编程语言无关、易于理解的智能合约模型. 本文中，在这个合约模型的基础上，我们形式化定义了智能合约的博弈问题，然后提出基于纳什均衡的漏洞检测方法；并提出了一系列的策略形式化简算法，实现高效的缺陷检测. 此外，我们实现了一个图形化的工具，可以对智能合约进行建模、分析和部署. 最后，通过实验对方法的可用性和效率进行了验证.

未来我们将进一步完善建模方法，支持更复杂的语义. 也会研究基于机器学习来求解纳什均衡的方法，以支持更为复杂的合同.

参 考 文 献

[1] Szabo N. Smart contracts: Building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought, 1996, 16:

[2] Luu L, Chu D-H, Olickel H, et al. Making smart contracts smarter//Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Vienna, Austria, 2016: 254-269

[3] Ministry of Industry and Information Technology. White Paper on China's Blockchain Technology and Application Development, October 2016(in Chinese)
(工信部.《中国区块链技术和运用发展白皮书》, 2016 年 10 月)

[4] Wang Pu-Wei, Yang Hang-Tian, Meng Ji, et al. Formal definition for classical smart contracts and reference implementation. Journal of Software, 2019, 30(9): 2608-2619(in Chinese)
(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. 软件学报, 2019, 30(9): 2608-2619)

[5] Desai N, Chopra A K, Singh M P. Representing and reasoning about commitments in business processes//Proceedings of the 22nd AAAI Conference on Artificial Intelligence. Vancouver, Canada, 2007: 1328-1333

《计算机学报》

[6] Osborne M J, Rubinstein A. A Course in Game Theory. USA: MIT Press, 1994

[7] Yang Z, Lei H. Formal process virtual machine for smart contracts verification. International Journal of Performability Engineering, 2018, 14(8): 1726-1734

[8] Bigi G, Bracciali A, Meacci G, et al. Validation of decentralized smart contracts through game theory and formal methods// Bodei C, Ferrari G, Priami C, eds. Programming Languages with Applications to Biology and Security, Lecture Notes in Computer Science 9465. Cham: Springer, 2015: 142-161

[9] Mavridou A, Laszka A. Designing secure Ethereum smart contracts: A finite state machine based approach//Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC 2018). Nieuwpoort, Curacao, 2018: 523-540

[10] Gao X, Singh M P. Extracting normative relationships from business contracts//Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). Paris, France, 2014: 101-108

[11] Singh M P, Chopra A K. Clouseau: Generating communication protocols from commitments//Proceedings of the 34th Conference on Artificial Intelligence (AAAI). New York, USA, 2020

[12] Desai N, Narendra N C, Singh M P. Checking correctness of business contracts via commitments//Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008). Estoril, Portugal, 2008: 787-794

[13] Chopra A K, Oren N, Singh M P, et al. Analyzing contract robustness through a model of commitments//Proceedings of the 11th AAMAS International Workshop on Agent Oriented Software Engineering (AOSE). Berlin, Germany, 2011: 17-36

[14] Adler J, Berryhill R, Veneris A, et al. Astraea: A decentralized blockchain oracle//Proceedings of the 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). Halifax, Canada, 2018: 1145-1152

[15] Shapley L S. A note on the Lemke-Howson algorithm// Balinski M L ed. Pivoting and Extension: Mathematical Programming Studies, vol 1. Berlin, Germany: Springer, 2009

[16] Sureka A, Wurman P R. Using Tabu best-response search to find pure strategy Nash equilibria in normal form games// Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005). Utrecht, The Netherlands, 2005: 1023-1029

CHEN Jin-Chuan, Ph. D., associate professor. His main research interests include distributed data management and blockchain.

XIA Hua-Hui, M. S. candidate. His main research interests include blockchain and smart contracts.

WANG Pu-Wei, Ph. D., associate professor. His main research interests include service computing and blockchain.

MA Na-Bo, M. S. candidate. His main research interests include distributed computing and blockchain.

WANG Qi, M. S. candidate. Her main research interest is blockchain.

LI Hao-Ran, M. S. candidate. His main research interests include service computing and blockchain.

DU Xiao-Yong, Ph. D., professor. His main research interests include distributed data management technology.

《计算机学报》

Background

This work focuses on the problem of modelling and analyzing smart contracts. Recent years, smart contracts and blockchain become a hot research topic in distributed computing, information security, database, and service computing etc. Most works regard smart contract as computer programs running on blockchain systems. Hence these smart contracts have nothing to do with "contracts". Our work tries to bridge the gap between business contracts and computer codes.

Moreover, in the multi-agent community there have been many works on modelling business contracts, or inter-organization processes. Most of these works are based on the commitment model. They try to describe business contracts with formal models, analyze the correctness of the proposed models, and discuss how to utilize the models to simulate the processes of executing business contracts.

Different from these works, the emphasis of this paper is about how to analyze the loopholes of contracts. For this purpose, we propose a method to compute the Nash equilibriums of formalized contracts and a series of techniques to speed up this process. To the best of our knowledge, this work is the first one that illustrates a practical process to detect loopholes of business contracts based on Game theory. This is the newest result of our research. Previous works on this topic have been published in ICSE, TPDS, Journal of Software (in Chinese) etc.

This work is supported by the National Natural Science Foundation of China (No. U1911203), and the joint projects from Guizhou College of Finance and Economics and Chinese Academy of International Trade and Economics Cooperation (No. 2017SWBZD08).