基于角色的区块链拍卖合约抽象建模及其时间安全性与公平性验证

王昌晶[1]，欧阳俊媛[2]，张取发[1]，左正康[1]，程着[3]，卢家兴[1]
（1.江西师范大学计算机信息工程学院，江西 南昌 330022；2.东华理工大学软件学院，江西 南昌 330013；
3.江西师范大学国家网络化支撑软件国际合作基地，江西 南昌 330022）

摘　要：为提升拍卖合约时间安全性验证效率及验证公平性，提出基于角色的拍卖合约抽象建模及其验证方法。首先，对合约源代码进行基于账户角色的抽象建模，转换为时间自动机网络模型，并对时间安全性进行形式化描述，用UPPAAL工具验证。其次，提取合约源代码机制，建立智能合约机制模型，同样转换为时间自动机网络模型，并对4种公平性进行形式化描述，再用UPPAAL验证。最后，通过2个经典案例证明了所提方法的可行性和有效性。

关键词：拍卖合约；时间安全性；公平性；时间自动机；UPPAAL
中图分类号：TP311
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2024074

Role-based abstract modeling of blockchain auction contracts and verification of temporal security and fairness

WANG Changjing[1], OUYANG Junyuan[2], ZHANG Qufa[1], ZUO Zhengkang[1], CHENG Zhuo[3], LU Jiaxing[1]
1. School of Computer and Information Engineering, Jiangxi Normal University, Nanchang 330022, China
2. School of Software, East China University of Technology, Nanchang 330013, China
3. National-level International Science and Technology Cooperation Base of Networked Supporting Software, Jiangxi Normal University, Nanchang 330022, China

Abstract: To enhance the efficiency of time security verification and fairness verification of auction contracts, an abstract modeling and verification method for role-based auction contracts was proposed. Firstly, the source code of the contract was abstractly modeled based on account roles and converted into a timed automaton network model. Formal descriptions of time security were provided and verified using the UPPAAL tool. Secondly, the mechanisms in the source code of the contract were extracted to establish a smart contract mechanism model, which was also converted into a timed automaton network model. Formal descriptions of four types of fairness were provided and verified using UPPAAL. Finally, the feasibility and effectiveness of the proposed method were demonstrated through two classic cases.

Keywords: auction contracts, temporal security, fairness, time automata, UPPAAL

收稿日期：2023–12–29；**修回日期：**2024–03–14
通信作者：卢家兴，ljxluois@jxnu.edu.cn
基金项目：国家自然科学基金资助项目（No.62462037, No.62462036）；江西省主要学科学术与技术带头人培养基金资助项目（No.20232BCJ22013）；江西省自然科学基金资助项目（No.20242BAB26017）；江西省教育厅科技基金资助项目（No.GJJ2200303, No.GJJ210340）
Foundation Items: The National Nature Science Foundation of China (No.62462037, No.62462036), Project for Academic and Technical Leader in Major Disciplines in Jiangxi Province (No.20232BCJ22013), Jiangxi Provincial Natural Science Foundation (No.20232BAB202010), Science and Technology Project of Education Department of Jiangxi Province (No.GJJ2200303, No.GJJ210340)

0 引言

区块链是一种安全共享的去中心化的数据账本，智能合约是部署在区块链上的计算机程序[1]，当满足必要条件时，就会自动执行。目前，基于以太坊的智能合约是世界上主流的智能合约。以太坊通过一套图灵完备的脚本语言来建立智能合约应用，主要过程是用Solidity[2]语言对智能合约进行编写，然后编译成字节码并在以太坊虚拟机（EVM, Ethereum virtual machine）中执行。智能合约包含了区块链的众多突出特性，如去中心化、不可篡改等，使得区块链与智能合约技术在许多领域都得到了广泛的应用，尤其是在金融、物联网等领域。

智能合约在运行时往往会伴随着大量数字资产的存储和转移，一旦合约代码中存在漏洞，攻击者就能通过对合约发起攻击而获取较大的经济利益，因此，智能合约的安全性至关重要。近几年来，智能合约时间安全方面出现的一些问题对合约参与方造成了巨大的损失[3]。

同时，智能合约的公平性也不可忽视[4]。由于智能合约的运行结果通常会涉及资产、权利等重要利益关系的转移，往往存在某些参与人为了获取更多利益而导致合约对其他参与人不公平的现象。例如，智能合约可能被宣传为“社群游戏”，承诺任何投资都能获得20%的回报，但最终证明是“庞氏骗局”，这是因为合约创造者故意忽略了合约最终可能会放缓且永远不会有回报的可能性[5]。本文将重点讨论Liu等[6]提出的基于机制模型的4种公平性属性，包括真实性、最优性、高效性和无共谋性，其具体定义见第5.2节。

智能合约拍卖协议是一种在区块链上运行的程序，用于自动化和规范化数字资产拍卖，其中智能合约管理交易流程和规则。智能合约拍卖协议增强了拍卖的透明性、去中心化和自动化，减少了信任和欺诈问题，提供了更安全和公平的数字资产交易方式。许多拍卖合约声称是安全和公平的，但仍然存在参与人之间相互勾结或与拍卖商勾结，以牺牲他人的利益牟利的情况[7]。

综上所述，对智能合约时间安全性和公平性问题的研究缺一不可，否则不仅会造成合约参与人的经济损失，还会使合约参与人对合约乃至区块链本身产生信任危机，阻碍区块链智能合约未来的发展。拍卖合约用于分配各种形式的资源，其作为一类典型的涉及参与者资产转移的区块链协议，如何保证其时间安全性和公平性，是目前研究的一大热点。

形式化验证方法[8]是验证智能合约时间相关性质的根本途径，常见的形式化验证方法包括定理证明[9]和模型检测[10]。目前，使用模型检测对智能合约进行建模与属性验证方面，已有一些研究。时间安全性方面，赵颖琪等[11]针对智能合约的时间性质可能引起的安全性问题进行了系统的研究，提炼了智能合约的5种时间约束模式，使用UPPAAL对智能合约带时间约束的性质进行验证与研究，但该工作在对智能合约进行建模的阶段并没有考虑模型空间复杂度的简化，缺少对提升模型验证效率方面的分析研究。公平性方面，Liu等[6]设计了将合约源代码通过定义的中间表示转换为数学机制模型的方法，然后对模型进行符号路径分析与公平性验证，但使用中间表示会对符号执行的结果产生影响，且符号执行中的约束求解精度较低、效率不高、验证结果可视性也不强。

本文采用模型检测技术进行属性验证，提出了一种基于账户角色的区块链智能合约抽象建模方法，通过抽象出智能合约中的账户角色作为时间自动机的模板，将合约在行为层次上转换成时间自动机网络模型，从而提高了合约的抽象程度和验证效率。

本文的主要贡献如下。

1)提出了一种基于账户角色的拍卖合约抽象建模方法。通过引入合约账户角色进行抽象建模，使得模型的时间自动机模板数与模型间的通信量减少，从而减少了系统的变迁数量，大大缩减了模型的状态空间，有效地提高了模型检测方法的验证效率。

2)提出了从智能合约Solidity代码到时间自动机网络模型的转换规则与算法，结合对与时间相关的安全性进行形式化描述，验证拍卖合约的时间安全性。

3)提出了从智能合约机制模型到时间自动机网络模型的转换规则与算法，结合对机制模型中的4种公平性进行形式化描述，验证拍卖合约的真实性、高效性、最优性和无共谋性这4种公平属性。

4)通过2个经典的拍卖合约案例，证明了提出方法的可行性和有效性。

1　相关工作

目前，关于区块链智能合约安全性、时间安全性和公平性的建模与验证已经引起了广泛关注，已有研究者对此展开了工作，相关工作阐述如下。

智能合约安全性建模与验证。主要包括静态分析方法和形式化方法。其中，基于静态分析的方法通常是利用已总结的安全漏洞模式进行验证，常用工具包括MythX[12]、Oyente[13]、Manticore[14]、Securify[15]等，但难以验证合约的可靠性问题，形式化方法可以很好地进行补充[11]。

形式化方法大致分为定理证明和模型检测2类，基于定理证明的方法在验证过程中需要人工干预，且对使用者的专业知识要求较高，而基于模型检测的方法能够全自动地执行验证过程。本文方法基于模型检测技术，下面主要介绍利用模型检测工具验证智能合约安全性的相关工作。文献[16]提出了一种从Solidity程序生成Promela模型的转换函数，使用线性时序逻辑描述智能合约的性质，然后使用模型检测工具SPIN进行验证。文献[17]提出了一种由不同实体开发和控制的交互式智能合约组成的系统验证方法，使用分支时序逻辑描述系统的属性，并使用NuSMV验证其属性。文献[18]提出了一种基于着色Petri网的区块链智能合约形式化验证方法，利用CPN（colored petri net）工具中基于分支定时逻辑的模型检测技术检测智能合约中的潜在漏洞。这些研究都只是验证了智能合约的安全性，并没有验证其带时间约束的安全性。

智能合约时间安全性建模与验证。目前，对智能合约时间安全性的验证工作还不多，但已有研究者针对智能合约的时序问题进行了研究。文献[19]针对不带循环、递归和动态内存管理的简单Solidity语言子集提出了一种符号模型检测技术和一种形式化规范方法，可以检测交易执行不当造成损失的问题，使用SMT求解器作为后端工具来查找给定性质的反例。文献[20]利用现有的命题投影时序逻辑（PPTL, propositional projection temporal logic）的模型检测算法对区块链中是否具有真正的并发性进行了建模和验证。文献[21]利用模型检测工具UPPAAL对智能合约进行建模与验证，手动地对投票合约建模，验证了一些智能合约带时间约束的性质。文献[11]提炼了智能合约的5种时间约束模式，提出了一种将智能合约转换为时间自动机网络模型的方法，使用UPPAAL对智能合约带时间约束的性质进行验证与研究。这些研究都对智能合约建模并验证了合约带时间约束的性质，然而其建模方法抽象程度不高，验证效率较低。

智能合约公平性建模与验证。与智能合约安全性引起的关注相比，其公平性尚未得到应有的重视，相关研究工作较少。文献[6]提出了一个名为FairCon的框架来验证智能合约的公平性，定义了一种中间表示将合约源代码转换为数学机制模型，然后对其进行符号路径分析，但中间表示的抽象程度会对符号执行产生影响[22]，且符号执行中的约束求解精度较低、效率不高。文献[23]中虽然也涉及了智能合约公平性的一些相关内容，但其不能对涉及数学公式的公平属性进行描述验证。

本文提出了一种基于账户角色的拍卖合约抽象建模及其时间安全性与公平性验证方法，不同于文献[11]，本文采用基于账户角色的抽象建模方法提高了时间安全性验证效率；不同于文献[6]，本文方法不仅提高了公平性验证效率，而且更为简单直观。

2　准备知识

本文的工作以拍卖合约Solidity代码及其机制模型为验证对象，以实时模型检测工具UPPAAL为后端验证工具，本节对相关内容做简要介绍。

2.1　智能合约的结构及其时间安全性

Solidity是一种面向对象的高级语言，用于实现智能合约，其语法与JavaScript类似。在以太坊中，使用Solidity语言编写的智能合约需要通过编译器编译后才能在以太坊虚拟机EVM[24]上运行。

智能合约的主体内容从contract开始，contract声明中主要包括变量、结构体、事件以及函数。变量可分为全局变量和局部变量。结构体由关键字struct声明，类似于高级语言中的结构体。事件是以太坊虚拟机日志基础设施提供的一个便利接口。

智能合约是一套以数字形式定义的承诺[25]。承诺指的是合约参与方同意的权利和义务，是各参与方需共同遵守的协议。承诺往往具有时效性，

即时间约束，因此，大多数的智能合约都与时间息息相关。赵颖琪等[11]已经总结出了智能合约的5种时间约束模式，说明了智能合约时间性质的重要性。

2.2　机制模型

机制设计用于设计经济机制或激励机制，以帮助实现参与指定活动的不同利益相关者的目标。这些目标主要与参与者的回报和其在活动中的回报描述的结果有关。本文用一个被称为机制的数学对象来建模智能合约代码背后的逻辑[26]。

在一个机制模型中，有一个有限数量的个体，用$N=\{1,2,\cdots,n\}$表示。每个个体i拥有一条私人信息，用一种类型表示，$\theta_i \in \Theta_i$。令所有个体的类型为$\theta=(\theta_1,\cdots,\theta_n)$，空间为$\Theta=\times i\Theta_i$。个体报告的策略描述$\hat{\theta} \in \Theta$很可能是不诚实的。根据每个个体的报告，机制模型决定一个的结果，由分配函数$d:\Theta| \to O$和传递函数$t:\Theta| \to R^n$共同指定，其中$O=\{o_i \in \{0,1\}^n | \sum_i o_i = 1\}$是可能结果的集合。

个体对不同的结果有不同的偏好程度，使用估值函数$v_i:O \times \Theta_i| \to R$表示。因此，$v_i(o,\theta_i)$表示类型为$\theta_i$的个体$i$从$o \in O$的结果中获得的利益，$v_i(o,\theta_i) > v_i(o',\theta_i)$表示$i$更喜欢结果$o$而不是$o'$。在策略描述$\hat{\theta}$下，个体$i$的效用是通过从某一结果的估值中减去将要支付的费用来计算的，即$u_i(\hat{\theta})=v_i(\hat{\theta},\theta_i)-t_i(\hat{\theta})$。

为了后续方便对4种公平性属性进行形式化定义及描述，首先引入主导策略这一重要概念。使用$\hat{\theta}_{-i}$表示除i以外的所有个体的策略描述，即$(\hat{\theta}_1,\cdots,\hat{\theta}_{i-1},\hat{\theta}_{i+1},\cdots,\hat{\theta}_n)$，使用$\hat{\theta}_{-ij}$来表示除个体$i$和个体$j$以外的所有其他个体的策略描述，即$(\hat{\theta}_1,\cdots,\hat{\theta}_{i-1},\hat{\theta}_{i+1},\cdots,\hat{\theta}_{j-1},\hat{\theta}_{j+1},\cdots,\hat{\theta}_n)$。因此，$(\hat{\theta}',\hat{\theta}_{-i})$用于表示仅在$\hat{\theta}_i$上与$\hat{\theta}$不同的策略分布。如果对于$\forall\hat{\theta}_{-i},\hat{\theta}_i' \in \Theta_i$，$u_i\left(\hat{\theta}_i,\hat{\theta}_{-i}\right) \geqslant u_i\left(\hat{\theta}_i',\hat{\theta}_{-i}\right)$成立，则$\hat{\theta}_i \in \Theta_i$是$i$的主导策略。当等号成立时，该策略是弱主导策略。

2.3　模型检测工具UPPAAL

模型检测是一种常见的形式化验证方法，其基本思想是用状态迁移系统M表示系统的行为，用模态/时序逻辑公式F描述系统的性质。这样就把“系统是否具有所期望的性质”转化为数学问题“状态迁移系统M是不是公式F的一个模型”。模型检测是一种基于状态空间搜索的自动化验证技术，主要包括并发系统模型检测、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。模型检测验证框架如图1所示。

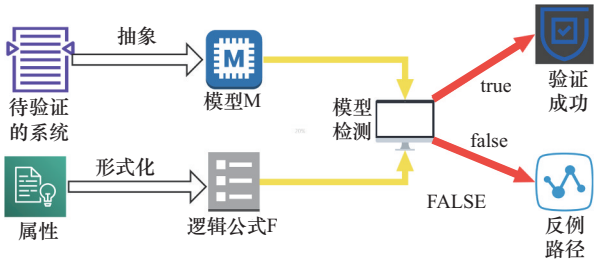


图1　模型检测验证框架

UPPAAL[27]是一种基于时间自动机的模型检测工具，可以对实时系统进行建模和自动验证。此外，它不仅可以用于算法分析和协议验证，还具有3个特点：一是基于时间自动机模型，便于对智能合约时间安全性进行建模与验证；二是UPPAAL能够进行可视化的建模与分析；三是UPPAAL是一个开源工具，不仅有广泛的用户社区和支持，而且具有可扩展性，例如可以扩展到公平性的建模与验证。

UPPAAL采用一组带有整型时钟变量的时间自动机对实时系统的行为进行建模，用CTL（computation tree logic）公式来描述系统的性质。UPPAAL根据时间自动机模型模拟实时系统的所有可能的状态序列，如果这些状态序列都满足某个性质，则系统满足该性质。下面给出时间自动机相关的语法及语义。

定义1　时间自动机定义。时间自动机（TA, time automata）可以定义为一个六元组(L,l_0,Σ,C,I,E)，其中，L表示位置的有穷集；l_0表示初始位置；$\Sigma=\{\alpha?|\alpha \in \text{chan}\} \cup \{\alpha!|\alpha \in \text{chan}\} \cup \{a\}$表示迁移的动作集，chan是所有信道的集合，$a$是引起系统状态变化的内部动作；$C$是时钟的集合，$B(C)$是定义在$C$上的时钟约束的集合；$I$表示位置$L$中的每一个位置指定$B(C)$中的某一个时钟约束作为该位置的不变式；$E \subseteq L \times \Sigma \times B(C) \times 2^C \times L$是有向边的集合。

定义2　时间自动机语义。一个TA的语义模型是一个时间迁移系统$T=<S,s_0,\to>$，其中，

$S \subseteq L \times Z^C$表示所有状态的集合，Z^C表示所有时钟赋值的集合；$s_0 = (l_0, u_0)$表示初始状态；$\rightarrow \subseteq S \times \{ Z_{\geqslant 0} \cup \Sigma \} \times S$表示迁移关系。

定义 3　时间自动机网络语义。一个由多个时间自动机组成的时间自动机网络可以表示为$A_i = (L_i, l_i^0, \Sigma, C, I_i, E_i)$，设$\overline{\boldsymbol{l}}_0 = (\boldsymbol{l}_1^0, \boldsymbol{L}, \boldsymbol{l}_n^0)$为时间自动机网络的初始位置向量，$I(\overline{l}) = \wedge_i I_i(l_i)$为组合位置的公共不变量，该时间自动机网络的语义模型是一个迁移系统$T =< S, s_0, \rightarrow >$，其中，$S \subseteq (L_1 \times \cdots \times L_n) \times Z^C$是状态集；$s_0 = (\overline{l}_0, u_0)$是初始状态集；$\rightarrow \subseteq S \times S$表示迁移关系。

3　智能合约账户角色抽象与时间自动机网络的转换

基于函数建模的方法是将合约的 external 函数转换为时间自动机模板[11]，其建立的时间自动机网络由环境时间自动机、进程时间自动机及系统声明构成，环境时间自动机作用是随机与其他进程时间自动机进行通信，进程时间自动机描述的是函数的功能。

第 3.1 节提出了一种基于账户角色的合约抽象建模方法。不同于基于函数建模的方法，使用模型检测中的抽象技术，增大建模的抽象粒度，抽象掉无关细节，从而缩小了整个系统状态空间的大小，提高模型的验证效率[28]。进一步地，第 3.2 节提出了一种将智能合约代码转换为时间自动机网络的方法。

3.1　智能合约账户角色抽象

本文根据智能合约账户的不同行为抽象出不同的账户角色，将每一个账户角色都转换成一个对应的时间自动机模板，最后将所有的时间自动机组合成一个完整的时间自动机网络。

基于账户角色建模方法是将智能合约所有的账户抽象为对应的账户角色，例如，对于一个购物的智能合约，根据不同的行为可以分为 3 个不同的角色：合约、买家和卖家。智能合约账户可以分为2种：外部账户和合约账户[29]。

外部账户是由用户的一对公钥和私钥控制，账户地址是由用户的公钥加密后生成，该账户存在账户余额并且能够触发交易。合约账户由智能合约代码控制，合约账户的地址在智能合约部署成功的时候自动生成，该账户也存在余额并且能触发交易。一个成功部署的智能合约有且只有一个合约地址，因此一个智能合约有且只有一个合约账户，且合约账户对应着一个合约账户角色；外部账户可以有多个，分别对应着多个参与交易的用户，根据用户在智能合约系统中行为的不同，外部账户又可以进行分类，对应多个账户角色，Solidity 智能合约的账户角色如图 2 所示。

合约账户是智能合约中特殊的角色，与其他的角色进行交互。如图2所示，合约账户对应一个合约账户角色，转换为一个时间自动机。外部账户根据用户不同的行为，抽象出不同的账户角色，行为相同的账户都隶属于同一个账户角色，每一个账户都只能抽象成一个账户角色。所有的账户角色转换为对应的时间自动机，每个时间自动机描述的是该角色的行为，具体的转换规则及算法在下节给出。最终所有的时间自动机将会组合为一个完整的时间

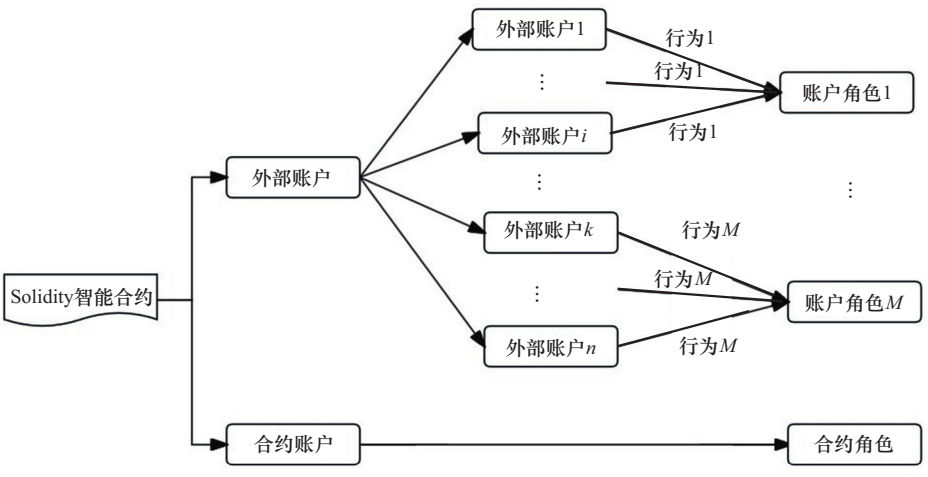


图2　Solidity 智能合约的账户角色

自动机网络。

3.2 智能合约代码转换规则与算法

本节设计了智能合约转换为时间自动机网络的算法，并阐述了具体的转换细节。

算法1描述的是将整个智能合约转换为时间自动机网络的过程。该算法输入的是Solidity文件，输出的是时间自动机网络模型。首先，从Solidity文件中抽象出合约的全局变量、合约账户以及外部账户。合约的全局变量转换为UPPAAL中的系统声明，Solidity主要的变量类型到UPPAAL的转换如表1所示。合约账户对应着合约账户角色，直接转换为一个时间自动机模板，具体的转换过程由算法2给出。外部账户根据用户的行为不同抽象出不同的账户角色，每一个账户角色都会生成一个对应的时间自动机模板，所有的账户角色会形成一个时间自动机列表，列表的具体生成过程在算法3中详细说明。最终，系统声明、合约时间自动机和外部时间自动机列表将会组合成一个完整的时间自动机网络。

算法1 TA_Generate1

输入 智能合约源文件Solidity_Source

输出 时间自动机网络模型TA_Model

1) abstract(solidity.source)→(gloal_variable, contract_account, external_acount_list);
2) global_variable→system_declarations;
//全局变量转换为系统声明
3) Template_Contract_Generate(function_list);
//调用算法2，生成合约时间自动机模板
4) Template_External_Generate(function_lists, external_role_list);
//调用算法3，生成外部账户角色时间自动机模板
5) Template_integration (system_declarations, Template_contract, Template_external_list);
//组合所有的时间自动机模板
6) return TA_Modle;

Solidity中integer类型的变量转换为UPPAAL中int类型变量，其中的运算符在UPPAAL中同样适用。bool类型对应着UPPAAL中的bool类型，bool类型的值为true和false，在UPPAAL中bool类型的取值对应着int类型的0或1。Solidity中数组和结构体可以分别转换为UPPAAL中的数组和结构体。对于Solidity中特殊的映射类型，在UPPAAL中可以用数组近似替代。bytes32是Solidity中特有的类型，是一个32字节的字符串，用于在以太坊区块链上存储数据，在UPPAAL中可以用int类型简单替代。除了智能合约中定义出来的变量外，还有一些变量在实际情况中存在，但没有在智能合约中定义，例如外部账户和合约账户的余额，它们的值会随着某些状态的变化而改变，因此要在UPPAAL中进行声明。

表1 Solidity主要的变量类型到UPPAAL的转换

Solidity	UPPAAL
integer	int
bool	bool
Type *a*[]	Type *a*[]
struct{}	struct{}
mapping	Type map[]
bytes32	int

算法2描述了合约账户生成合约时间自动机模板的过程，输入的是智能合约中的external函数列表（包括构造函数），输出的是一个时间自动机模板。state_list表示的是账户角色行为的状态列表，该状态列表可以用来表示由账户角色生成的时间自动机。首先，对含初始值的变量进行初始化，此时状态列表增加一个状态。开始遍历函数列表，如果是构造函数，则执行该函数，状态列表加1，然后将其从函数列表中移除，并进行下一次循环；如果不是构造函数，则下一个状态将会与该函数进行同步通信，状态列表加1，接着，将该函数从函数列表中删除，直到遍历完合约中所有的external函数。

算法2 Template_Contract_Generate

输入 智能合约函数列表function_list

输出 合约时间自动机Template_contract

1) initial(variable);
2) state_list.add(state);
3) while(function_list)
4) if(contructor) //执行构造函数
5) state=contructor();
6) state_list.add(state);
7) function_list.delete(constructor);
8) else
9) state_list.add(synchronous);
//与当前的函数进行同步通信

```
10) function_list.delete(current_function);
    //将遍历过的函数移除
11) Template_contract = state_list;
    //状态列表对应着时间自动机
12) return Template_contract;
```

算法 3 描述了外部账户生成时间自动机模板列表的过程。输入的是所有 external 函数列表（不包括构造函数）和账户角色列表，外部账户角色用来区分用户的不同行为，输出的是时间自动机模板列表。每种角色对应着一个时间自动机模板，所有时间自动机的初始状态是 idle。接下来是两层循环，外循环是对账户角色列表进行遍历，内循环是对函数列表进行遍历。内循环中，先判断当前函数的调用是不是该账户角色的行为，如果不是，则将该函数从中移除，并且进行下一次循环；如果该函数的调用是该账户角色的行为，则遍历函数的所有语句，Solidity 语句的转换规则如表 2 所示。当某个账户角色遍历完所有的函数列表后，生成了该账户角色对应的时间自动机模板。当遍历完所有的账户角色，最后生成时间自动机列表。

算法 3　Template_External_Generate

输入　外部函数列表和账户角色列表(function_lists, external_role_list)

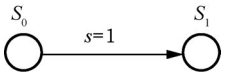
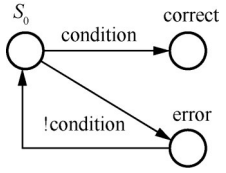
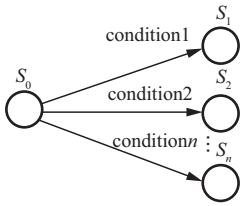
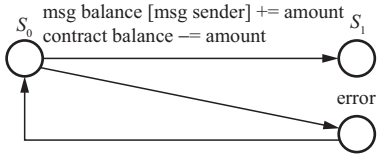
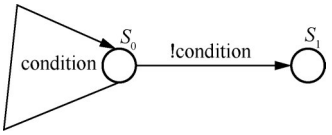
输出　账户角色对应的时间自动机模板列表 Template_external_list

```
1) state_list[role] = idle;
2) while(external_role_list) //账户角色列表
3) function_list = function_lists;
   //每个账户角色都要遍历所有的函数列表
4)  while(function_list)
5)   if(!current_role_behavior)
6)      function_list.delete(current_function);
        //当前函数不是该账户角色的行为
7)      continue;
8)   else//当前函数是该角色的行为
9)   for statement s of current_function
       //遍历函数
10)  if s is assignment statement //赋值语句
11)    state_list[current_role].add(new_state);
12)  if s is require statement //require 语句
13)    if(condition)
14)      state_list[current_role].add(correct);
15)    else
16)      state_list[current_role].add(error);
17)      state_list[current_role].next=idle;
18)  if s is condition statement //条件语句
19)    if(condition1)
20)      state_list[current_role].add(state1);
21)    if(condition2)
22)      state_list[current_role].add(state2);
23)    ……
24)    if(conditionN)
25)      state_list[current_role].add(stateN);
26)  if s is transfer statement //transfer 语句
27)    if(success)
28)      state_list[current_role].add(success);
29)    else
30)      state_list[current_role].add(error);
31)      state_list[current_role].next=idle;
32)  if s is loop statement //循环语句
33)    first_state=state_list[current_role].state;
34)    if(condition)
35)      state_list[current_role].next=first_state;
36)    else
37)      state_list[current_role].add
         (second_state);
38)  end for
39) end while
40) external_role_list.delete(current_role);
41) end while
42) Template_external_list = state_list[role];
    //状态列表数组对应账户角色生成的时间自动机
43) return Template_external_list;
```

表 2 中赋值语句是最常见的语句，它是对变量的值做了一定的改变，进而改变系统的状态。在 UPPAAL 中使用 2 个位置表示该变量变化前后系统的状态，然后在有向边上对变量进行赋值。Require 语句是 Solidity 中特有的一种语句，用来判断条件是否满足，如果条件满足则继续运行，如果条件不满足则回滚到该语句开始的状态；在 UPPAAL 中使用一个 correct 位置表示满足条件后的状态，用 error 位置表示不满足条件后的状态并且该状态会直接回到最初的状态。条件语句主要是判定变量满足

什么条件然后执行对应的语句；在UPPAAL中，初始位置会根据有向边上的guard条件指向对应的位置。transfer语句是Solidity中的转账语句；在UPPAAL中，如果转账成功则到达状态成功的位置，并更新对应的变量值，如果转账失败会到达error位置并回滚到最初的状态。循环语句在Solidity中出现的比较少；在UPPAAL中，循环从初始位置s_0开始，满足循环条件，则继续回到s_0，直到不满足循环条件，则到达一个新的位置s_1。

表2　　Solidity语句的转换规则

Solidity语句	UPPAAL时间自动机
赋值语句	
require语句	
条件语句	
转账语句	
循环语句	

4　机制模型提取与时间自动机网络的转换

仅仅靠对智能合约的代码进行形式化验证，只能发现合约代码中潜在的安全性漏洞，在本文的工作中，还关注了由合约代码的逻辑设计所带来的公平性问题。公平性通常取决于参与人的理解和偏好，对某人公平的合约可能对其他人不公平。为了解决这一问题，本文将在第4.1节根据第2.2节引入的机制模型，从智能合约中提取其机制，建立合约机制模型；进而在第4.2节设计合约机制模型转换规则与算法，将其转换为时间自动机网络，便于后续对公平性进行验证。

4.1　机制模型提取

公平性在很大程度上是由个人偏好决定的主观概念，当在拍卖合约中只考虑一个参与者时，没有讨论意义。因此，本文将拍卖合约视为一个机制，接受来自多个参与者输入的私人信息，并根据一些预定义的规则来输出结果。尽管不同类型的拍卖合约规则大不相同，但都可以使用机制模型中的函数来处理。在结束时，每个参与者都会根据结果获得特定的效用，以此来对公平性属性进行刻画。在第5.2节，本文将讨论4种公平性属性，即真实性、最优性、高效性和无共谋性。

根据第2.2节中对机制理论模型的定义，可以从拍卖合约Solidity源代码中提取其机制。以一价拍卖合约为例，在一价拍卖智能合约中，每个参与人的相关信息可以被视为一个三元组—individual$_i$(id$_i$,bid$_i$,value$_i$)，分别代表参与人i的唯一标识符、提交的竞价值和其对拍卖物品的估值。@individual(msg.sender,msg.value,VALUE)表示参与人的三元组相关信息分别储存在变量msg.sender、msg.value和VALUE中。

根据每个参与者提交的竞价值bid，触发合约代码的bid()函数，机制模型通过分配函数allocation()和传递函数price()得出一个结果，@allocation(highestBidder)和@price(highestBid)表示该一价拍卖的分配结果和成交价格分别存储在变量highestBidder和highestBid中。另外，如果是二价拍卖，这2个函数输出的就分别是最高出价者和第二高的出价。当然，还有一些其他规则的情况，都可以通过具体的合约代码进行分析提取得到。

根据上述过程得到的结果，每个参与者都会获得特定的效用，即参与人从结果中获得的利益。例如，对于没有获胜的参与者来说，没有赢得拍卖，因此不需要支付任何费用，此时他们的效用为0。而对于拍卖的获胜者i，则需要根据规则支付相应的费用，如果是一价拍卖，支付的金额则是highestbid，此时参与人i的效用等于估值减去将要支付的费用，公式为u_i=VALUE$_i$-highestbid。每个参与者的效用，其计算需要依据该参与者在拍卖中的获胜结果，因此，通过给定一个outcome数组标记每个参与者的获胜情况，如果参与者i获胜，那么outcome$_i$=1，否则，outcome$_i$=0。根据outcome数组，可以使用一个估值函数valuation()来获取每个

参与者对结果的估值，@valuation(outcome×VALUE)表示对应个体对该结果的估值。这样，对于每个不同的个体就有了统一的效用公式，比如，u_i=valuation$_i$-highestbid 是个体 i 的效用公式。@utility(@valuation-highestBid)表示对应个体在该结果下获得的效用。

拍卖合约机制模型建立过程如图3所示，再建立 K 个参与人的拍卖合约机制模型的实例过程。

4.2　机制模型转换规则与算法

本节设计了智能合约机制模型转换为时间自动机网络的算法，并阐述了具体的转换细节。

算法4描述的是将整个智能合约机制模型转换为时间自动机网络的过程。该算法输入的是智能合约机制模型，输出的是时间自动机网络模型。首先，从合约机制模型中抽象出合约模型的全局变量、合约账户以及外部账户。机制模型中的全局变量转换为UPPAAL中的系统声明，合约账户对应着合约账户角色，直接转换为一个时间自动机模板，具体的转换过程由算法5给出。外部账户根据用户的行为不同抽象出不同的账户角色，每一个账户角色都会生成一个对应的时间自动机模板，所有的账户角色会形成一个时间自动机列表，列表的具体生成过程在算法6中详细说明。最后系统声明、合约时间自动机和外部时间自动机列表将会组合成一个完整的时间自动机网络。

算法4　TA_Generate2

输入　智能合约机制模型 Mechanism_Model

输出　时间自动机网络模型 TA_Model

1) abstract(Mechanism_Model)→(bidders, global_variable, contract_account)
2) global_variable→system_declarations //全局变量转换为系统声明
3) contract_account→Template_Contract (K-player);//调用算法5，生成合约时间自动机模板
4) bidders→Template_Bidders(BID[k][m]); //调用算法6，生成合约账户角色时间自动机模板
5) Template_integration(system_declarations, Template_Contract, Template_Bidders) //组合所有的时间自动机模板
6) return TA_Model;

算法5描述了机制模型中 K 个参与人生成合约时间自动机模板的过程。输入的是合约机制模型中的 K 个参与人，输出的是一个时间自动机模板。state_list 表示合约时间自动机模板的状态列表。首先对含初始值的变量进行初始化，此时状态列表增加一个状态，再设置 K 个参与人的范围，确认好参与人以后，下一个状态将会与 bidder 时间自动机进行同步通信，状态列表加1，直至所有的参与人都与 bidder 时间自动机通信。

算法5　Template_Contract_Generate

输入　K 个参与人 K-player

输出　合约时间自动机 Template_contract

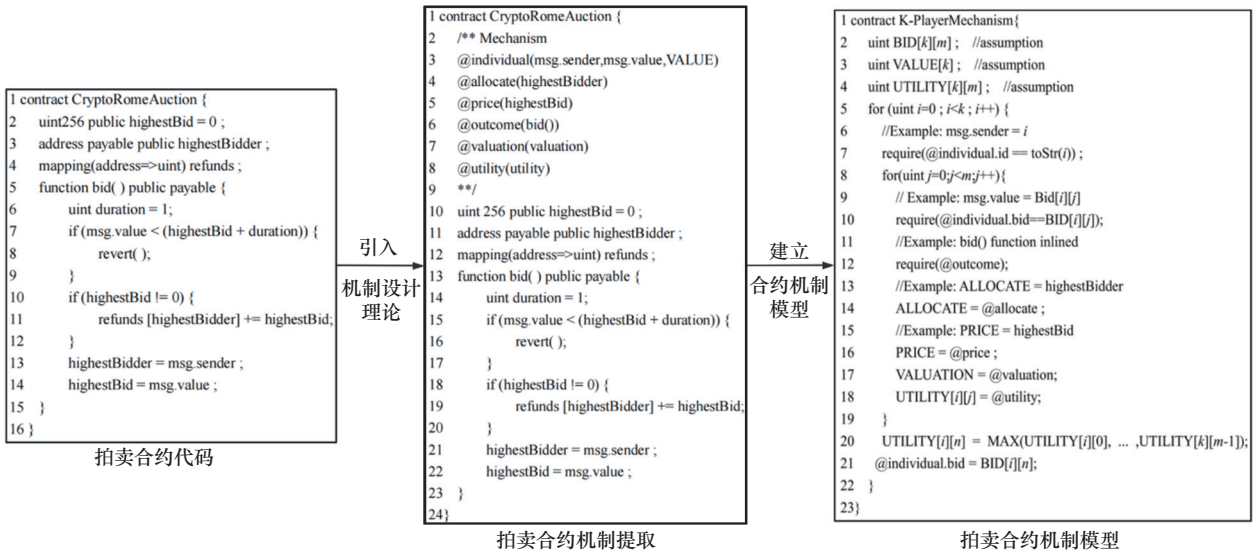


图3　拍卖合约机制模型建立过程

1) initial(variable); //初始化变量
2) state_list.add(state);
3) set_senderDomain(*K*-player);
//设置参与人的范围
4) state_list.add(sender);
5) state_list.add(synchronous);
//与 bidder 时间自动机同步通信
6) Template_contract = state_list;
//状态列表对应着时间自动机
7) return Template_contract;

算法 6 描述了外部账户出价生成时间自动机模板的过程。输入的是该账户角色所有可能的出价，输出的是时间自动机模板列表。当遍历完某个账户角色所有可能的出价后，根据每个可能的出价计算出的效用值，得出该账户角色的主导策略，更新变量，生成该账户角色对应的时间自动机模板。

算法 6 Template_Bid_Generate
输入 账户角色出价 BID[*k*][*m*]
输出 时间自动机列表 Template_Bidder
1) state_list.add(Idle);
2) state_list.add(synchronous);
//与 contract 时间自动机同步通信
3) set_valueDomain(BID[*k*][*m*]);
//设置参与人的出价
4) state_list.add(state);
5) while (*j*<*m*)
//根据第 *i* 个参与人的第 *j* 个出价进行计算
6) calculate (@individual.bid, @outcome, @allocate, @price, @valuation, @utility)
7) *j*++;
8) end while;
9) state_list.add(state);
10) getMax (utility);
//寻找效用值最大的出价
11) state_list.add(Bidded);
12) update(variable);
13) state_list.add(Success);
14) Template_Bidder=state_list;
//状态列表对应着时间自动机
15) return Template_Bidder;

将智能合约机制模型转换为 UPPAAL 时间自动机模型，需要考虑 UPPAAL 工具对机制模型语言的支持性，UPPAAL 支持的机制模型语言子集如图 4 所示。

```
type ::= address | uint | mapping
operator ::= + | - | * | / | ++ |-- | += | -= | *=| /=
logicoperator ::= || | && | > | < | >= | <= | != | ==
statement ::= assignment | condition statement | send |
              for statement | ! return | require | transfer
```

图 4 UPPAAL 支持的机制模型语言子集

表 3 给出了 6 种常见的机制模型语句转换成 UPPAAL 时间自动机的转换规则。更新语句是最常见的语句，通过改变一个或多个变量的值，进而改变系统的状态。require 语句用来判断条件是否满足，如果条件满足则继续运行，如果条件不满足则回滚到该语句开始的状态。if 语句主要是判定变量满足的条件，然后根据不同的结果执行对应的语句。循环语句在智能合约中出现得比较少，满足循环条件，则继续循环，直到不满足循环条件时则跳出循环，到达一个新的位置。

5 案例分析

近年来已有研究对拍卖合约进行分析，本节将通过简单公开拍卖合约和加密罗马拍卖合约来验证本文提出方法的可行性和有效性，这 2 个案例涵盖了拍卖合约的基本功能且相对简单、易于理解，重点是验证智能合约的时间安全性和公平性。

5.1 拍卖合约的时间安全性建模与验证

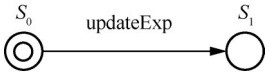
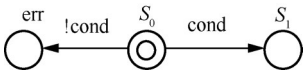
依据第 3 节基于账户角色的建模方法，本节通过一个简单公开拍卖合约的案例来进行建模与时间安全性验证。

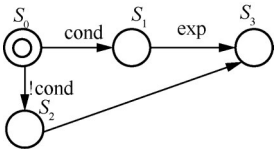
5.1.1 拍卖合约转换为时间自动机网络模型

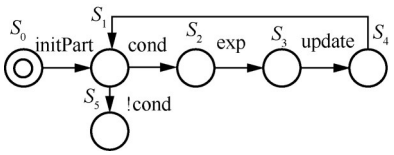
简单公开拍卖合约的内容是：定义拍卖时间期限和受益人，在规定的拍卖时间内，所有的用户都可以竞价，系统将会观测最高的竞拍价格及其竞价人。如果当前的时间大于规定的拍卖时间期限，系统将会结束拍卖，并且最高的竞拍金额将会转给受益人，竞价失败的用户可以主动调用 withdraw 获得退款。

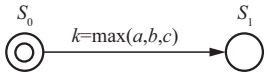
分析简单公开拍卖合约的流程，根据账户不同的行为可以抽象出 3 种账户角色：合约（合约账户）、竞价人（外部账户）、受益人（外部账户）。合约的行为包括初始化变量、运行构造函数、与其

表3　机制模型到UPPAAL模型的转换规则

序号	机制模型	UPPAAL 对应成分
1	uint *a*[] //type	typedef int[*m*,*n*] *a*;
2	updateExp; //assignment	
3	require(cond);	

4	if(cond) {
	exp
	}
		

5	for (initPart; cond; updateExp){
	exp;
	}
		

6	max(*a*,*b*,*c*)
	//function
	k=function;
	//assignment
		//function　　//assignment
		int max(int *a*,int *b*,int *c*){
		if(*a*>*b*){
		if(*a*>*c*) return *a*;
		else return *c*;
		} else {
		if(*b*>*c*) return *b*;
		else return *a*;
		}
		}
		 

他的时间自动机进行交互；根据算法 2 的转换过程，可以把合约账户角色转换为一个合约时间自动机，如图 5(a)所示，合约时间自动机包含了合约账户角色的所有行为。竞价人的行为是：在拍卖时间内出价、拍卖结束后主动取得退款；根据算法 3 的转换过程，可以把竞价人这一账户角色转换为一个时间自动机，竞价人时间自动机如图 5(b)所示，该时间自动机包含了竞价人的 2 种行为。受益人的行为是：拍卖时间截止的时候主动结束拍卖；根据算法 3 的转换过程，可以将受益人这一账户角色转换为一个时间自动机，生成的时间自动机描述了受益人的行为，受益人时间自动机如图 5(c)所示。最后根据算法 1，综合 UPPAAL 中的系统声明与账户角色转换的时间自动机，将智能合约转换为时间自动机网络模型。

简单公开拍卖合约的全局变量和函数如表 4 所示。根据表 1 中的转换规则将所有的全局变量转换为 UAAPPL 中的全局声明。为了简化建模过程，竞价结束时间、最高的竞拍价格和最高竞价人的地址直接转换为 UPPAAL 中的 int 类型。退款金额映射是指通过用户的地址获取到相应的金额，在 UPPAAL 中用数组表示，数组的下标表示用户的地址，数组值对应着相应的金额。bool 变量 ended 是用来判断竞拍是否的标志，在 UPPAAL 中用 bool 类型声明，且初始值设为 false，表示拍卖没结束。

表4　简单公开拍卖的全局变量和函数

全局变量和函数	含义
address payable public beneficiary	受益人地址
uint public auctionEndTime	竞价结束的时间
address public highestBidder	最高竞价人地址
uint public highestBid	最高的竞拍价格
mapping(address => uint) pendingReturns	地址映射到退款
bool ended	拍卖结束的标志
constructor() {}	构造函数
function bid() {}	竞拍函数
function withdraw() {}	退款函数
function auctionEnd() {}	结束竞拍函数

建立简单公开拍卖时间自动机网络还需要定义一些隐含的辅助变量：全局时钟 time 用来表示整个系统的时间流逝；senderDomain 表示竞价人的范围，可以随机模拟多人竞价；valueDomain 表示竞价人竞价的范围，用来模拟每个竞价人的出价；msg_balance[]表示竞价人的余额，竞价的时候要保证账户的余额大于竞拍价格；contract_balance 表示合约账户的余额，在竞拍结束前，用户的每一次出价都会使合约账户的余额增加，合约账户的初始余额为 0；be_balance 表示受益人的余额，竞拍结束后最高的竞价会转入受益人的账户。

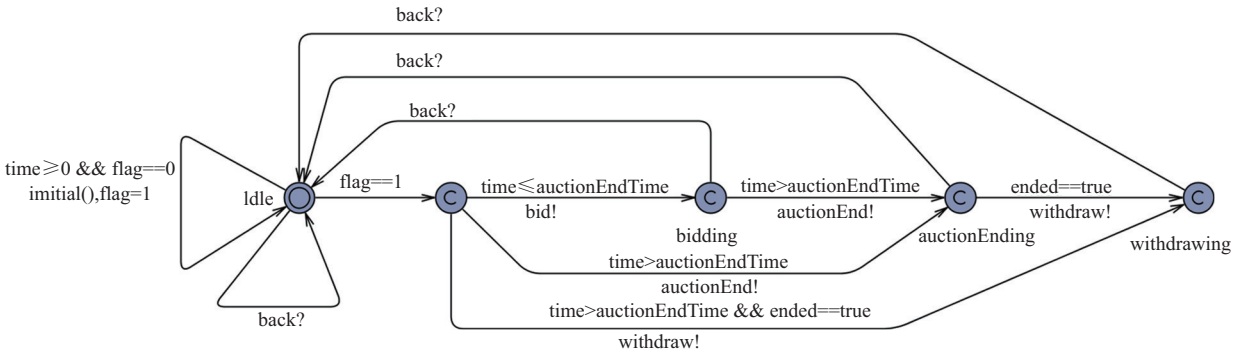
5.1.2　时间安全性验证与分析

本节主要对拍卖合约时间相关的安全性进行验证，下面给出了简单公开拍卖合约的 3 个时间安全性。

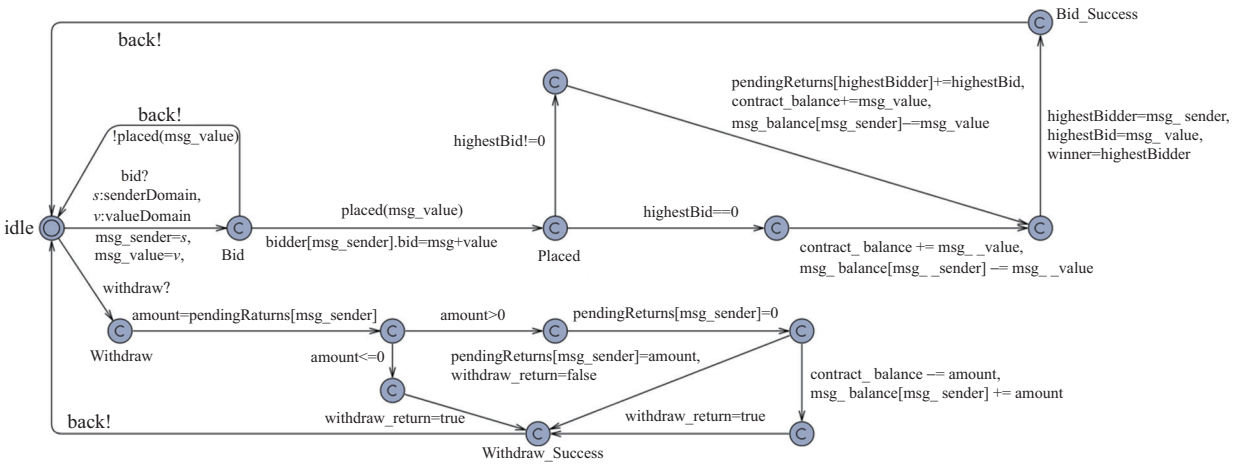
性质 1　竞拍结束后系统会把最高的竞价转给受益人，根据 UPPAAL 规定的语法形式化表示为

$A[]$ (be_balance==highestBid && highestBid != 0) imply (ended==true && time>auctionEndTime)

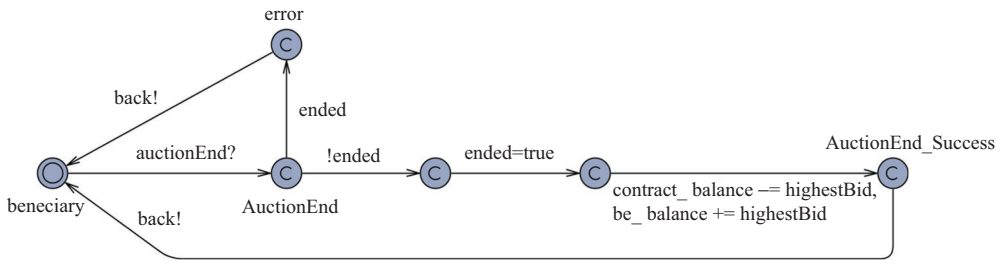
性质 2　只有在到达竞拍时间后才能结束竞



(a) 合约时间自动机



(b) 竞价人时间自动机



(c) 受益人时间自动机

图 5　时间自动机网络模型

拍，根据UPPAAL规定的语法形式化表示为

A[] Beneficiary. AuctionEnd_Success imply (ended == true && time > auctionEndTime)

性质3　竞拍结束后合约账户的余额最终将会变为0，根据UPPAAL规定的语法形式化表示为

A[] Beneficiary.AuctionEnd_Success imply (contract_balance == 0 && time>auctionEndTime)

竞价人时间自动机模板被设计用于模拟所有竞价人的出价过程。该模板包括2个参数：senderDomain和valueDomain，分别表示竞价人范围和出价范围。在每一次时间安全性验证与分析实验中，UPPAAL工具会依次选择一个竞价人和一个出价，以模拟竞拍过程。由于UPPAAL工具验证时会遍历整个状态空间，因此当某个参数范围扩大时，整个模型状态空间就会随之增大，导致验证时间随着竞拍人数的增加而显著增加。为了避免运行时间过长，本文通过在实验过程中固定valueDomain范围，改变senderDomain范围，对2种模型进行了对比实验，简单公开拍卖合约验证时间比较如表5所示。

表5的结果表明了在同样的实验参数下，基于账户角色的方法建立的模型，其验证效率比基于函数的方法建立的模型更高，并且在senderDomain为[0, 6]时，基于函数的方法建立的模型由于状态空间过大已不能验证，而基于账户角色的方法建立的模型仍然能有效地验证。

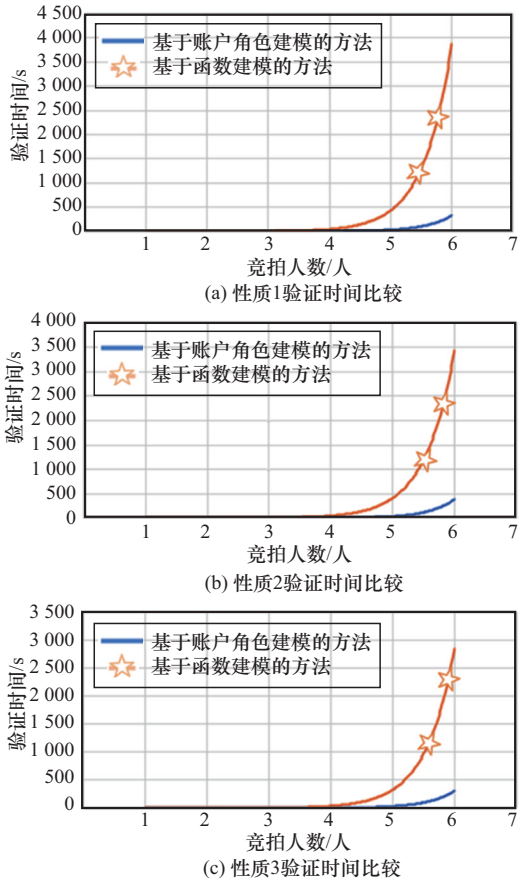
根据表5的结果，固定竞价范围，逐渐增加参与竞拍的人数，绘制2种方法的验证时间对比曲线，如图6所示。从图6中可以很直观地看出，在同样的参数下，基于账户角色的方法建立的模型验证时间更短，验证效率更高。

5.2　拍卖合约的公平性建模与验证

基于第4节提出的建模方法，本节通过一个加密罗马拍卖合约的案例来进行建模与公平性验证。具体过程包括提取合约机制、建立合约机制模型、对4种公平性进行形式化描述及使用UPPAAL工具进行验证。

表5　简单公开拍卖合约验证时间比较

验证性质	时间约束模式[11]	实验参数	验证结果	验证时间/s	
				基于账户角色建模的方法	基于函数建模的方法
性质1	五	senderDomain[0,1]	满足	0.002	0.012
		senderDomain[0,2]		0.125	2.146
		senderDomain[0,3]		0.406	7.585
		senderDomain[0,4]		9.307	193.142
		senderDomain[0,5]		26.533	531.122
		senderDomain[0,6]		216.499	内存耗尽
性质2	二、五	senderDomain[0,1]	满足	0.001	0.014
		senderDomain[0,2]		0.071	1.314
		senderDomain[0,3]		0.407	31.494
		senderDomain[0,4]		12.019	128.754
		senderDomain[0,5]		38.782	359.193
		senderDomain[0,6]		149.278	内存耗尽
性质3	五	senderDomain[0,1]	满足	0.001	0.014
		senderDomain[0,2]		0.106	1.283
		senderDomain[0,3]		0.250	4.758
		senderDomain[0,4]		5.988	126.852
		senderDomain[0,5]		38.255	357.847
		senderDomain[0,6]		149.021	内存耗尽



(a) 性质1验证时间比较

(b) 性质2验证时间比较

(c) 性质3验证时间比较

图 6　简单公开拍卖合约时间性质验证比较

5.2.1　机制模型提取

拍卖合约 Solidity 代码如下所示，是一个简化的以太坊智能合约，名为 CryptoRomeAuction，以 Solidity 语言编写。

```
1) contract CryptoRomeAuction {
2) /** Mechanism
3) individual(msg.sender , msg.value , VALUE)
4) allocate(highestBidder)
5) price(highestBid)
6) outcome(bid())
7) valuation(VALUE)
8) utility(VALUE-highestBid)
9) **/
10) uint 256 public highestBid = 0 ;
11) address payable public highestBidder ;
12) mapping(address=>uint) refunds ;
13) function bid( ) public payable {
14) uint duration = 1;
15) if (msg.value < (highestBid + duration)) {
16) revert( );
17) }
18) if (highestBid != 0) {
19) refunds [highestBidder] += highestBid;
20) }
21) highestBidder = msg.sender ;
22) highestBid = msg.value ;
23) }
24)}
```

该合约为基于区块链的策略机制实现了一种公开英语拍卖的变体，即玩家可以使用加密货币购买虚拟土地。拍卖有一个预定义的生命周期，由开始和结束时间参数化。参与者可以通过向合约发送指示投标价值的信息进行投标。参与者的地址和出价金额分别存储在变量 msg.sender 和 msg.value 中。当前出价最高的投标人的地址记录在最高投标人（第 11 行）中，并使用映射退款来保存每个参与者的出款（第 12 行），以便以后可能退款。Bid()函数（第 13~23 行）在收到消息时触发。如果投标金额不超过当前最高投标和最小增量值持续时间之和，则投标将被拒绝（第 15~17 行）。否则，先前的最高出价者将获得退款（第 18~20 行），最高出价者（第 21 行）和最高出价（第 22 行）将相应更新。该合约的全局变量和函数及其含义如表 6 所示。

表 6　CryptoRomeAuction 的全局变量和函数及其含义

全局变量和函数	含义
uint Bid[][]	参与人出价
uint VALUE[]	参与人估价
uint UTILITY[]	参与人效用
ALLOCATE	最高的竞价人地址
PRICE	最高的竞拍价格
constructor() {}	构造函数
function bid() {}	竞拍函数
function allocate() {}	分配函数
function price() {}	传递函数
function valuation() {}	估值函数
function utility() {}	效用函数

根据上述描述的合约机制，本文建立了如下所示的拍卖智能合约机制模型。

```
1) contract K-PlayerMechanism{
2) uint BID[k][m] ; //assumption
```

```
3) uint VALUE[k] ; //assumption
4) uint UTILITY[k][m] ; //assumption
5) for (uint i=0 ; i<k ; i++) {
6) //Example: msg.sender = i
7) require(@individual.id == toStr(i)) ;
8) for(uint j=0;j<m;j++){
9) // Example: msg.value = Bid[i][j]
10) require(@individual.bid==BID[i][j]);
11) //Example: bid() function inlined
12) require(@outcome);
13) //Example: ALLOCATE = highestBidder
14) ALLOCATE = @allocate ;
15) //Example: PRICE = highestBid
16) PRICE = @price ;
17) VALUATION = @valuation;
18) UTILITY[i][j] = @utility;
19) }
20) UTILITY[i][n] = Max(UTILITY[i][0], … , UTILITY[k][m-1]);
21) @individual.bid = BID[i][n];
22) }
23)}
```

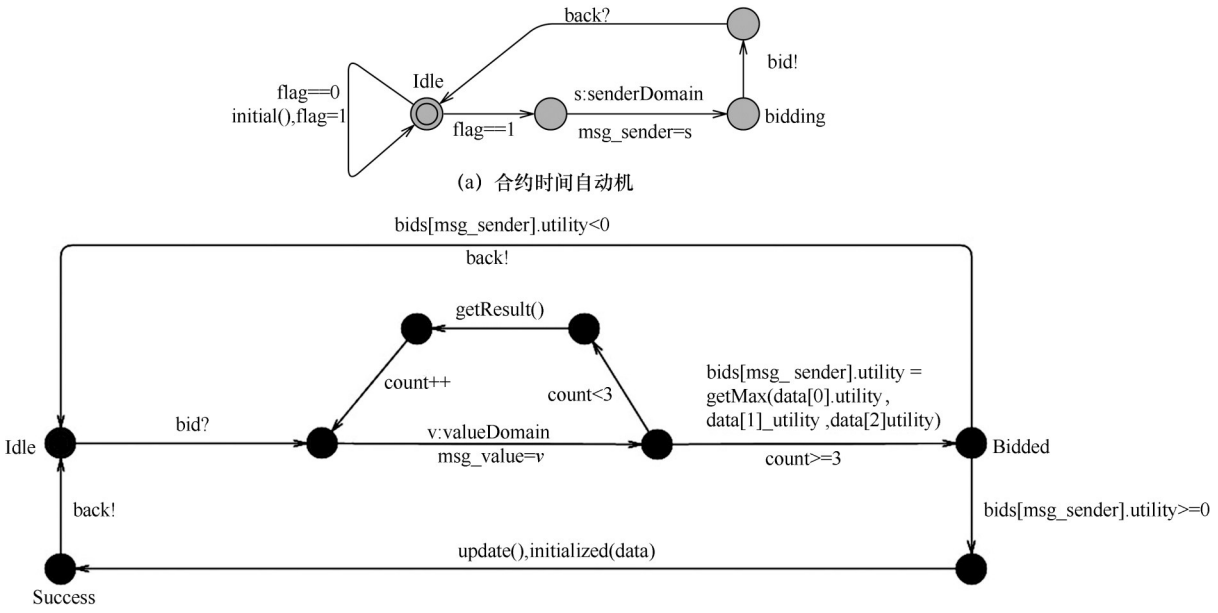
机制模型的全局变量需要在UPPAAL中进行变量声明。为了简化后续抽象建模的过程，在UPPAAL中使用int类型的数组来声明合约参与人的出价、对物品的估价和参与拍卖获得的效用，数组的下标k对应着参与账户的地址，是参与人的标识，其他的全局变量和函数，则直接根据图4所示的转换规则，转换为UPPAAL的声明。

5.2.2　机制模型转换为时间自动机网络模型

通过分析拍卖智能合约的机制模型，根据账户角色的不同行为可以抽象出2种账户角色：合约（合约账户）、竞价人（外部账户）。合约机制模型的行为包括初始化变量、运行构造函数、与其他的时间自动机进行交互；根据算法5的转换过程，可以把合约账户角色转换为一个参与人时间自动机，如图7(a)所示，合约时间自动机包含了合约账户角色的所有行为。竞价人的行为是提交出价、根据出价计算其获得的效用值；根据算法6的转换过程，可以把竞价人这一账户角色转换为一个合约时间自动机，如图7(b)所示，该时间自动机包含了竞价人的2种行为。最后根据算法4，综合UPPAAL中的系统声明与账户角色转换的2个时间自动机，将智能合约转换为时间自动机网络模型。

5.2.3　公平性验证与分析

本节主要是对加密罗马拍卖合约机制模型公平性属性进行验证。一般来说，所有的属性都可以用第2.2节中定义的机制模型来表示。本文将重点验证Liu等[6]提出的基于机制模型的公平性，包括真实性、最优性、高效性和无共谋性这4种属性。



(a) 合约时间自动机

(b) 参与人时间自动机

图7　时间自动机网络模型

真实性。在形式上，当且仅当 $\forall\theta_{-i},\hat{\theta}_i\in\Theta_i$，$u_i(\theta_i,\theta_{-i})\geqslant u_i(\hat{\theta}_i,\theta_{-i})$ 时，一个机制才是真实的。给定一个拍卖智能合约，如果拍卖阻止竞标者通过降低出价获得更多利益，那么它是真实的。当不诚实的竞价不是一个好的策略时，拍卖通常可以吸引更多诚实的投标人，拍卖师可以从出售的商品中获得更高的收入。

高效性。在形式上，当且仅当 $\forall\hat{\theta}_i\in\Theta,\forall d'$，$\sum_i v_i(d(\hat{\theta}),\theta_i)\geqslant\sum_i v_i(d'(\hat{\theta}),\theta_i)$ 时，该机制是高效的，此时分配函数达到最大总值。假设没有投标人可以影响其他投标人的估价。如果唯一的赢家是对商品估价最高的竞标者，那么拍卖就是高效的。

最优性。当且仅当其传递函数达到最大总值，即 $\forall\hat{\theta}_i\in\Theta,\forall t',\sum_i t_i(\hat{\theta})\geqslant\sum_i t_i'(\hat{\theta})$ 时，该机制是最优的。同样的，如果获胜者是出价最高的人，那么拍卖就是最优的。在这种情况下，拍卖商获得的收入最高。

2-无共谋性。如果不存在 2 个个体 i 和 j 之间的合谋，其不诚实策略增加了群体效用，则机制是 2-无共谋的，形式上表示为 $u_i(\hat{\theta}_i,\hat{\theta}_j,\theta_{-ij})+u_j(\hat{\theta}_i,\hat{\theta}_j,\theta_{-ij})\geqslant u_i(\theta_i,\theta_j,\theta_{-ij})+u_j(\theta_i,\theta_j,\theta_{-ij})$。

在拍卖和其他多人游戏中，共谋是一个大问题。拍卖中基本的 2-无共谋性是指任何 2 个投标人的合谋都不能帮助他们获得更高的收益。这在一定程度上防止了竞价操纵，这有助于保证所有竞买人的公平机会，并为拍卖商保持良好的收入。

将上述 4 个属性转化为如下 UPPAAL 可验证的 CTL 公式。

真实性根据 UPPAAL 规定的语法形式化表示为

A[] forall(*i*:int[0,4]) bids[*i*].price != bids[*i*].value imply bids[*i*].utility ⩽ uti[*i*]

高效性根据 UPPAAL 规定的语法形式化表示为：

A[] forall(*i*: int[0, 4]) forall(*j*: int[0, 4]) bids[*i*].value > bids[*j*].value imply bids[*i*].utility > bids[*j*].utility

最优性根据 UPPAAL 规定的语法形式化表示为

A[] forall(*i*: int[0, 4]) forall(*j*: int[0, 4]) bids[*i*].price > bids[*j*].price && bids[*j*].price == highestBid imply bids[*i*].outcome > bids[*j*].outcome

2-无共谋性根据 UPPAAL 规定的语法形式化表示为

A[] forall(*i*: int[0, 4]) forall(*j*: int[0, 4]) bids[*i*].price != bids[*i*].value && bids[*j*].price != bids[*j*].value imply (bids[*i*].utility + bids[*j*].utility) ⩽ (uti[*i*] + uti[*j*])

本节给出了 CryptoRomeAuction 拍卖合约的 4 种公平属性，对其机制模型及属性进行了 UPPAAL 建模与验证。将验证结果与采用符号执行的结果进行了对比，拍卖合约公平性验证结果如表 7 所示。

表 7　　拍卖合约公平性验证结果

验证性质	验证结果	验证时间/s	
		账户角色的方法	符号执行的方法[6]
真实性	不满足	0.015	0.038
高效性	不满足	0.001	0.008
最优性	不满足	0.015	0.033
2-无共谋性	不满足	0.016	0.035

根据表 7 的结果显示，该合约不满足 4 种公平性属性中的任何一种，此外，在同样参数下，采用本文提出的建模方法进行属性验证比符号执行的方法验证的时间有所减少，并且很好地保持了属性验证的准确率，证明了本文提出方法在提高模型以及属性验证效率上的有效性。

相比于文献[6]中符号执行的方法，本文方法在模型上和验证结果上，都更加简单直观。对于不满足的属性，真实性验证结果如图 8 所示，以真实性为例，UPPAAL 工具的模拟器会给出一条可视化的反例路径，如图 9 所示，通过这个路径，可以很容易地找到模型存在不足的地方，从而加以改进。

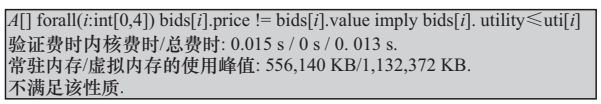


图 8　真实性验证结果

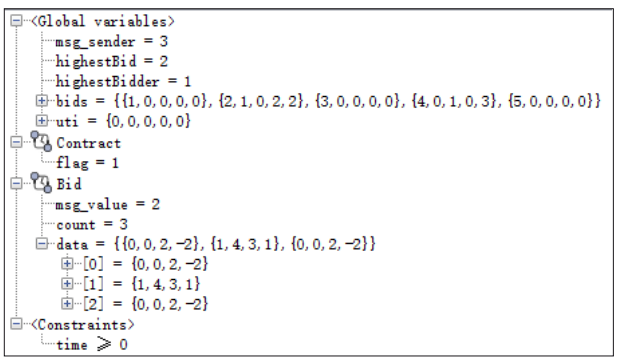


图 9　反例路径

6　结束语

本文提出了一种验证拍卖类智能合约时间安全性和公平性的抽象建模方法，并分别在简单公开拍卖合约和加密罗马拍卖合约2个实例上进行了实验验证。结果表明，本文的方法有效验证了拍卖合约的时间安全性和公平性。在未来，笔者希望将这种方法扩展应用到更多类型的智能合约协议中，如投票合约协议。同时，由于本文智能合约机制模型的提取部分是手工进行的，希望能够开发一个工具，实现从智能合约Solidity代码中自动化提取其机制。

参考文献：

[1] TOLMACH P, LI Y, LIN S W, et al. A survey of smart contract formal specification and verification[J]. ACM Computing Surveys, 2021, 54 (7): 148.

[2] CHRISTOPH F S. Introducing Ethereum and solidity: foundations of cryptocurrency and blockchain programming for beginners[J]. Computing Reviews, 2019, 60(1): 3.

[3] MEHAR M I, SHIER C L, GIAMBATTISTA A, et al. Understanding a revolutionary and flawed grand experiment in blockchain[J]. Journal of Cases on Information Technology, 2019, 21(1): 19-32.

[4] 丁伟. 基于区块链的网络资源公平拍卖与交易框架研究[D]. 合肥: 安徽大学, 2020.
DING W. Research on fair auction and trade framework of network resources based on blockchain[D]. Hefei: Anhui University, 2020.

[5] BARTOLETTI M, CARTA S, CIMOLI T, et al. Dissecting Ponzi schemes on Ethereum: identification, analysis, and impact[J]. Future Generation Computer Systems, 2020, 102: 259-277.

[6] LIU Y, LI Y, LIN S W, et al. Towards automated verification of smart contract fairness[C]//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM Press, 2020: 666-677.

[7] WU S K, CHEN Y J, WANG Q, et al. CReam: a smart contract enabled collusion-resistant e-auction[J]. IEEE Transactions on Information Forensics and Security, 2019, 14(7): 1687-1701.

[8] 郑红军, 张乃孝. 软件开发中的形式化方法[J]. 计算机科学, 1997, 24(6): 90-96.
ZHENG H J, ZHANG N X. Formal methods in software development[J]. Computer Science, 1997, 24(6): 90-96.

[9] PAULSON L C. Isabelle: The next 700 theorem provers[J]. Computer Science, 1990, 31: 361-386.

[10] DEBBI H. Counterexamples in model checking-a survey[J]. Informatica: An International Journal of Computing and Informatics, 2018, 42(2): 145-166.

[11] 赵颖琪, 朱雪阳, 李广元, 等. 智能合约的时间约束模式及其形式化验证[J]. 软件学报, 2022, 33(8): 2875-2895.
ZHAO Y Q, ZHU X Y, LI G Y, et al. Time constraint patterns of smart contracts and their formal verification[J]. Journal of Software, 2022, 33 (8): 2875-2895.

[12] SAYEED S, MARCO-GISBERT H, CAIRA T. Smart contract: attacks and protections[J]. IEEE Access, 2020, 8: 24416-24427.

[13] BADRUDDOJA S, DANTU R, HE Y Y, et al. Making smart contracts smarter[C]//Proceedings of the 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). Piscataway: IEEE Press, 2021: 1-3.

[14] MOSSBERG M, MANZANO F, HENNENFENT E, et al. Manticore: a user-friendly symbolic execution framework for binaries and smart contracts[C]//Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE Press, 2019: 1186-1189.

[15] TSANKOV P, DAN A, DRACHSLER-COHEN D, et al. Security: practical security analysis of smart contracts[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2018:67-82.

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

[17] CIMATTI A, CLARKE E, GIUNCHIGLIA F, et al. NuSMV: a new symbolic model verifier[C]//Proceedings of the 11th International Conference on Computer Aided Verification. Berlin: Springer, 1999: 495-499.

[18] LIU Z T, LIU J. Formal verification of blockchain smart contract based on colored petri net models[C]//Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). Piscataway: IEEE Press, 2019: 555-560.

[19] SHISHKIN E. Debugging smart contract's business logic using symbolic model checking[J]. Programming and Computer Software, 2019, 45(8): 590-599.

[20] ZHU W J. PPTL model checking for blockchains[C]//Proceedings of the 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC). Piscataway: IEEE Press, 2020: 792-795.

[21] 李书霞, 王国卿, 庄雷. 区块链智能合约安全的逆向实时模型检测方法[J]. 小型微型计算机系统, 2020, 41(10): 2030-2035.
LI S X, WANG G Q, ZHUANG L. Reverse real-time model detection method for blockchain smart contract security[J]. Journal of Chinese Computer Systems, 2020, 41(10): 2030-2035.

[22] 吴皓, 周世龙, 史东辉, 等. 符号执行技术及应用研究综述[J]. 计算机工程与应用, 2023, 59(8): 56-72.
WU H, ZHOU S L, SHI D H, et al. Review of symbolic execution

technology and applications[J]. Computer Engineering and Applications, 2023, 59(8): 56-72.

[23] KALRA S, GOEL S, DHAWAN M, et al. ZEUS: analyzing safety of smart contracts[C]//Proceedings of 2018 Network and Distributed System Security Symposium. Reston: Internet Society, 2018: 1-12.

[24] VUJIČIĆ D, JAGODIĆ D, RANĐIĆ S. Blockchain technology, Bitcoin, and Ethereum: a brief overview[C]//Proceedings of the 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH). Piscataway: IEEE Press, 2018: 1-6.

[25] 周匀. 基于承诺的区块链智能合约研究[D]. 上海: 上海交通大学, 2018.
ZHOU Y. Research on commitment-based blockchain smart contract[D]. Shanghai: Shanghai Jiao Tong University, 2018.

[26] MATTEW O. Mechanism theory[M]. Pasadena: Humanities and Social Sciences, 2000.

[27] BENGTSSON J, LARSEN K, LARSSON F, et al. UPPAAL—a tool suite for automatic verification of real-time systems[C]//International Hybrid Systems Workshop. Berlin: Springer, 1995: 232-243.

[28] 王晓楠, 符劲轩, 虞红芳, 等. 基于抽象原则和模型检测的网络协议安全分析[J]. 北京邮电大学学报, 2021, 44(2): 40-46.
WANG X N, FU J X, YU H F, et al. Network protocol security analysis based on abstract principle and model detection[J]. Journal of Beijing University of Posts and Telecommunications, 2021, 44(2): 40-46.

[29] 张濛藜, 马佳利, 刘子昂, 等. 以太坊Solidity智能合约漏洞检测方法综述[J]. 计算机科学, 2022, 49(3): 52-61.
ZHANG Y L, MA J L, LIU Z A, et al. Overview of vulnerability detection methods for ethereum solidity smart contracts[J]. Computer Science, 2022, 49(3): 52-61.

[作者简介]

王昌晶（1977– ），男，江西丰城人，博士，江西师范大学教授、博士生导师，主要研究方向为可信软件、智能化软件等。

欧阳俊媛（1999– ），女，江西萍乡人，江西师范大学硕士生，主要研究方向为模型检测、区块链智能合约等。

张取发（1997– ），男，江西上饶人，江西师范大学硕士生，主要研究方向为模型检测、区块链智能合约安全性验证、下一代互联网等。

左正康（1980– ），男，江西抚州人，博士，江西师范大学副教授、硕士生导师，主要研究方向为形式化方法、智能化软件等。

程着（1988– ），男，湖北随州人，博士，江西师范大学讲师，主要研究方向为形式化方法、实时系统、边缘计算等。

卢家兴（1976– ），男，江西九江人，江西师范大学副教授，主要研究方向为模型检测、区块链智能合约安全性验证、下一代互联网等。