Formal Modeling and Verification of Smart Contracts

Xiaomin BAI
State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
(86)13051386669
baixiaomin@buaa.edu.cn

Zijing CHENG
State Key Laboratory of Space-Ground Integrated Information Technology, Beijing Institute of Satellite Information Engineering, Beijing 100191, China
linuxdemo@126.com

Zhangbo DUAN
State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
(86)18610613100
duan.z@qq.com

Kai HU
State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
(86)010-82339460
hukai@buaa.edu.cn

ABSTRACT

Smart contracts can automatically perform the contract terms according to the received information, and it is one of the most important research fields in digital society. The core of smart contracts is algorithm contract, that is, the parties reach an agreement on the contents of the contract and perform the contracts according to the behaviors written in certain computer algorithms. It not only needs to make sure about the correctness of smart contracts code, but also should provide a credible contract code execution environment. Blockchain provides a trusted execution and storage environment for smart contracts by the distributed secure storage, consistency verification and encryption technology. Current challenge is how to assure that smart contract can be executed as the parties' willingness. This paper introduces formal modeling and verification in formal methods to make smart contract model and verify the properties of smart contracts. Formal methods combined with smart contracts aim to reduce the potential errors and cost during contract development process. The description of a general and formal smart contract template is provided. The tool of model checking, SPIN, is used to verify the correctness and necessary properties for a smart contract template. The research shows model checking will be useful and necessary for smart contracts.

CCS Concepts

• **Software and its engineering →Formal methods**

• **Software and its engineering → Model checking**

Keywords

Smart contracts; Formal methods; Model checking; Modeling; Formal Verification; SPIN.

1. INTRODUCTION

On June 17, 2016, the DAO[1][2] smart contract running on Ethereum's public chain was attacked, and the public funds raised by smart contract were continually being recused by a function to its subcontracts. This attack involved more than 300 thousand dollars, and how to avoid being attacked is a serious challenge for smart contracts. The DAO is essentially a VC (venture capital fund) and funds raised through Ethereum[3] are locked in a smart contract and no one can spend the money alone. The incident was caused by the exploitation of a bug in The DAO's smart contract itself. Therefore, the security and trustworthiness issue of smart contracts arouse people's attention. How to write smart contracts with high reliability and high security has become an urgent problem to be solved.

Smart contract (or contract for short) is one of the basic concepts to solve the code contract proposed by Nick Szabo in 1994in the paper "Formalizing and Securing Relationships on Public Networks"[4]. He gave a car deal scene: a car loan, if the lender does not repay, the car smart contract would automatically withdraw the digital car key. There is no doubt that car dealers will find this automatic contract attractive. Smart contracts utilize protocols and user interfaces to facilitate all steps of the contracting process and obviate the ambiguity of the contractual clauses. Smart contracts aim to reduce mental and computational transaction costs imposed by either principal, third parties, or their tools, it is one of the necessary conditions to build the digital society.

Smart contracts, as a new technology in computational law[5], has a very important feature: when certain conditions are met, contracts would execute appropriate actions automatically. However, this feature has been applied in similar technology in other applications. For example, knowledge-based systems had this feature in 1980s. The first one is rule-based systems. When certain conditions are met, the corresponding rule will be triggered. If several rules are triggered at the same time, there will be a corresponding resolution mechanism to coordinate execution of these rules. The second one is blackboard architecture. There are multiple agents monitoring simultaneously. When a certain condition is met, the corresponding agent will active its own rule and execute corresponding process. The different point from rule-based system is that these agents can be grouped, and these agents those are in the same group will be in the same platform and share the same information. The third one is the database trigger. When a change in the data in the database satisfies the conditions for the database trigger, the corresponding program will be activated to perform. The last one is service-oriented system. When the service caller meets the certain condition, system will provide corresponding service to the service caller.

Szabo's smart contracts theory and the Internet (World Wide Web) appeared almost at the same time, but the application has been far behind the theory, there is no clear idea to make it true. There are two problems mainly. Firstly, there is no way to control the physical property effectively. Vending machines can control the ownership of goods by storing goods in the boxes, but the computer program is difficult to control real-world assets, such as cash, shares. Secondly, there is no trusty execution environment for smart contracts, where the contractors can observe and verify the performances of other contractors. The blockchain[6] is one way to solve these problems, it is not only a safe distributed ledger to store the contract code, but also a distribution execution environment to control the digital assert directly. The blockchain nodes will execute the contract code in a distributed way, which is like the law and regulation executor of commercial transactions, supervision and management, and it reduces plenty of cost of escrow. Today, many blockchain systems, such as Ethereum[3], not only provide the blockchain platforms, but also have the contract programming language with it.

Contract states in blockchain cannot be changed without correct transactions, and each change of state on it needs to go through the blockchain’s consistency algorithm. Ethereum stores the contract itself and its state in the blockchain, when the terms and conditions of the contract are met, the contract code stored in the blockchain will be executed. Since the execution of smart contracts in Ethereum is completed by distributed nodes, so there is no single point failure, and the smart contracts' execution will be immutable and verifiable. Therefore, there is much room for the development to combine smart contracts and blockchain, many companies focus on the research on blockchain and smart contracts, such as Codius, SmartContract, IBM[7] and Eris, etc.

But the development of smart contract exists many critical problems. For example, how do people trust smart contracts is fair for every parity in smart contracts? Is there no bug in the program? If the contract is in favor of the one contract party obviously, how to fix? How to verify the logic of contract is correct and how to eliminate the loopholes in the contract?

Smart contracts must meet at least the following two conditions, so that people can trust and use smart contracts: 1) Smart contracts is executable code, it cannot have any grammar errors and sematic errors; 2) Smart contracts have higher requirements for the correctness and several related properties to make sure safety of the asserts, so it needs a way to generate the credible contract code.

Formal methods[8] are mathematical-based techniques that describe the attributes of the system for the specification, development and validation of computer software. Using formal methods for software design is expected to be able to use the appropriate mathematical analysis to enhance the reliability and robustness of the design, as in other engineering disciplines. Among them, a very important step for formal methods is formal verification. Formal verification can be a more formal way to produce procedures. For example, you can go from specification to program properties or tessellation.

In this paper, we introduced formal modeling and verification[9], which aims to assure the correct and the security of smart contracts, so that users can trust smart contracts code.

The main contributions of this paper are as follows:

- Formal methods are applied to smart contract for the correctness and security;
- A general description of smart contracts template is given, and it can be represented by tuple and finite state machine;
- The PROMELA[10] model of smart contract will be made. And we can verify the whole properties of shopping smart contract.

2. RELATED WORK

2.1 Smart Contract

The premise of modeling is that the properties of model must be understood clearly. We can learn more characters and working principle about smart contracts from the following papers.

Ethereum is the earliest platform to use smart contracts combined with electronic coin. [11] proposed Ethereum including a new protocol and a new coin based on bitcoin. Ethereum is focused on smart contracts. Ethereum put forward decentralized autonomous organizations(DAOs) and a Turing-complete programming language to encode smart contract.

[12] described how a distributed peer-to-peer network works and also researched smart contracts-scripts that reside on the blockchain in Internet of Things(IoT). The paper provided several issues when blockchain-smart contracts are combined with IoT, and made a conclusion that blockchain-smart contract-IoT combination is powerful and can cause significant transformations across several industries.

Through observing the security when Ethereum smart contracts are running in an open distributed network. [13] proposed some new security problems in which an adversary can manipulate smart contract execution to gain profit. And it proposed ways to enhance the operational semantics of Ethereum to make contracts less vulnerable.

2.2 The Correctness and Safety Research in Smart Contract

In the field of smart contract, correctness and security are the most important factors to determine whether smart contract can be used in specific application scenarios. Some security researches have been proposed several times, but formal methods and available implementations are still few.

Writing trustworthy smart contracts can be extremely difficult due to the intricate semantics of EVM and its openness. [14] outlined a framework to analyze and verify both the runtime safety and the functional correctness of Solidity contracts in F*, a functional programming language aimed at program verification.

Numerous common pitfalls are exposed in designing safe and secure smart contracts. [15] documented several typical classes and provided some suggestions to fix or avoid them, and advocate best practices for programming smart contracts. It also resulted in online open course materials to program smart contracts.

Decentralized smart contracts is the next step to development of protocol. The validation of such an early developing technology is as necessary as it is complex. [16] combined theory and formal methods to tackle the new challenges posed by the validation of such systems.

Therefore, smart contract language and its executing process may have potential safety problems. Although formal methods are applied to smart contracts, it is only a framework. A specific model and verification methods have not been given yet.

3. SMART CONTRACTS DESCRIPTION

3.1 Formal Description of Smart Contracts

The execution of smart contracts is from one state to other state, so we can use contract state machine to represents smart contracts.

Contract automata $M^*$is a quintuple:

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

Among them:

- $\mathrm{Q} = \{q_1^*, q_2^*, \cdots, q_m^*\}$. Q is the set about all states of contract execution automata, q_i^* is contained in the state set of contract party, $q_i^* \in q_i, (i = 1, \cdots, m)$;
- $\sum$ is the set of all input events;
- δ^* is the set of all the transit functions, $\delta^*: Q \times \sum \to \mathrm{Q}$;
- $s^*$is the initial state, $s^* \in Q$;
- F^* is the set of termination states, $F^* \subset Q$.

3.2 General Smart Contract Template

Here is a general and simplified template of a smart contract. The contract parties are *Contract party A* and *Contract party B*. The basic functions include: initiated, accepted, interrupted or finished the transaction.

We can describe the contract content in Table 1.

Table 1. General Contract Template

```
General Contract Template ()
Begin
IF Contract party A initiates a transaction AND condition(i) is met
    Set timestamp OR trigger event
ELSE
    transaction failed, Contract party A and Contract party B state regressed and ended the transaction
IF condition(j) is met AND no timeout
    Contract party B confirmed the transaction AND quit
IF TIMEOUT
    transaction failed, Contract party A and Contract party B state regressed and ended the transaction
END
```

This is only a smart contract template. The parties in smart contracts can add the contract terms and set parameters on the template according to their own needs.

The symbols M_1 and M_2 are used to represent the state machine, as Table 2.

Table 2. the contract contents

Contract State Machine	M*
The parties' executing State Machine	*Contract party A*, represented by M_1
	Contract party B, represented by M_2

The state machine M_1 is a quintuple $(q_1, \sum, \delta_1, s_1, F_1)$. q_1 is the executing state of *Contract party A*, as Table 3. $\sum$ is the set of input events. δ_1 is the set of the transit functions, $\delta_1: q_1 \times \sum \to q_1$; s_1 is the initial state, $s_1 \in q_1$; F_1 is the set of termination states, $F_1 \subset q_1$, $F_1 = \{C, E\}$.

Table 3. the set of q_1

State	Description by natural language
s_1	the initial state
A	*Contract party A* waited for the response of *Contract party B*
B	*Contract party A* set timestamp or trigger event to *Contract party B*
C	*Contract party A* finished the transaction
D	Transaction interrupted
E	*Contract party A* state regressed

The state machine M_2 is a quintuple $(q_2, \sum, \delta_2, s_2, F_2)$. q_2 is the executing state of *Contract party B*, as Table 4. $\sum$ is the set of input events, as Table 5. δ_2 is the set of the transit functions, $\delta_2: q_2 \times \sum \to q_2$; s_2 is the initial state, $s_2 \in q_2$; F_2 is the set of termination states, $F_2 \subset q_2$, $F_2 = \{4,5\}$.

Table 4. the set of q_2

State	Description by natural language
s_2	the initial state
1	*Contract party B* accepted order requests
2	*Contract party B* accepted the timestamp or trigger event
3	Transaction interrupted
4	*Contract party B* finished the transaction
5	*Contract party B* state regressed

Table 5. contract events $\sum$

Events Σ	a	*Contract party A* initiates a transaction to *Contract party B*
	b_1	Transaction interrupted
	b_2	*Contract party A* and *Contract party B* responded the transaction
	b_3	*Contract party B* refused the request from *Contract party A*
	c_1	*Contract party A* and *Contract party B* finished the transaction
	c_2	*Contract party A* and *Contract party B* state regressed

(All events would be recorded in smart contracts)

The contract state machine of *Contract party A* and the contract state machine of *Contract party B* can be combined to the whole contract state machine, as Figure 1.

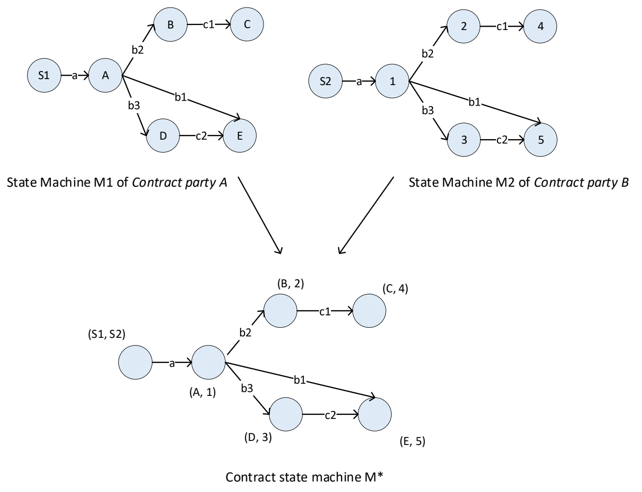

S1
A
B
C
D
E
a
b2
b3
b1
c1
c2
State Machine M1 of Contract party A
S2
1
2
3
4
5
State Machine M2 of Contract party B
(S1, S2)
(A, 1)
(B, 2)
(C, 4)
(D, 3)
(E, 5)
Contract state machine M*

Figure 1. The general smart contract state machine

4. MODELINGAND VERIFICATION

4.1 Formal Description Language and Tool

SPIN[10] is a general tool to verify the correctness of distributed software models in a rigorous and mostly automated fashion. It was written by Gerard J. Holzmann and others in the original Unix group of the Computing Sciences Research Center at Bell Labs, beginning in 1980. The software has been available freely since 1991 and continues to evolve to keep pace with new development in the field.

Systems to be verified are described in PROMELA (Process Meta Language), which supports modeling of asynchronous distributed algorithms as non-deterministic automata (SPIN stands for "Simple PROMELA Interpreter"). Properties to be verified are expressed as Linear Temporal Logic (LTL) formulas, which are negated and then converted into Büchi automata as part of the model-checking algorithm[17]. In addition to model checking, SPIN can also operate as a simulator, following one possible execution path through the system and presenting the resulting execution trace to the user.

4.2 Shopping Smart Contract Model

We provided a shopping smart contract (SSC) and it simplified an internet shopping process. A SSC description is as flows: when the user launches a shopping order, he needs to submit the funds that the shopping needs to the SSC, the SSC keeps the funds. At the same time the SSC starts the two sub processes, the user process and the shop process. User process, if *Merchant* does not deliver the goods within certain days, the user will cancel the transaction, the SSC will return the funds to the user, the user process timing cycle detection delivery status; the shop process: after receiving the order, firstly the SSC judges whether the order is over, if the order is not overtime, then the shop delivers goods, if *Customer* can receive the goods within the stipulated time, *Merchant* would receive the payment. The SSC needs to ensure the security of the funds and the transaction process of the various states of the reachability.

The PROMELA model of SSC is built as Table 6:*user_money* represents the user's money, its initial value is 100; *shop_money* represents the shop's money, its initial value is 0; and *money* represents smart contract that is used to save money temporarily. *max_day* represents the maximum number of days allowed for a transaction; *day* represents the current number of day for a transaction.

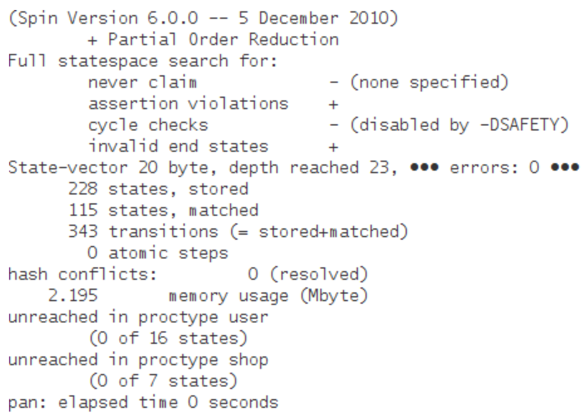
Table 6. Main PROMELA Model of the SSC

```
active proctype user() {
        do
        :: isSend -> atomic{
                shop_money=price;
                money=0;
                break;}
        :: (day>max_day)->{
                user_money=price;
                money=0;
                break;
        }
        :: else -> day=day+1;
        od
}

active proctype shop() {
        do
                ::(day<max_day-1)->isSend=true;
                ::break;
                od
}
```

4.3 Formal Verification

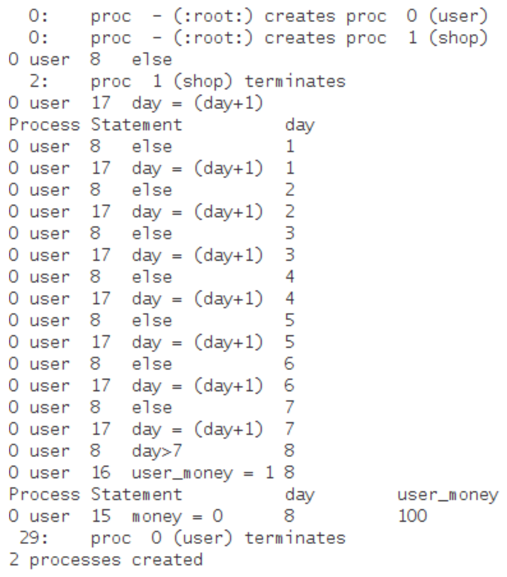
The spin tool is used to detect the SSC model, the simulation results for the model is shown in the Figure 2~Figure 5. From these figures, we can verify the state accessibility, no deadlock and no livelock. The two typical verification results including transaction finished and transaction timeout are shown.



```
(Spin Version 6.0.0 -- 5 December 2010)
        + Partial Order Reduction
Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +
State-vector 20 byte, depth reached 23, ••• errors: 0 •••
      228 states, stored
      115 states, matched
      343 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)
    2.195       memory usage (Mbyte)
unreached in proctype user
        (0 of 16 states)
unreached in proctype shop
        (0 of 7 states)
pan: elapsed time 0 seconds
```

Figure 2. Model verification result

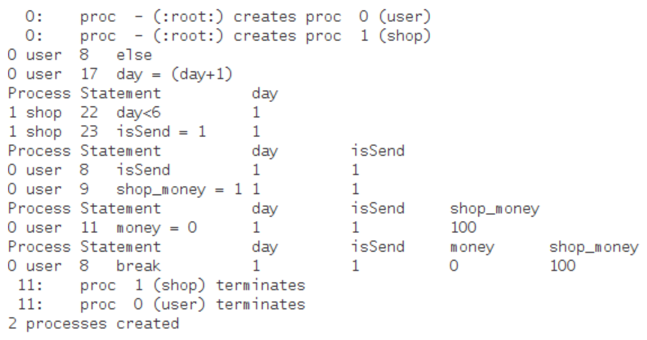
Figure 2 shows the verification result of SSC model. There are 0 errors and the modelgenerates343 transition results including 115 matched state and 228 stored state.



```
  0:    proc  - (:root:) creates proc  0 (user)
  0:    proc  - (:root:) creates proc  1 (shop)
0 user  8   else
  2:    proc  1 (shop) terminates
0 user  17  day = (day+1)
Process Statement          day
0 user  8   else           1
0 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
0 user  8   else           4
0 user  17  day = (day+1)  4
0 user  8   else           5
0 user  17  day = (day+1)  5
0 user  8   else           6
0 user  17  day = (day+1)  6
0 user  8   else           7
0 user  17  day = (day+1)  7
0 user  8   day>7          8
0 user  16  user_money = 1 8
Process Statement          day        user_money
0 user  15  money = 0      8          100
 29:    proc  0 (user) terminates
2 processes created
```

Figure 3. Model simulation results (timeout refund)

In Figure 3, when *day* equals 8th day, the user did not receive the goods, so *user_money*=100 and money=0.The model represents that transaction failed.



```
  0:    proc  - (:root:) creates proc  0 (user)
  0:    proc  - (:root:) creates proc  1 (shop)
0 user  8   else
0 user  17  day = (day+1)
Process Statement          day
1 shop  22  day<6          1
1 shop  23  isSend = 1     1
Process Statement          day        isSend
0 user  8   isSend         1          1
0 user  9   shop_money = 1 1          1
Process Statement          day        isSend      shop_money
0 user  11  money = 0      1          1           100
Process Statement          day        isSend      money       shop_money
0 user  8   break          1          1           0           100
 11:    proc  1 (shop) terminates
 11:    proc  0 (user) terminates
2 processes created
```

Figure 4. Model simulation results (finish)

In Figure 4, the user received the goods in the first day(*day*=1), so *shop_money*=100 and *money*=0. The model represents that transaction succeed.

Figure 5 shows the state transition of one smart contract running. The number in the circle corresponds to the number of rows in the SSC model. The row represents state transition.

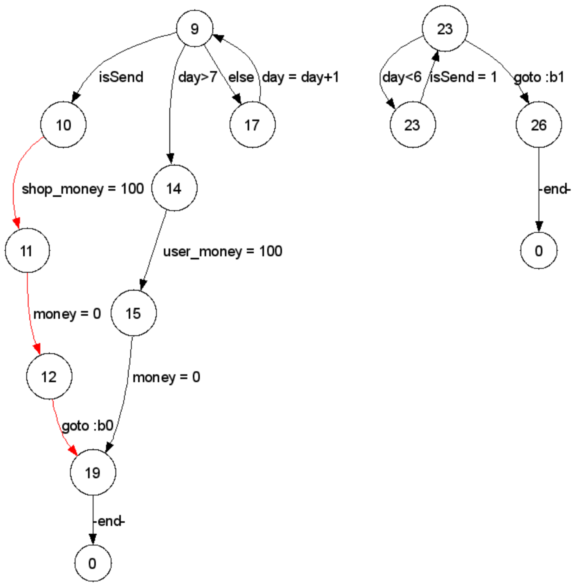


Figure 5. Spin Spider

From these model simulation results, we can verify that SSC is no deadlock, and it only has one state at a moment. It can run as the theoretical state machine. SSC is fair to *customer* and *merchant*. The contract state machine is triggered only if time or events meet the condition.

5. CONCLUSION

This paper introduces the application of smart contracts and some critical issues in smart contracts. It proposed to apply formal methods to smart contracts and gave the description of a general smart contract template. Smart contracts will be an important technology in the future to promote our lives, so the security of smart contract must be assured. Model checking in formal methods can be used for model checking to make smart contracts correct, we can use a model verification tool to verify the correctness and important properties of smart contracts. A case study is verified by a famous model checking tool SPIN to illustrate the verification process and effects. It shows that formal methods can be applied to verify many properties. It will be widely used for the design and development of smart contracts in the future.

6. ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China under Grant 61672074 and 91538202, Funding of Ministry of Education and China Mobile MCM20160203, Project of the State Key Laboratory of Software Development Environment of China under Grant SKLSDE-2016ZX-16.

7. REFERENCES

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

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

[3] Ethereum[EB/OL].http://www.ethereum.org/, 2017.

[4] Szabo N. Formalizing and securing relationships on public networks[J]. First Monday, 1997, 2(9).

[5] Lessig L. Code is law[J]. The Industry Standard, 1999, 18.

[6] Blockchain. URL https://en.wikipedia.org/wiki/Block_chain_(database), 2016

[7] Hyperledger[EB/OL]. https://www.hyperledger.org/, 2017.

[8] Formal Methods. [URL] http://en.wikipedia.orgi/Formal_methods.

[9] Sanghavi, Alok (21 May 2010). "What is formal verification?". EE Times-Asia.

[10] Mikk E, Lakhnech Y, Siegel M, et al. Implementing statecharts in PROMELA/SPIN[C]//Industrial Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE Workshop on. IEEE, 1998: 90-101.

[11] Buterin V. Ethereum white paper: a next generation smart contract & decentralized application platform[J]. 2013.

[12] Christidis K, Devetsikiotis M. Blockchains and smart contracts for the internet of things[J]. IEEE Access, 2016, 4: 2292-2303.

[13] Luu L, Chu D H, Olickel H, et al. Making smart contracts smarter[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016: 254-269.

[14] Bhargavan K, Delignat-Lavaud A, Fournet C, et al. Short Paper: Formal Verification of Smart Contracts[J].

[15] Delmolino K, Arnett M, Kosba A, et al. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab[C]//International Conference on Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2016: 79-94.

[16] Bigi G, Bracciali A, Meacci G, et al. Validation of Decentralised Smart Contracts Through Game Theory and Formal Methods[M]//Programming Languages with Applications to Biology and Security. Springer International Publishing, 2015: 142-161.

[17] Katoen, Joost-Pieter. Principles of model checking, The MIT Press, 2008.The Spin Model Checker — Primer and Reference Manual, Addison-Wesley, 2003. ISBN 0-321-22862-6.