Smart Contract modeling and verification techniques: A survey

Adnan Imeri, Nazim Agoulmine, Djamel Khadraoui

Smart Contract modeling and verification techniques: A survey

Adnan Imeri[1,2], Nazim Agoulmine[2], and Djamel Khadraoui[1]

[1] Luxembourg Institute of Science and Technology
adnan.imeri@list.lu
[2] Universitè of Èvry Val d'Essonne - Paris Saclay University
nazim.agoulmine@univ-evry.fr

Abstract

The capabilities of smart contracts for supporting and enhancing business processes in distributed-decentralized environments have affected the technological transformation of numerous industries. Designing and developing blockchain-based solutions requires model checking and verification of the components of the system such as smart contracts, for well-behave, correct execution and fulfilling of the business process requirements. Certainly, there are concerns about the execution of smart contracts in such distributed environments. This study shows the research results about model checking of smart contracts, performing a deep analysis of current approaches on modeling and verifying smart contracts and reviewing available tools for such practices. Modeling and verifying smart contracts are addressed at the levels of programming and run time execution.

1 Introduction

Blockchain maintains a distributed decentralized and shared ledger, that allows securely exchanging transactions between users. The network of blockchain nodes follows a peer-to-peer communication protocol, and the involved nodes contain the same ledger. The transaction data are governed by consensus protocol, that guarantees the trust and reliability of users (end-user, miners). These transactions are stored into blocks, besides the block characteristics (blockhead, nonce, transactions root), it contains also the hash of the previous block, thus forming the link of the blocks. This presents a fundamental characteristic of blockchain which in addition it supports the properties of immutability, data integrity, and non-repudiation properties [36] [34].

Smart contract (SC) is an autonomous computer programming code, that runs on the blockchain, and it is executed when a certain event happens, based on specified parameters [4]. For the SC that is deployed on the blockchain, a unique address is assigned, that identified the SC. The blockchain users can invoke the SC, by sending a transaction to the SC address [4] [26]. The logic implemented by SC is based on domain-specific and the source of the SC can be a natural language law, scope of any agreement between parties, and other possible sources depend on the business process requirements [29]. For the transaction that is accepted on the blockchain, and if they contain the contract address as a message received, all the miners will execute the code of the SC and react according to the specific tasks given on SC. SC is self-executed programs, moreover, it can invoke another SC, call external service, for fulfilling given task and they have the ability to automatize and implement a wide range of applications of domain-specific [26] [4] [23].

1.1 Formal representation of SC

The mathematical model for the formal representation of the SC in the state-transition system is a quintuple set of elements [2]:

$$M = (Q, \Sigma, \delta, s_0, F), \text{ where}$$

- Q, is finite set of all possible states of the SC;
- Σ, is the set of all input event on the SC;
- δ, is the set of transition-function of the SC , δ: Q x $\Sigma \rightarrow Q$;
- F, is final state the SC, $F \in$ Q;
- s_0, is the initial state of SC, $s_0 \in$ Q;

If the blockchain state is noted by γ, for any successful transaction executed by the SC, the blockchain state will be updated into γ' [26]:

$$\gamma \xrightarrow{Tx} \gamma'$$

The new state γ', might impact many user accounts, or other SC, that might have their impact on the empirical data on the blockchain.

Simultaneously with the advantageous technological characteristics of the SC, still, relevant questions are rising when designing the SC: *"Does the SC is behaving as is intended?";"Does SC fulfill the shared intention of the parties?"; "Are the SC reliable enough to perform complex tasks, e.g., financial transactions?"*

For responding to these questions and other SC vulnerabilities, the formal proof is required to ensure the SC is behaving as intended and fulfilling user requirements.

1.2 The vulnerabilities of SC

The security issues of SC, need high attention since they contain millions of dollars in virtual coins, or they run the business process tasks daily. Primarily, high attention is required before deploying SC. The risks stand on the fact that once deploying SC into the blockchain, they remain immutable (impossible to patch) and there is no way of stopping them [21] [25]. The well-know case of SC vulnerability is *theDAO* attack, which causes the loss of more than sixty million dollars in ethers [21][3].

Amongst the security bugs and other SC issues discovered are [25] [26] [16] :

- *Dependence on transaction-ordering*: Presents security issues where an event (of function) of SC is depended on the other previous event in order to behave correctly.
- *Timestamp*: The dependence of the SC for performing an event.
- *Throwing an uncontrollable exception*: This is the situation when a SC calls another SC, or some function of the SC by using *someThing.send(someValues)*. In case there is an exception for a certain reasons, and the called SC or function returns *false*, the process value *"someValues"*, will not reach the destination. In this situation, there is required that the SC that calls any other SC or function should check priory if the calls are made properly.
- *Reentrancy*: This is a security flaw when a SC function calls another entrusted SC function, on the SC. The called SC, can take control of data flow and make changes over data. The *theDAO* attack sourced from the reentrancy issues [7].

- *Linearizability*[1]: This is a security issue raised in SC when an off-chain service is called.
- *SC misbehaviour*: Present an issue when a SC is not behaving as it was intended.

2 Research methodology and research results

Blockchain and SC are currently in the highest level of interest as research topics from scholars and market researchers. The results in this study signify that the research field of SC and model checking and verification is relatively new based on the research articles published. For achieving significant results on this survey paper, we define a research method for selecting, classifying, and analyzing the most significant research results. Initially, we define main research questions that are the core of this research: *Model-checking techniques for SC?; How to verify the SC is running as it intended?; How to confirm the correctness of SC with natural laws and regulation?; Tools and best practices on verifying SC?*

From these main question, there are formalized set of queries that are used on main research libraries, such as Web of Science, IEEE, ACM, Scopus, Google Scholar, etc. The research method is composed of the following steps:

```
S1: Query definition by using keywords: Smart contract & (or)  formal
modeling, model checking, security, verification, contract validate
tools, compliant smart contract, consistency, correctness.
S2: Search (using S1) for research articles in main research Libraries;
S3: Formalizing the corpus of articles;
S4: Analysis of the abstract and details of articles (selected on S3),
    moreover, classifying the articles based on the research topic:
S5: Eliminating the non-relevant articles and reformulation queries (S1),
    if necessary after discovering other possible research challenges;
S6: Repeating steps S2-S5, until the same results are shown again;
```

The method presented above allows us to formulate a systematic study of SC modeling and verification. There are retrieved a significant number of articles from our research method. The research tendency for modeling and verification of the SC is rising. For practical reasons, we do now show here results and graphs.

3 SC model checking and verification approaches

This section introduces an overview of model checking techniques and further, we highlight the most relevant scientific approaches for model checking and verification of the SC, for responding SC vulnerabilities presented in section 1.2. Table 1, summary the current most relevant tools, frameworks and approaches for a secure and well-behaved SC.

3.1 Overview of model checking and verification techniques

The formal method allows expressing a complex model for a computer system based on mathematical expressions. For obtaining the correct behavior of the model, formal methods use mathematical proofs to ensure the correctness of the model [6]. Further, the model checking techniques allow verifying all the states that are provided by the model. Initially, there is a required specification of the model, mainly by using temporal logic[2] and then systematically

[1] A classic example of linearizability is that all the users involved in the concurrent process should see the same state of data. Source: https://jepsen.io/consistency/models/linearizable

[2] Temporal Logic Model Checking

performing verification over all the specifications defined [9] [6]. This means that all possible "theorems" defined on the specification, need to be examined for all possible states of the model [6]. The model would be possible to be implemented when the previous stages "specification" and "verification" are successfully completed [6]. Model-checking and verification is a way to determine the behavior of the SC. For designing SC that is intended to run correctly and securely on the blockchain, model checking and verification is necessary. Mainly a model checking for SC will provide the necessary proof, to avoid the well-known vulnerabilities of SC 1.2, and possibly to discover new SC security and misbehavior issues.

3.2 The scientific approach for model checking and verification of SC

The *theDAO* attack raised the attention for the researches and scholars to improve and avoid the vulnerabilities and security issues of the SC. Research in [31], uses NuSMV for the expression of the blockchain and SC model. The model is composed of three main parts, highlighting, first, the Ethereum (kernel layer) as a distributed system for managing transactions between users. Secondly, it uses the SC (application layer) that is expressed on Solidity [10], to represent them in model checking language, i.e., in NuSMV, and the third part determines the execution environment for the application [31]. This research is to verify if the SC is behaving as they are expected. For achieving this, the expected properties need to be formalized into temporary logic (Computation Tree Logic (CTL)) [31]. In case the property does not behave as requested, the model-checker produces a counterexample, that allows determining the problem and its genesis [31]. The research in [17] use SPIN [30] for formal verification of the properties of the SC. This research contributes by formally defining SC and providing a model for SC based on PROMELA/SPIN [17]. A formal verification of SC based on user and blockchain behaviors is proposed by research [14]. The author highlights the fact that the previous efforts for capturing the SC vulnerabilities by documenting them and by using formal verification fail because of not considering user and blockchain behaviors. The authors from [19] use the non-cooperative game theory to model the transaction performed by two players. This is possible since the terms of the contract are agreed and the players act independently. A finite-state machine (FSM) based tool, named FSolidM [1], is presented in [28], for designing secured Ethereum based SC. Further, a formal verification for SC behaviour, by using F* [5], is showed in research [18]. The security of SC is an extremely difficult task due to the openness of the blockchain frameworks [18]. The research focuses on the behavior of the SC, and proposes a framework for analyzing and verifying the functional correctness and the run time safety of the SC by using F* [18]. Initially, there is given a clear guide for translating Solidity and bytecode generated from the SC, into F *. Then a detection of vulnerabilities of SC is presented. Besides, verification of the functional correctness of SC by using the Solidity subset into F*, further, the framework proposed in [18] analysis the byte code generated for given SC and intend to prove the equivalent running of SC in solidity level (functional level) and bytecode (runtime level). In [25], the authors present a tool that intends to find the run time errors of SC, in the class of bugs of event-ordering. Basically, the idea behind this research is to see if the output from SC differs when the input order of the event (functions) is changed. In [24], researchers present a a framework for "correctness" at the level of programming aspects and the business process "validity" of SC. The formal verification of SC is performed by using abstract interpretation and symbolic model checking, where SC is taken as input, while the output in XACML style [13] is the generation of correctness or fairness [24]. Also, an intermediate-level programming language for SC is presented on [32], and the intention behind this research is to verify the high-level language programming language, e.g., Solidity, before deploying it into the blockchain. Symbolic verification of the SC is showed in

[26]. The values of program variables are represented by the symbolic parameters. The symbolic paths are formulas over the symbolic input, which these inputs should satisfy [26]. Also, the authors from [26] implemented a tool that verifies the correctness of the SC by using the SC byte code. This tool is able also to catch the famous DAO bug (reentrancy) [3] on the SC. Ethereum is proposing Vyper environment [11] to prevent the reentrancy attack [8]. Another symbolic approach based on the dependency graph, that verifies the SC behavior in the report with given properties and classify it as safe/unsafe is presented in [33].
Besides being focused on verifying the SC, on the programming level, other research highlight the verification of the SC at the runtime level, i.e., bytecode. In [20] a framework for verification of the SC is proposed by combining SC and its specification. The misbehavior of the SC is identified when a specification is violated. The research from [35] uses Coq proof assistant [12] for formal symbolic development and verification of processes of virtual machine (VM). The intention behind this study is to prove the reliability and security of the Ethereum-based SC [35]. The K framework has been used to build a tool that allows formal specification and analysis of the Ethereum VM bytecode of SC [22]. Another approach that applies formal verification of SC at the bytecode level by using Isabelle/HOL, is explained in [15]. The bytecode is structured in a block of code, and further creating a logic for reasoning this code [15].

Model Checking Tools	**Main Characteristics**	**Operation over SC sources**	**Limitations**
NuSMV	model checking-functional correctness	solidity	It does not support the complete expression of a blockchian environment [31]
F*	functional and runtime checking	solidity; bytecode	The presented tool for model-checking SC based on F*, does not support entirely the syntax features of Solidity, e.g., loops [18]
BIP Framework	component based and statistical model checking	solidity and blockchain	The current model does not support entirely the blockcian components, e.g., mining process, block, etc. [14]
Scilla	intermediate checking	solidity	Explicit exception are not covered on this version of Scilla [32]
EthRacer	runtime checking	bytecode	Focused only on event-order bugs by suing notions of linearizability and synchronisation [25]
ZEUS	runtime checking	solidity and bitcode	It requires to add the policy specification of the SC [24]
Oyente	pre-deployment SC checking	bytecode	Limited only on the bytecode, and thus losing the contextual information e.g. types, integer underflow (or overflow) [24]

Table 1: Summery of the main approaches related to modeling and verification of SC

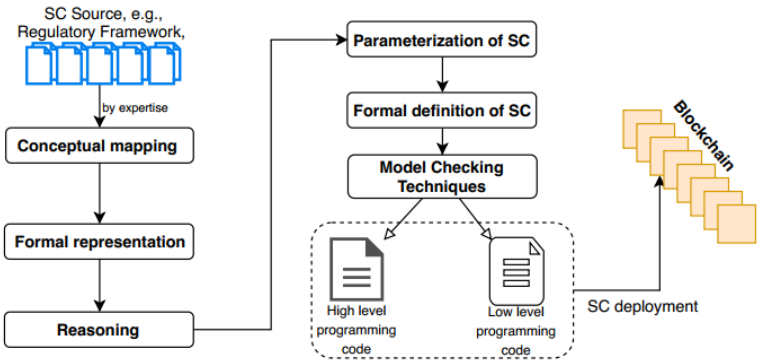

Figure 1: The schema for deriving a secured and well-behaved SC

.

4 Perspectives, conclusions and future works

We consider that the security and auditing aspects of the SC is one of the key steps before deploying and blockchain-based solution. The model checking methods that are showed in this research have essential elements to fulfill the model checking for SC. Considering, the SC are emerging from different sources, based on domain-specific requirements, the above-mentioned model checking methods do not fulfill entirely the model checking components for each SC. For the SC that are emerging from natural language, it is necessary to checker their validity in terms of if natural contract are correctly transformed into SC, and if they are running as they are intended [27]. Thus, an adaptation of the methods (or tools) is necessary for considering all possible gaps in behavior and security aspects of the SC.

For a wide examination of the behavior and security aspects of SC, a strict design method is required by considering in advance the source of SC. In terms of a method (techniques) for model checking of the SC, we propose, initially, formal reasoning over the source of SC, further at the SC level, in code level and run time level. In the context of our proposed approach, initially we select the source of the SC, which is from the perspective of regulatory frameworks, and further, we apply reasoning technique (over ontology's, e.g., by using LIKA) over the concepts that emerge from this regulatory document, e.g., legal or procedural text. Once the behavior of the conceptual model is verified, further we extract the necessary parameters for defining SC. Moreover, we express formally SC, and then we apply the relevant model checking techniques. The outline steps of our proposed approach for a secured and well-behaved SC are showed in Figure 1.

Conclusions: This paper summarizes the model checking techniques for SC, and we propose a new perspective on the way of modeling and verifying SC. To the best of our knowledge, there is not any model that encounter all the components of blockchain and models and verifies them. This is more due to the perspectives of the use case, which means some of the SC need to verify the financial instruments, some of them the interaction between stakeholders, and some of the fulfillment of a given task in the appropriate way.

Future works: We intend to apply our proposed approach, to model and verify the SC in case of anomalies, e.g., accidents or new ad-hoc decisions, on the business process. Meaning that we intend to respond to questions on adapting SC, which allows system running normally without any long disturbance.

References

[1] anmavrid/smart-contracts. https://github.com/anmavrid/smart-contracts. (Accessed on 10/31/2019).

[2] automata2.pdf. https://www3.cs.stonybrook.edu/~cse350/slides/automata2.pdf. (Accessed on 10/28/2019).

[3] The dao attacked: Code issue leads to $60 million ether theft - coindesk. https://www.coindesk.com/dao-attacked-code-issue-leads-60-million-ether-theft. (Accessed on 10/09/2019).

[4] Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf. http://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf. (Accessed on 10/11/2019).

[5] F*: A higher-order effectful language designed for program verification. https://www.fstar-lang.org/. (Accessed on 09/28/2019).

[6] Formal methods. https://users.ece.cmu.edu/~koopman/des_s99/formal_methods/#targetText=Formal%20methods%20are%20system%20design,order%20to%20ensure%20correct%20behavior. (Accessed on 10/16/2019).

[7] Known attacks - ethereum smart contract best practices. https://consensys.github.io/smart-contract-best-practices/known_attacks/. (Accessed on 10/11/2019).

[8] Learn vyper in y minutes. https://learnxinyminutes.com/docs/vyper/#targetText=Vyper%20lets%20you%20program%20on,requiring%20centralized%20or%20trusted%20parties.&targetText=Like%20objects%20in%20OOP%2C%20each,functions%2C%20and%20common%20data%20types. (Accessed on 10/31/2019).

[9] Model checking overview [read-only]. http://www.cs.cmu.edu/~emc/15-398/lectures/overview.pdf. (Accessed on 10/16/2019).

[10] Solidity — solidity 0.5.11 documentation. https://solidity.readthedocs.io/en/v0.5.11/. (Accessed on 09/16/2019).

[11] Vyper — vyper documentation. https://vyper.readthedocs.io/en/v0.1.0-beta.13/. (Accessed on 10/31/2019).

[12] Welcome! — the coq proof assistant. https://coq.inria.fr/. (Accessed on 11/01/2019).

[13] Xacml 3.0 xacml:policy - complete documentation and samples. http://www.datypic.com/sc/xacml30/e-xacml_Policy.html. (Accessed on 10/16/2019).

[14] Tesnim Abdellatif and Kei-Leo Brousmiche. Formal verification of smart contracts based on users and blockchain behaviors models. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5. IEEE.

[15] Sidney Amani, Myriam Bégel, Maksym Bortin, and Mark Staples. Towards verifying ethereum smart contract bytecode in isabelle/hol. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 66–77. ACM, 2018.

[16] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts. *IACR Cryptology ePrint Archive*, 2016:1007, 2016.

[17] Xiaomin Bai, Zijing Cheng, Zhangbo Duan, and Kai Hu. Formal modeling and verification of smart contracts. In *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, pages 322–326. ACM, 2018.

[18] Karthikeyan Bhargavan, Nikhil Swamy, Santiago Zanella-Béguelin, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, and Thomas Sibut-Pinote. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security - PLAS'16*, pages 91–96. ACM Press.

[19] Giancarlo Bigi, Andrea Bracciali, Giovanni Meacci, and Emilio Tuosto. Validation of decentralised smart contracts through game theory and formal methods. In Chiara Bodei, Gianluigi Ferrari,

and Corrado Priami, editors, *Programming Languages with Applications to Biology and Security*, volume 9465, pages 142–161. Springer International Publishing.

[20] Joshua Ellul and Gordon J Pace. Runtime verification of ethereum smart contracts. In *2018 14th European Dependable Computing Conference (EDCC)*, pages 158–163. IEEE, 2018.

[21] Shelly Grossman, Ittai Abraham, Guy Golan-Gueta, Yan Michalevsky, Noam Rinetzky, Mooly Sagiv, and Yoni Zohar. Online detection of effectively callback free objects with applications to smart contracts. *Proceedings of the ACM on Programming Languages*, 2(POPL):48, 2017.

[22] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, et al. Kevm: A complete formal semantics of the ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 204–217. IEEE, 2018.

[23] Adnan Imeri, Nazim Agoulmine, and Djamel Khadraoui. A secure and smart environment for the transportation of dangerous goods by using blockchain and iot devices. 2019.

[24] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. Zeus: Analyzing safety of smart contracts. In *NDSS*, 2018.

[25] Aashish Kolluri, Ivica Nikolic, Ilya Sergey, Aquinas Hobor, and Prateek Saxena. Exploiting the laws of order in smart contracts. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 363–373. ACM, 2019.

[26] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269. ACM, 2016.

[27] Daniele Magazzeni, Peter McBurney, and William Nash. Validation and verification of smart contracts: A research agenda. 50(9):50–57.

[28] Anastasia Mavridou and Aron Laszka. Designing secure ethereum smart contracts: A finite state machine based approach. In *International Conference on Financial Cryptography and Data Security*, pages 523–540. Springer, 2018.

[29] Eliza Mik. Smart contracts: terminology, technical limitations and real world complexity. *Law, Innovation and Technology*, 9(2):269–300, 2017.

[30] Erich Mikk, Yassine Lakhnech, Michael Siegel, and Gerard J Holzmann. Implementing statecharts in promela/spin. In *Proceedings. 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*, pages 90–101. IEEE, 1998.

[31] Zeinab Nehai, Pierre-Yves Piriou, and Frederic Daumas. Model-checking of smart contracts.

[32] Ilya Sergey, Amrit Kumar, and Aquinas Hobor. Scilla: a smart contract intermediate-level language. *arXiv preprint arXiv:1801.00687*, 2018.

[33] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 67–82. ACM, 2018.

[34] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimba. A taxonomy of blockchain-based systems for architecture design. In *Software Architecture (ICSA), 2017 IEEE International Conference on*, pages 243–252. IEEE, 2017.

[35] Zheng Yang and Hang Lei. Formal process virtual machine for smart contracts verification. *arXiv preprint arXiv:1805.00808*, 2018.

[36] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *Work Pap.–2016*, 2016.