2023 IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE) | 979-8-3503-5043-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/WETICE57085.2023.10477834

Formal Verification of Smart Contracts based on Model Checking: An Overview

Rim Ben Fekih [a]
ISITCom, ReDCAD Lab
University of Sousse
Hammam Sousse, Tunisia.
rim.benfekih@sofrecom.com

Mariam Lahami[b]
ENIS, ReDCAD Lab
University of Sfax
Sfax, Tunisia
mariam.lahami@redcad.org

Mohamed Jmaiel[b]
ENIS, ReDCAD Lab
University of Sfax
Sfax, Tunisia
Mohamed.jmaiel@redcad.org

Salma Bradai[b]
Orange Innovation Tunisia
Sofrecom
Tunis, Tunisia
salma.bradai@sofrecom.com

***Abstract*—Focusing on important features in blockchain applications, smart contracts are one of the most studied in the literature. Despite the trusted implementations that smart contracts offer, different security problems and vulnerabilities are rising during their development and execution. Trying to deal with such issues, different researches are proposed to give eventual solutions. Such studies focus on the verification of smart contracts and adopt different techniques. While various formal methods are considered significant effective to ensure the trustworthiness and correctness of smart contracts, this work deals with formal verification of smart contracts using model checking. In this survey, we conduct an overview on smart contracts verification using model checking. We analyze and classify each study according to four main aspects; the adopted formalism, the verified properties, the system under verification and to which blockchain platform the contribution is dedicated. Finally, we suggest some promising future directions to stir research efforts into this area.**

***Index Terms*—Blockchain, Smart contracts, Formal verification, Model checking, Safety, Quick review**

I. Introduction

Among blockchain [1] features, smart contracts are one of the most appealing that facilitate, execute, and enforce predefined terms of coded contracts without intermediaries. Proposed in 1997 by cryptographer Szabo [2], smart contracts are defined as a digital agreement promised by contract participants. Hence, it is a piece of code that can be executed automatically on a computer. While the adoption of smart contracts on different blockchain platforms generates absolute confidence in their execution, Ethereum blockchain has led to tens of thousands of contracts holding millions of dollars in digital currencies. Although the use of blockchain presents several benefits essentially the development of secured and decentralized applications between several stakeholders, several vulnerabilities and bugs may occur within smart contracts. Malicious attackers may extract digital assets from a contract, cause damage by leading a smart contract into a deadlock, which prevents account owners from withdrawing or spending their assets [3]. Various attacks took place including the theft of Ether through the well-known attacks on the DAO[1] [4] as well as Parity Multisig Wallet [5]. These attacks led to Millions of dollars losses.

To overcome such issues, a multitude of researches has been done on the Verification and Validation (V&V) of smart contracts over the past several years. Several V&V techniques have been used such as theorem proving, model checking and testing. Motivated by the attention that researchers pay to model checking in order to enhance smart contracts' safety, we propose this paper to review the advances related to contracts correctness, revealing the adopted formalism and techniques. Therefore, we present in our work a quick review that surveyed the most relevant studies related to model checking of smart contracts dated from 2018. Particularly, we tackle the following main research questions:

- **RQ1**: What are the methodologies, approaches and tools based on model checking to verify smart contracts?
- **RQ2**: How the verification focus differs from one approach to another using the same formal method?

The answers to these questions help researchers to understand the studied topic, to identify the challenges in this research area and their solutions and also to discuss future directions. To do so, we first chose four well-known scientific and electronic databases (ScienceDirect Elsevier[2], ACM Digital Library[3], SpringerLink[4] and IEEE Xplore[5]) with the aim of extracting the most relevant papers related to our research topic. Second, we used the following search keywords which were the same in all databases: *"Model Checking AND Smart contract" OR "Model Checking AND Blockchain"*. Then, the selection of articles was performed by removing irrelevant articles after checking their titles and their abstracts and after being fully read we selected **26** as primary studies.

The rest of this paper is organized as follows. Section II provides key concept materials to understand the rest of the paper. Section III discusses some related work that apply formal verification for smart contracts. In Section IV, we provide an overview and a detailed analysis of 26 selected papers. At the end of this section, we will sum up the main

Identify applicable funding agency here. If none, delete this.
[a] Sofrecom, Tunisia.
[b] National School of Engineers of Sfax, Sfax University, Tunisia.
[1] Decentralized Autonomous Organization
[2] https://www.elsevier.com
[3] https://portal.acm.org
[4] https://www.springerlink.com
[5] https://www.ieee.org/web/publications/xplore/

results. In Section V, research challenges and opportunities are discussed. Finally, we summarize paper contributions, in Section VI, and we identify some future work.

II. Background

This section provides background materials to understand the rest of this paper.

A. Smart contracts

Smart contracts are self-executing code on the blockchain framework that allow for straight-through processing, which eliminates the need for manual intervention to execute transactions [6]. Once a smart contract is embedded in the blockchain, it becomes an autonomous agent that is permanently *tamper-proof*, as no one can change what's been programmed. In addition, smart contracts are *self-verifying* due to automated possibilities and *self-enforcing* when the rules are met at all stages. Besides the application of a specific code, smart contracts are used to encode the terms and conditions of an agreement into the transaction workflow. They are also capable of handling large numbers of virtual coins worth hundreds of dollars apiece, easily making financial incentives high enough to attract adversaries.

B. Smart contracts issues

In the following, we list the most well-known smart contracts vulnerabilities in the literature.

- **Reentrancy**: In a reentrancy attack, (a.k.a. a recursive call attack) a vulnerable contract calls into the calling contract before the first invocation of the function has finished. This could lead to undesired interactions between the different invocations of a contract's functions [7].
- **Unhandled Exceptions**: The developer is in charge of examining each call's outcome and handling any exceptions appropriately. However, many developers overlook or choose to disregard the handling of such exceptions, which prevents money from being transferred to their proper owners and leads to inconsistencies [8]–[12].
- **Transaction Order Dependency**: In Ethereum, a person who's running a node can predict which transactions will take place before they are completed, consequently the updates order. A contract's final state is therefore dependent on how the miner orders the transactions invoking it. Such contracts are called as transaction-ordering dependent contracts [10], [11], [13].
- **Callstack Depth Limitation**: When one smart contract calls another contract, the depth of the call stack will increase by one in the stack-based architecture used by EVM[6], which has a maximum depth of 1024. Attackers can take advantage of this functionality to purposefully exceed the stack length by calling themselves 1023 times before calling another contract to execute the attack [14].

In regard to such vulnerabilities, adopting a formal verification technique to ensure the safety of smart contracts is a crucial step. In this work, we are especially interested in model checking and studies that adopt this approach to verify Blockchain contracts.

C. Formal Verification based on Model Checking

Model checking is a well-established formal verification technique that investigates all possible system states in a brute-force manner [15]. A system model describes how the state of the system may change over time on the occurrence of transitions. It is typically expressed in terms of finite-state automata that describes all possible states, the initial state, and the possible one or more state transitions. The models are so-called abstractions that neglect irrelevant details for checking the desired properties of the system. When applied to a smart contract, the latter represents the main system, with an informal requirement to check: for example, we want our system to behave safely if we try to transfer an ownership of an NFT[7]. In order to ensure such requirement, the system will be modeled through a transition system, and the requirement is specified into a formal property. Then the model will be checked against the specified property by an existing model checker, which determines if the property is satisfied or violated.

III. Related Literature

We provide in this section an overview of the related surveys/reviews that deal with formal verification of smart contracts.

To this end, Bartoletti et al. [16] surveyed the formal models of Bitcoin contracts and compared the various languages and models for Bitcoin contracts. Authors aimed to recommend the right formal model for contracts programmers, based on the required expressiveness, usability, and suitability for verification. Furthermore, regarding legal smart contracts modeling, Ladleif and Weske [17] define a set of essential components for a fully specified legal contracts as a unifying model. Authors assessed eight existing smart contracts modeling languages and demonstrate the usability of the proposed model as a basis for a comprehensive comparison of the languages' expressiveness. Similarly, Singh et al. [18] conducted a systematic literature review (SLR) on 35 research study where authors address issues and vulnerabilities on formal verification and specification techniques as well as smart contracts languages. Similarly, Krichen et al. [19] reviewed the state-of-the-art on smart contracts specification and verification. Authors present the different modeling and specification formalism at both contract and program levels. Smart contracts limitations as well as those of formal verification are outlined. Otherwise, Imeri et al. [20] restricted their study by performing an analysis on model checking techniques of smart contracts at programming and execution levels. While authors' aim is to conduct a review on the available model checkers, they did include symbolic execution tools such as Oyente which performs a different verification approach compared to model

[6]Ethereum Virtual Machine

[7]Non-Fungible Token

checkers. Closer to the aim of this paper, Tolmach et al. [21] provide a systematic overview on modeling and specifications formalism applied for the purpose of verifying smart contracts, as well as the common trends and gaps in smart contract formal approaches.

Different from the discussed works, in this study we aim to cover the most of the established researches in regard to contracts verification using model checking technique. Furthermore, our analysis is more exhaustive in terms of specific aspects such as the covered system under verification and the considered application platforms.

IV. Formal Verification of Smart Contracts

A. Overview

We selected 26 papers published between 2018 and 2022 that use model checking to verify smart contracts [22]–[47]. Figure 1b presents the number of selected papers classified according to year of publication. The distribution shows that research in this field is somehow constant. Figure 1a illustrates the selected publications organized according to the used model checker. From 26 papers in total, 10 use NuSMV and nuXmv model checkers, while 3 papers use SPIN and BIP-SMC, 2 use CPN and the rest of studies use other verifiers such as Maude, MCMAS and Helena. Furthermore, we illustrate in Figure 1c the number of papers according to the used Blockchain platform. Ethereum is clearly the first choice in such studies given the important number of security incidents and vulnerabilities resulting from Solidity contracts.

B. Model Checking to resolve Smart Contracts Issues

This section respond to the first research question **RQ1** where we provide an analysis for each paper to reveal the different approaches used to ensure smart contracts safety.

a) Model checking of Bitcoin contracts: We identified 2 out of 26 papers that ensure the verification of Bitcoin smart contracts [32], [40]. Atzei et al. [32] develop a toolchain that evaluates properties on BitML smart contracts. Authors reason about liquidity to ensure that no funds still frozen forever. To do so, LTL formulas are specified and checked. A use cases benchmark is used for evaluation, where representative contracts are included such as lotteries and financial contracts. Similarly, Fedotov et al. [40] present an approach based on statistical model checking that prevents three real-life attacks, respectively, *consensus delay*, *DNS attack* and *double-spending with memory pool flooding*.

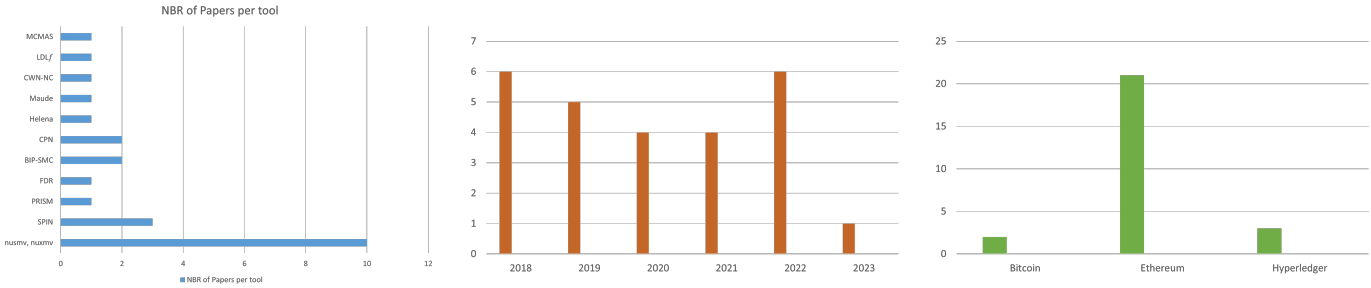
b) Model checking of Ethereum contracts: Ethereum is, until now, the most used and popular blockchain platform, we found that 21 out of 26 papers of the selected papers verify Ethereum contracts. Hence, multiple works verify smart contracts by deriving a PROMELA[8] model from a solidity contract and used the SPIN[9] tool to verify whether the contract logic was correct or not [22], [23], [35]. Thus, Bai et al. [22] verify safety properties by providing a formal smart contract template. Molina et al. [23] check the formal contractual model of a smart contract for data selling against frequent smart contracts problems like clause duplication. In [35], authors introduce a tool-chain for checking the compliance of smart contracts coding with its specification in order to enhance the system's safety and liveness.

Similarly, safety and liveness properties of Ethereum contracts are checked by multiple researches using NuSMV and nuXmv model checkers [25], [28], [29], [33], [39], [44], [46], [47]. Thus, Nehai et al. [25] outline an approach that verifies the implementation compliance of an Ethereum application with its respective predefined specification. Authors formalize the specification by a set of temporal logic propositions. The approach is therefore illustrated through an energy market case study. On the other hand, Kongmanee et al. [29] model and verify smart contracts and mitigate the impact of the state explosion problem by applying abstraction techniques to contracts. Another promising study is [28], where Mavridou et al. introduce VERISOLID, a framework that ensures the correctness of a single smart contract at design-time. Authors bring extensions to VERISOLID in [33] to verify the development and deployment of interacting smart contracts. VERISOLID is able to reason about all possible behavior of EVM bytecode, since it supports a complete formal semantics of EVM. Lotfi et al. [39] propose a verification approach for Solidity contracts. Authors propose a detailed design for EVM operations such as gas, memory and storage. Similarly, Al Shorman et al. [44] identify smart contracts vulnerabilities through the modeling and verification of interacting contracts behavior. Authors specify fairness, safety and liveness properties to recognize contracts issues. Furthermore, Fekih et al. [46] propose a verification approach of a single smart contract and takes into consideration the blockchain environment. Execution and functional properties are checked through CTL formulas.

Another interesting approach introduced in [24] where Abdellatif et al. suggest a novel formal modeling strategy to validate Ethereum smart contracts within their execution context. The smart contract for name registration built on the Ethereum platform has been turned concrete by the authors using this formalism. On the basis of the simulated executions, the authors identified smart contract vulnerabilities and suggested alternative approaches to fix them. Qu et al. [26] checked Ethereum smart contracts vulnerabilities, especially from the perspective of concurrency using the Communicating Sequence Processes (CSP) theory and Failure Divergence Refinement (FDR) model checker. Other contributions proposed in [31], [36], include the security analysis of Solidity contracts using Colored Petri Nets (CPN); a language for systems' design, specification and simulation. Liu et al. [31] utilize ASK-CTL, a branch timing logic to detect contracts vulnerabilities. While the authors of [36] make their approach more realistic by including a model of the EVM execution process that takes gas usage into consideration. Stephens et al. [37] introduce SmartPulse, a smart contracts verification tool. The authors develop a linear temporal logic (LTL) language called SmartLTL for expressing temporal safety features in smart

[8]Process Meta Language

[9]Simple Promela Interpreter



(a) Papers percentage using model checker. (b) Papers distribution by year. (c) Papers distribution by platform.

Fig. 1: Overview on retrieved papers

contracts and enforcing them with the SmartPulse verifier. As an input, SmartPulse take a set of manual specifications in addition to a smart contract. In the same context, Mazurek et al. [38] performs an automatic formal verification of smart contracts through EthVer. Authors formally define and model smart contracts as Markov decision processes (MDP) and use the PRISM model checker for verification. Nam et al. [41] propose a novel verification technique for a multi-agent system using ATL[10] model checking. Authors capture and validate game properties between smart contracts and potential users. Garfatta et al. [42] extend their work proposed in [48] where they introduce a CPN-based formal verification method for Solidity smart contract. First, authors convert the Solidity smart contracts to CPN, Then they analyze and verify this model by checking temporal-based properties. In [42] authors take into consideration the verification of function calls as well as the definition of smart contracts correctness properties as LTL formulas. On the other hand, Crincoli et al. [43] propose an automatic approach to identify a smart contract's vulnerability. The proposed approach models smart contracts as automata and applies μ-Calculus rules to identify *Front running, Bad randomness, Short addresses and Other* vulnerabilities. Bao et al. [45] introduce mcVer, a verification tool for Solidity contracts. McVer enables the automatic conversion of a Solidity programming language subset to the VERDS[11] modeling language.

c) Model checking of Hyperledger contracts: Approaches proposed in [27], [30], [34] reason about the correctness of Hyperledger Fabric contracts (aka. chaincodes). Hence, Sato et al. [27] propose a domain specific language based on Linear Dynamic Logic (LDLf) properties for chaincodes implementation. Instead of encoding the state-transition model in a separate modeling language like Promela, LDLf is utilized directly to represent a smart contract due to its greater expressiveness as compared to LTL. Similarly, Madl et al. [30] use interface automata to validate the semantics of loyalty point marketplace smart contracts. Instead of exchanging official currencies, buyers and sellers can put bids on the marketplace and swap loyalty points. Madl et al. use several relevant interface automata encoded directly in the nuXmv model checking language to model the market, and they specify several reachability conditions to check whether a deal is possible. In the same context, Alqahtani et al. [34] deal with the formalization of interacting smart contracts in order to verify their conformance with specifications. The suggested approach involves translating smart contracts into finite state machines, identifying contract relationships using the Behavior Interaction Priority (BIP) model, and then converting state machines to nuXmv models using the BIP-to-nuXmv tool. The study encodes temporal property requirements and verifies them against nuXmv models.

To answer the research question **RQ2**, and as summarized in Table I, the selected papers propose different approaches to verify smart contracts while relying on the same verification technique. We notice that multiple variants of model checking are used by the selected studies such as statistical [24], [40], symbolic [28], [33] and probabilistic model checking [38]. In addition, it is clear that the majority of studies use NuSMV and nuXmv model checkers to ensure the functional correctness of the system through temporal logic. Furthermore, the system under verification vary depending on the research; multiple studies verify a single smart contract while others consider contracts interactions and blockchain environment behavior. Finally, while properties specifications are mainly formalized into LTL and CTL, extensions of them are also used to overcome incompleteness in informal system specifications.

V. Challenges and Opportunities

Based on the studied papers, we identified several limitations regarding smart contracts' verification using model checking as follows.

- Model checking limitations: Model checking methods can only be properly applied to finite-state systems, which is a negative aspect. In addition, an important issue in model checking is the state-explosion problem. The complex nature of the CTL model checking algorithm makes it clear that the size of the state space has a significant impact on the algorithm's practical utility. In essence, the complexity of the verification process increases with the number of states, potentially making the technique

[10]Alternating-time Temporal Logic

[11]A model checker tool available on, *https://lcs.ios.ac.cn/ zwh/verds/*

TABLE I: Formal Verification of Smart Contracts using Model Checking

Ref	Year	Verifier	Model Formalism	Prop. Spec.	Verified properties	SUV	Platform	Automated
[22]	2018	SPIN	Promela model	LTL	Safety, Liveness	Single	Ξ	-
[23]	2018	SPIN	epromela model	LTL	Safety, Liveness, contract compliance	Interacting Sc + env	Ξ	-
[24]	2018	BIP-SMC	Timed automata	PB-LTL	Safety	Single + env	Ξ	-
[25]	2018	NuSMV	Transition system	CTL	Liveness	Interacting SCs + env	Ξ	No
[26]	2018	FDR	Process algebra (CSP)	Path-level patterns	Concurrency	Interacting contracts	Ξ	No
[27]	2018	LDL_f	-	LDL formula	Safety	Single	H	Yes
[28]	2019	nuXmv-BIP	Transition system	CTL	Safety, Liveness	Single	Ξ	-
[29]	2019	NuSMV	Transition system	CTL	Safety, Liveness	Single SC	Ξ	No
[30]	2019	NuSMV	Interface automata	CTL	Liveness	Single SC + Interacting users	H	-
[31]	2019	CPN	Transition system	ASK-CTL	Liveness, safety, boundedness	Single SC	Ξ	N/A
[32]	2019	Maude	Process Algebra	LTL	Liquidity	Single	₿	Yes
[33]	2020	nuXmv, BIP	Transition system	CTL	Safety & Liveness	Interacting SC	Ξ	Yes
[34]	2020	NuSMV, BIP-to-NuSMV	Transition System	LTL	Safety, Liveness	Interacting SC	H	Semi
[35]	2020	SPIN	Promela	LTL	Liveness	interacting contracts + env	Ξ	Yes
[36]	2020	CPN	Transition system, AST, CFG	ASK-CTL	Security	Single SC	Ξ	Yes
[37]	2021	UltimateAutomizer	Büchi automaton	SMARTLTL	Safety, Liveness	Interacting contracts + env	Ξ	Yes
[38]	2021	PRISM	Markov decision process	Probabilistic temporal logics	Correctness, Security	Interaction + exec env.	Ξ	Yes
[39]	2021	nuXmv	Transition system	LTL	Safety	Single + env	Ξ	Yes
[40]	2021	BIP-SMC	Finite state automata	Metric Temporal Logic	Security	interacting SC + env	₿	No
[41]	2022	MCMAS	Transition system	ATL	Safety	interacting contracts	Ξ	No
[42]	2022	Helena	Colored Petri nets	LTL	Safety	Interacting contracts	Ξ	No
[43]	2022	CWN-NC	Automata	μ-Calculus	Security	Single	Ξ	Yes
[44]	2022	NuSMV	Transition system	CTL	Safety	Interacting contracts	Ξ	No
[45]	2022	VERDS	Transition system	CTL	Safety & liveness	intercating contracts + env	Ξ	Yes
[46]	2022	nuXmv	Transition system	CTL	Safety & liveness	Single + env	Ξ	No
[47]	2023	nuXmv	Extended FSM	CTL	Safety & liveness	Single	Ξ	Semi

Ξ: Prop. Spec.: Properties specification, SUV: System Under Verification, Ethereum, ₿: Bitcoin, H: Hyperledger Fabric

useless if a simpler definition or an abstraction is not considered when applied to smart contracts.

- Soundness: Due to commonly rely on model abstraction to make model checking feasible, inaccuracies are added to the verification process, making it in some cases invalid. Thus, the majority of proposed studies are unsound.
- A minority verify private smart contracts: based on the selected studies, private contracts get less attention compared to public ones. Such limit is justified by the fact that public blockchains, such as Ethereum, are among the first platforms that support smart contracts and, therefore, have a bigger community. In addition, formal methods are used with the aim to prevent and detect code flaws, which are more common in public Blockchains.
- ERC-based contracts are neglected: ERC-based contracts are Ethereum contracts that inherit from predefined standards. For example, the management of NFTs is done through a Solidity smart contract that implements ERC-721 interface. Due to the important losses that occurred recently, these kinds of contracts must have special attention to ensure a safe handling of the pre-encoded terms.

VI. Conclusion

Despite the growing popularity of smart contracts, ensuring that contracts meet particular requirements still a serious issue. Within this survey, we presented a comprehensive analysis of 26 research papers that use model checking to confirm that smart contracts code behaves as specified. We classified each paper according to the used modeling and specification formalism, the system under verification and the Blockchain platform. Furthermore, we outlined the specified properties to reason about the safety and liveness of smart contracts.

For future work, we plan to (1) study and analyze papers that use different formal methods such as theorem proving, program and runtime verification and (2) investigate efforts towards ERC-based smart contracts.

References

[1] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] N. Szabo, "Formalizing and securing relationships on public networks," *First monday*, 1997.

[3] X. Feng, Q. Wang, X. Zhu, and S. Wen, "Bug searching in smart contract," *arXiv preprint arXiv:1905.00799*, 2019.

[4] M. I. Mehar, C. L. Shier, A. Giambattista, E. Gong, G. Fletcher, R. Sanayhie, H. M. Kim, and M. Laskowski, "Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack," *Journal of Cases on Information Technology (JCIT)*, vol. 21, no. 1, 2019.

[5] L. Breidenbach, P. Daian, A. Juels, and E. G. Sirer, "An in-depth look at the parity multisig bug," *URL: http://hackingdistributed.com/2017/07/22/deep-dive-parity-bug/(visited on 09/03/2018)*, 2017.

[6] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, 2014.

[7] "Swc-107: Reentrancy," 2020, accessed: 19-07-2023. [Online]. Available: https://swcregistry.io/docs/SWC-107

[8] C. Ferreira Torres, A. K. Iannillo, A. Gervais, and R. State, "The eye of horus: Spotting and analyzing attacks on ethereum smart contracts," in *25th International Conference of Financial Cryptography and Data Security: FC 2021, Virtual Event, Revised Selected Papers, Part I 25*, 2021, pp. 33–52.

[9] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in *Proceedings of the 1st international workshop on emerging trends in software engineering for blockchain*, 2018, pp. 9–16.

[10] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, 2016, pp. 254–269.

[11] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts," in *25th Annual Network and Distributed System Security Symposium, NDSS*, 2018, pp. 18–21.

[12] L. Brent, A. Jurisevic, M. Kong, E. Liu, F. Gauthier, V. Gramoli, R. Holz, and B. Scholz, "Vandal: A scalable security analysis framework for smart contracts," *arXiv preprint arXiv:1809.03981*, 2018.

[13] "Smart contract weakness classification (swc)," 2020, accessed: 19-07-2023. [Online]. Available: https://swcregistry.io/

[14] P. Zhang, F. Xiao, and X. Luo, "A framework and dataset for bugs in ethereum smart contracts," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 139–150.

[15] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[16] M. Bartoletti and R. Zunino, "Formal models of bitcoin contracts: A survey," *Frontiers in Blockchain*, p. 8, 2019.

[17] J. Ladleif and M. Weske, "A unifying model of legal smart contracts," in *International Conference on Conceptual Modeling*. Springer, 2019.

[18] A. Singh, R. M. Parizi, Q. Zhang, K.-K. R. Choo, and A. Dehghantanha, "Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities," *Computers & Security*, 2020.

[19] M. Krichen, M. Lahami, and Q. A. Al-Haija, "Formal methods for the verification of smart contracts: A review," in *15th International Conference on Security of Information and Networks (SIN)*, 2022.

[20] A. Imeri, N. Agoulmine, and D. Khadraoui, "Smart contract modeling and verification techniques: A survey," in *8th International Workshop on ADVANCEs in ICT Infrastructures and Services*, 2020, pp. 1–8.

[21] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, "A survey of smart contract formal specification and verification," *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–38, 2021.

[22] X. Bai, Z. Cheng, Z. Duan, and K. Hu, "Formal modeling and verification of smart contracts," in *Proceedings of the 2018 7th International Conference on Software and Computer Applications*. ACM, 2018.

[23] C. Molina-Jimenez, I. Sfyrakis, E. Solaiman, I. Ng, M. W. Wong, A. Chun, and J. Crowcroft, "Implementation of smart contracts using hybrid architectures with on and off–blockchain components," in *IEEE 8th International Symposium on Cloud and Service Computing (SC2)*, 2018, pp. 83–90.

[24] T. Abdellatif and K.-L. Brousmiche, "Formal verification of smart contracts based on users and blockchain behaviors models," in *9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2018.

[25] Z. Nehai, P.-Y. Piriou, and F. Daumas, "Model-checking of smart contracts," in *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 980–987.

[26] M. Qu, X. Huang, X. Chen, Y. Wang, X. Ma, and D. Liu, "Formal verification of smart contracts from the perspective of concurrency," in *International Conference on Smart Blockchain*. Springer, 2018.

[27] N. Sato, T. Tateishi, and S. Amano, "Formal requirement enforcement on smart contracts based on linear dynamic logic," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 945–954.

[28] A. Mavridou, A. Laszka, E. Stachtiari, and A. Dubey, "Verisolid: Correct-by-design smart contracts for ethereum," *arXiv preprint arXiv:1901.01292*, 2019.

[29] J. Kongmanee, P. Kijsanayothin, and R. Hewett, "Securing smart contracts in blockchain," in *34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*. IEEE, 2019.

[30] G. Madl, L. Bathen, G. Flores, and D. Jadav, "Formal verification of smart contracts using interface automata," in *IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2019, pp. 556–563.

[31] Z. Liu and J. Liu, "Formal verification of blockchain smart contract based on colored petri net models," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2019.

[32] N. Atzei, M. Bartoletti, S. Lande, N. Yoshida, and R. Zunino, "Developing secure bitcoin contracts with bitml," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[33] K. Nelaturu, A. Mavridou, A. Veneris, and A. Laszka, "Verified development and deployment of multiple interacting smart contracts with verisolid," in *Proceedings of the 2nd IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020.

[34] S. Alqahtani, X. He, R. Gamble, and P. Mauricio, "Formal verification of functional requirements for smart contract compositions in supply chain management systems," 2020.

[35] T. Osterland and T. Rose, "Model checking smart contracts for ethereum," *Pervasive and Mobile Computing*, 2020.

[36] W. Duo, H. Xin, and M. Xiaofeng, "Formal analysis of smart contract based on colored petri nets," *IEEE Intelligent Systems*, vol. 35, no. 3, pp. 19–30, 2020.

[37] J. Stephens, K. Ferles, B. Mariano, S. Lahiri, and I. Dillig, "Smartpulse: automated checking of temporal properties in smart contracts," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.

[38] Ł. Mazurek, "Ethver: Formal verification of randomized ethereum smart contracts," in *International Conference on Financial Cryptography and Data Security*. Springer, 2021, pp. 364–380.

[39] A. Lotfi Takami, "A reduction from smart contract verification to model checking," Master's thesis, University of Waterloo, 2021.

[40] I. Fedotov and A. Khritankov, "Statistical model checking of common attack scenarios on blockchain," *arXiv preprint arXiv:2109.02803*, 2021.

[41] W. Nam and H. Kil, "Formal verification of blockchain smart contracts via atl model checking," *IEEE Access*, 2022.

[42] I. Garfatta, K. Klai, M. Graïet, and W. Gaaloul, "Model checking of vulnerabilities in smart contracts: a solidity-to-cpn approach," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 316–325.

[43] G. Crincoli, G. Iadarola, P. E. La Rocca, F. Martinelli, F. Mercaldo, and A. Santone, "Vulnerable smart contract detection by means of model checking," in *Proceedings of the Fourth ACM International Symposium on Blockchain and Secure Critical Infrastructure*, 2022, pp. 3–10.

[44] A. Alshorman, K. E. Sabri, and M. A. AbuShariah, "Formalizing and verifying the behaviors of interacting smart contracts using model checking," *Available at SSRN 4048956*, 2022.

[45] Y. Bao, X.-Y. Zhu, W. Zhang, W. Shen, P. Sun, and Y. Zhao, "On verification of smart contracts via model checking," in *International Symposium on Theoretical Aspects of Software Engineering*. Springer, 2022, pp. 92–112.

[46] R. Ben Fekih, M. Lahami, M. Jmaiel, A. B. Ali, and P. Genestier, "Towards model checking approach for smart contract validation in the eip-1559 ethereum," in *Proceeding of IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2022.

[47] R. B. Fekih, M. Lahami, M. Jmaiel, and S. Bradai, "Formal modeling and verification of erc smart contracts: Application to nft," in *2023 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2023.

[48] I. Garfatta, K. Klai, M. Graïet, and W. Gaaloul, "A solidity-to-cpn approach towards formal verification of smart contracts," in *IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2021, pp. 69–74.