Formal Methods for the Verification of Smart Contracts: A Review

Moez Krichen
Faculty of CSIT
Al-Baha University
Al-Baha, Saudi Arabia
moez.krichen@redcad.org

Mariam Lahami
ReDCAD Laboratory
National School of Engineers of Sfax, Sfax Universiy
Sfax, Tunisia
mariam.lahami@redcad.org

Qasem Abu Al-Haija
Computer Science Department
Princess Sumaya University For Technology
Jordan
q.abualhaija@psut.edu.jo

***Abstract*—Smart contracts are digital contracts that rely on Blockchain technology to make their terms and execution conditions unforgeable. The purpose of a smart contract is to eliminate the need for a middleman in business and trade between anonymous and identified participants. Since 2016, smart contracts have been gaining traction in various areas, including public management, supply chain, energy, finance, communication, and healthcare. Anyone who interacts with a smart contract after it is launched on the blockchain system will be in danger if it contains vulnerabilities or faults. Therefore, the use of formal methods which are mathematical techniques for modeling, designing, and testing software and hardware systems to ensure they are constructed correctly, is highly required. In this paper, the applied state-of-the-art formal methods on smart contracts specification and verification have been reviewed with the aim of minimizing the risk of faults and bugs occurrence and avoiding possible resulting costs. Also, we have discussed several challenges and future research guidelines related to this emerging research topic.**

***Index Terms*—Formal Methods,Blockchain, Smart Contracts, Verification, Review.**

I. Introduction

Smart Contracts are the technologies that will shape how humans interact in the future, and they have already begun to explode in popularity. A smart contract is a computer code that executes all or parts of the contract automatically and is stored on a blockchain-based platform. The code can either be the only representation of the parties' agreement, or it can supplement a typical text-based contract by carrying out certain clauses, such as money transfers from Party X to Party Y. Because the code is duplicated over several nodes of a blockchain, it benefits from the blockchain's permanence, security, and immutability. Smart contracts are now best adapted to automate two types of "transactions" featured in many contracts: (1) assuring the payment of monies upon the occurrence of specified triggering events and (2) applying monetary penalties if certain objective conditions are not met. Human intervention, such as through a trusted escrow holder or even the judicial system, is not necessary once the smart contract has been launched and is operational in each case, lowering the contracting process' execution and enforcing costs. Smart contracts, for example, might reduce so-called procure-to-pay gaps. When a product is delivered to a warehouse and scanned, a smart contract might immediately request the necessary approvals and, once granted, send cash from the buyer to the seller. Sellers would get paid faster and no longer need to engage in dunning, while purchasers' account payable fees would be reduced. This could influence working capital needs and make financial operations easier for both parties. On the enforcement side, a smart contract might be configured to disable access to an internet-connected item if payment is not received. For instance, access to certain content may be automatically prohibited if payment is not received.

Scientists from the two universities NUS[1] and UCL[2] found that 34,200 out of roughly one million Ethereum smart contracts are vulnerable to attack, with 2,365 of them taking only 10 seconds to hack using a tool they developed called MAIAN [31]. At the time of the investigation, their analysis revealed that over $13.8 million worth of ether was at risk. Because such risks and even slight flaws might have catastrophic and permanent implications, working with smart contracts necessitates extreme vigilance. The extent to which a software system accomplishes the objective for which it was designed is the fundamental measure of success in today's commercial context. On the other hand, a more competitive market frequently expects higher quality, faster turnaround times, and less expensive software. Many IT organizations struggle to provide high-quality products on schedule and within their budgetary constraints. The number of faults discovered during development significantly impacts the software metrics discussed above. The project budget can be reduced if an issue is recognized early in the program development process.

When software developers discover a mistake made during the requirements phase during testing, they must correct the wrong requirements, check all consequences through design and implementation, and then retest the product. Cost-effective solutions that handle main risks and give visible evidence of trustworthiness are necessary to produce a certain software product and solve the budget overrun problem (due to inac-

[1]NUS=National University of Singapore
[2]UCL=University College London

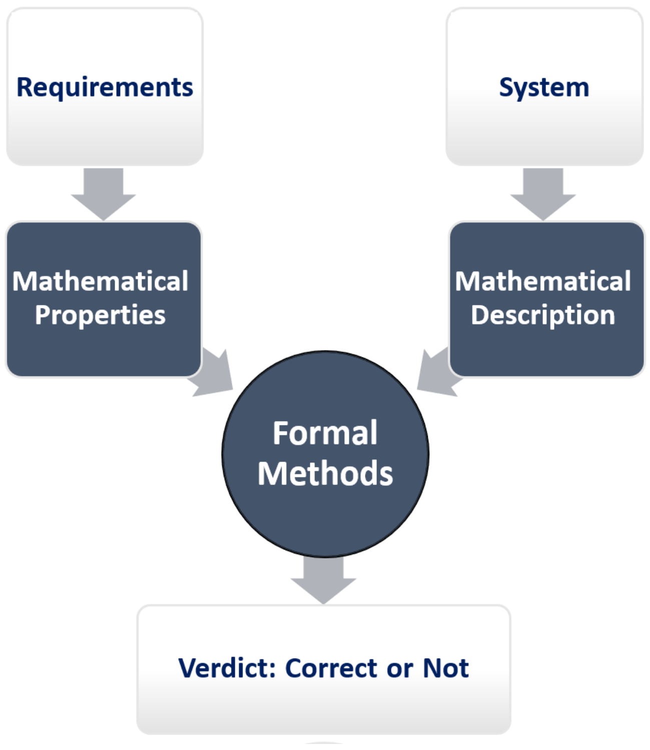


Fig. 1. A simplified illustration of how Formals Methods work.

curacies in requirement specifications). The issues mentioned above can be solved using formal methods [8]. Formal methods [35] are mathematical approaches used to specify, build, and verify software and hardware systems. A formal specification language refers to the mathematical representation utilized in formal techniques. As illustrated in Figure 1, both the system under investigation and the set of interest requirements are translated into appropriate formal specification languages. Then, the adopted formal techniques are in charge of checking whether the mathematical description of the system is correct concerning the obtained mathematical properties or not. These techniques are indeed useful at various stages of the development process of a wide range of systems [32], and they are now making their way into industry standards.

In this direction, several efforts were made to help developers in creating correct and secure smart contracts by using formal methods. Researchers have adopted different approaches, including theorem proving [2], [19], model checking [1], [5], [27], [30], and software testing [3], [23]. In this work, our goal is to conduct a review that deals with applying formal methods to check the correctness of smart contracts. Also, we identified several challenges and future research guidelines related to this research topic.

This paper is structured as follows. Section 2 presents preliminaries about smart contracts and blockchain. Subsequently, a brief introduction to formal methods is given in Section 3. Then, several research approaches applying formal methods to verify and validate the correctness of smart contracts are discussed in Section 4. Section 5 lists limitations and challenges

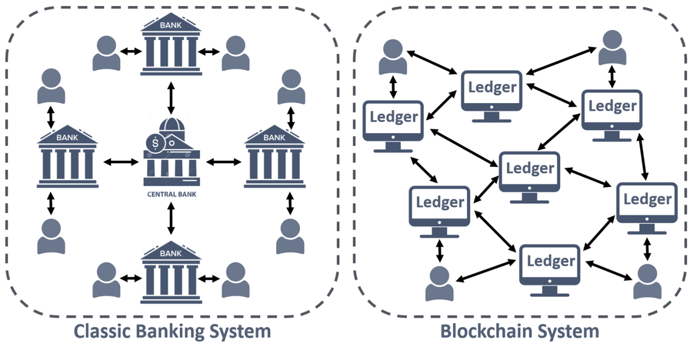


Fig. 2. Illustration of the difference between the classic banking system (left) and the blockchain system (right).

related to using smart contracts and formal methods. Finally, Section 6 summarizes paper contributions while identifying potential areas for future research.

II. Preliminaries about Smart Contracts

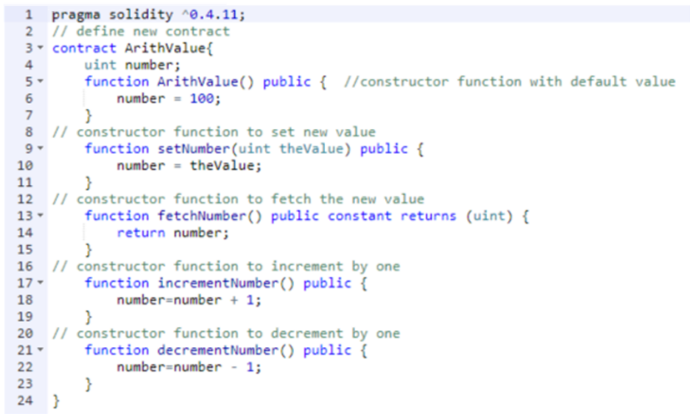
In this section, we propose a short overview of smart contracts. Because smart contracts are mostly built on top of blockchain technology, we begin with a brief overview of blockchain.

A. *What is Blockchain?*

A blockchain is a distributed database shared by computer network nodes. A blockchain functions as a database, storing data in digital form. Blockchain is well recognized for its vital role in maintaining a secure and decentralized record of transactions in cryptocurrency systems such as Bitcoin. The blockchain is unique in that it guarantees the accuracy and security of a data record while also creating trust without the need for a 3rd party. A blockchain's data structure differs from that of a traditional database. A blockchain organizes data into blocks, each of which includes data collection. When a block's storage capacity is reached, it is closed and linked to the previous block, resulting in a data chain known as the blockchain. All subsequent information incorporated after that newly added block is compiled into a new block, which is then incorporated into the chain once it is complete. A database arranges data into tables, whereas a blockchain, as the name implies, arranges data into chunks (blocks) that are strung together. This data structure produces an irreversible data chronology when implemented decentralized. When a block is completed, it becomes permanent and part of the timeline. A new block is added to the chain and assigned a time stamp.

B. *What is a Smart Contract?*

Smart contracts are programs run when certain criteria are met and stored on a blockchain. They are typically used to computerize the execution of a consensus so that all stakeholders can be certain of the outcome immediately, with no need for brokers or timewasting. They could also automate a workload, beginning the next step when some conditions are



```
pragma solidity ^0.4.11;
// define new contract
contract ArithValue{
    uint number;
    function ArithValue() public {  //constructor function with default value
        number = 100;
    }
// constructor function to set new value
    function setNumber(uint theValue) public {
        number = theValue;
    }
// constructor function to fetch the new value
    function fetchNumber() public constant returns (uint) {
        return number;
    }
// constructor function to increment by one
    function incrementNumber() public {
        number=number + 1;
    }
// constructor function to decrement by one
    function decrementNumber() public {
        number=number - 1;
    }
}
```

Fig. 3. An example of a smart contract written in Solidity [29].

met. To make smart contracts work, simple "if/when...then..." lines are written into code on a blockchain. When certain conditions are met and validated, the operations are carried out by a computer network. These operations may include transferring funds to the appropriate parties, registering a vehicle, sending alerts, or issuing tickets. The blockchain is updated once the transaction is completed. This indicates that the transaction cannot be changed and that the results are only accessible to those given permission.

A smart contract can include as many specifications as necessary to persuade participants that the job will be done correctly. To set the terms, participants should concur on how transactions and associated data are expressed on the blockchain, on the "if/when...then..." rules governing those transactions, inspect all possible exceptions, and design a framework for resolving disputes. A programmer could then program the smart contract. However, companies that use blockchain for business are growingly providing templates, web interfaces, and other online tools to simplify smart contract creation.

Smart contracts can be created using a variety of programming languages. Below, we list some examples:

- Solidity: Solidity is a statically typed, object-oriented programming language that was created to enable the creation of smart contracts.[3]
- Yul: Yul may be utilized as an intermediary language in the compilation for compiling Solidity-based Ethereum smart contracts.
- Rust: Rust is an ideal smart contract language because it lacks undefined behavior and is type and memory safe. It produces compact binaries.
- Vyper is a python-like programming language employed to produce Ethereum Virtual Machine-compatible smart contracts.
- Javascript: For example, Nebulas is a platform for creating smart contracts. It offers a method for creating Smart Contracts with JavaScript.

[3]The image displays the Solidity code for an increment and decrement operation. Additionally, it features a method that accepts input and another function that retrieves this number when the contract is executed.

C. *Examples of Use of Smart Contracts*

Smart contracts have numerous applications, including healthcare, supply chain, and financial sectors. Some examples are as follows:

- Financial sector: In a variety of ways, smart contracts assist in the transformation of traditional financial services. When it comes to insurance claims, they verify for errors, route them, and then send cash to the user if everything checks up. Smart contracts include essential bookkeeping capabilities and eliminate the chance of accounting records being tampered with. They also allow shareholders to participate in transparent decision-making. They also assist in trade clearing, the process of transferring payments after the amounts of trade settlements have been computed.
- Construction: Contracts are well-established in the building business. Everything is normally done through specified contracting procedures, from material delivery to cash transfer to performance measurement. Smart contracts can help the construction industry improve productivity and reduce payment processing times. Construction invoice processing can also be streamlined with smart contracts.
- Property ownership: Smart contracts may be used to track the ownership of a wide range of assets, including buildings, land, and even phones and watches. Smart contracts in the property market can eliminate the need for high-cost services such as those supplied by lawyers and real estate agents. Sellers will be able to manage the transaction entirely on their thanks to this new technology.
- Voting systems: Smart contracts create a secure voting environment that is less vulnerable to tampering. Votes in smart contracts would be ledger-protected, making them extremely difficult to interpret. Moreover, smart contracts can grow voter turnout, which has traditionally been low due to an ineffective system that needs voters to queue, ask For id, and fill out forms. Once voting is transmitted online using smart contracts, it can increase the number of respondents in a voting system.
- Healthcare: Highly sensitive data, including patient records, might be safely encrypted and shared between departments/research institutes using blockchain technology. On the other hand, medical research organizations have a massive amount of data to protect, including test findings and novel drug formulae. If they need to disclose any of this information to a third party, smart contracts might be used to protect it. One smart contract blockchain application, in particular, could be extremely beneficial to the medical research sector.

D. *Main Advantages of Smart Contracts*

The following are the most significant benefits of using smart contracts:

- Autonomy: The most significant advantage of smart contracts is their automation. In simple terms, there will be no interruptions, and no third parties will be able to change the agreement or decision. This automation can go a long way in assisting businesses in automating some areas of their operations.
- Security: Since blockchain transaction records are encrypted, they are very difficult to hack. In addition, since every record on a distributed ledger is linked to the next and previous entries, hackers would have to alter the entire chain to alter any record.
- Transparency and trust: Since there is no 3rd party involved and encrypted transaction logs are exchanged among participants, there is no need to be concerned about information being falsified for self-benefit.
- Backup: All documents kept on the blockchain are duplicated many times, allowing for the restoration of originals in the event of data loss.
- Efficiency, precision, and speed: The contract is swiftly carried out if a requirement is satisfied. Since smart contracts are computerized and automated, there is no paperwork to deal with and no time lost correcting mistakes when filling out paperwork by hand.
- Cost Reduction: Smart contracts do away with the need for intermediaries and all associated costs and delays.

E. Possible Bugs in Smart Contracts

Below, we present five common smart contract vulnerabilities [14]:

- Reentrancy: This occurs when a malicious smart contract is embedded in the victim smart contract, allowing the malicious contract to interfere with the first execution of the victim contract.
- Miner manipulation: In these types of flaws, a malicious miner manipulates variables outside of the smart contract's execution to manipulate the system to their advantage.
- Access control: Access control in smart contracts can be linked to governance and crucial logic such as token minting, withdrawing money, stopping and upgrading contracts, transferring ownership, voting on propositions, etc.
- Integer overflow/underflow: When arithmetic operations result in an output larger than the maximum size allocated to it, an overflow occurs; when the output is smaller than the minimum size, an underflow happens.
- Distributed Denial of Service: Smart contracts that have been maliciously designed can overload blockchain nodes with bandwidth. This would force the network's processing resources to be diverted to dealing with the overload, rendering the network unusable.

III. Formal Methods

A. Brief Overview

Formal methods are mathematical tools for modeling complex systems. They apply mathematics to software and hardware engineering to bring confidence to the design and testing of these systems. Before design, formal methods are used to describe a system's functions with descriptive languages, ensuring the system's functionality. Depending on the rigor with which the system is specified, formal approaches may be utilized in development. Formal specifications can serve as a roadmap for determining requirements. Formal analysis methods describe functions that can be used to verify a program. The degree to which formal procedures are used can differ. Only in the case of essential systems and complicated designs, when errors can result in costly redesigns or refabrication, is a complete implementation of formal methods judged prohibitively costly. Formal approaches are frequently used merely to describe and steer the development of the intended function. This is referred to as level 0 or lite formal approaches. It is considered level 1 when formal methods are also utilized to validate functions. The complete system is machine validated through all of its functions at Level 2, the highest level of formal procedures.

B. Different Types

There are different types of formal methods. Below, we list some of them:

- Model Checking: Model checking is a method for automatically confirming the correctness of finite-state systems. It refers to the algorithms for fully and automatically evaluating the state space of a transition system to verify whether a particular system model satisfies a given specification.
- Theorem Proving: Theorem proving is a formal technique that offers a symbolic logic proof. The deductive inference is used. Each phase of the proof will (a) present a premise or hypothesis; (b) use only legal inference rules to give a statement that follows logically from the outcomes that have already been established. These formal proofs are frequently drawn-out and tiresome. A large portion of the procedure can be automated using sophisticated software called theorem provers.
- Symbolic Execution: Symbolic execution is a technique for running programs abstractly, allowing for a single abstract execution to cover all of the program's potential inputs that follow the same execution path through the code. These inputs are handled symbolically throughout execution, and the outcome is stated in terms of constants that symbolize the input values.
- Runtime Verification: Runtime verification is a simple verification method that examines a program's attributes while operating. Runtime verification often offers a reactive defense against flaws or violations of correctness at runtime. It has the potential to reveal susceptible situations that model checking or symbolic execution could not possibly reach because of the state or path explosion.

C. Main Advantages

The following are the key benefits of using formal methods [8]:

- Abstraction: A formal specification is an abstract, precise, and in some ways complete description. The abstraction makes it simple for a human reader to grasp the big picture of the software product.
- Rigorous Analysis: Formal descriptions are created from several perspectives to determine important aspects such as the fulfillment of high-level criteria or the accuracy of a proposed design.
- Early Defect Discovery: Formal Methods can be applied to the earliest design artifacts, resulting in early design flaw detection and eradication.
- Correctness guarantees: Model checkers and other formal analysis tools analyze all potential execution pathways across the system. A model checker will detect any faults or errors that may exist. Testing will not be able to provide this degree of coverage.
- Reliability: In a highly regulated industry like aviation, formal techniques give the kind of evidence that is required. They show and provide evidence of the product's reliability.
- Efficient Test Scenarios: We can systematically build effective test cases directly from the formal specification. This is a low-cost method of generating test scenarios [20], [22].

IV. Formal Methods for Smart Contracts

Depending on the level of abstraction at which modeling and analysis are conducted, the formalization approaches for smart contracts can be divided into two major categories.

A. Program-Level

The program-level models aim to provide a white-box view of the target contracts. In this case, the adopted approaches primarily analyze the contract implementation and are thus platform-dependent. Such approaches enable precise reasoning about low-level details of the smart contract execution process. Below, we present four types of models that these program-level approaches may use:

- Control Flow Graph: A control-flow graph depicts every path that could be taken through a program while it is being executed using graph notation. Control-flow/Data-flow analyses and Symbolic Execution are two common approaches that may be performed on control flow graphs for conventional software applications. Some techniques rely on control flow graphs derived from smart contracts, while others are built from the ground up to accommodate specific language features of contract programs. The authors of [9], for instance, developed a tool called Ethainter, which uses information flow analysis to uncover composite vulnerabilities. Other tools accept extensions to control flow graphs with annotations, enabling the specification and validation of specific properties [36].
- Abstract Syntax Tree: An abstract syntax tree is a formal tree representation that may be used for reasoning about the syntax of a given program. Abstract syntax trees may enhance lightweight pre-analyses, maximizing the efficiency and scalability of heavier downstream investigations. For example, the authors of [11] proposed a formal analysis based on abstract syntax trees to check whether a smart contract complies with the ERC20 standard. In the same direction, the authors of [37] suggested a method for scanning the abstract syntax trees of smart contracts to look for library imports and potentially harmful code patterns. These purely syntactic methodologies have the obvious disadvantage of not always respecting the operational semantics of smart contracts, which compromises the validity and thoroughness of the analysis.

B. Contract-Level

In this case, smart contracts are seen as "black boxes" that accept outside transaction messages and may operate according to them. The contract-level models primarily focus on three elements: 1. Contracts (public functions, visible effects, etc.); 2. Blockchain State (environment variables, global variables, etc.); 3. Users (account address, balance, transactions, etc.). The interactions between smart contracts and the outside world are well captured by the contract-level models, which are typically stated in terms of:

- STS[4]: An STS is made up of states and transitions between states that can be labeled with symbols from a set; the same label can appear on multiple transitions. A smart contract's behaviors can be naturally interpreted as an STS. Indeed, some smart contract languages even use a state-oriented programming model, including Bamboo [5] and Obsidian [12]. Many state-transition formalisms are used in the literature for modeling and verifying smart contracts:
  - CG[6] A CG is a group of autonomous agents operating in a finite-state environment. Each player chooses an action to take throughout each round in a game with endless rounds. For instance, an atomic swap smart contract was regarded as a CG in [28]. To determine if a contract is fair and accurate given the provided user behaviors, the authors employed the MCK model checker. Similar objectives were pursued in [10]; however, the proposed analysis was founded on the CG model's objective function.
  - TA[7]: A TA is a finite graph with a finite number of locations and labeled edges that have been extended with real-valued variables. This graph type can be considered an abstract model of timed systems. For instance, the authors of [1] created TA models of the smart contract, the contract's users, and a compacted

[4]STS=State Transition System.

[5]GitHub - pirapira/bamboo: Bamboo see https://github.com/cornellblockchain/bamboo

[6]CG=Concurrent Game.

[7]TA=Timed Automaton.

blockchain system. To determine the likelihood of a successful assault on the contract, statistical model checking is used to examine the relations between these elements.

- PN[8]: A PN is a bipartite graph with two types of elements: places and transitions, which are represented as white circles and rectangles, respectively. A location can have an unlimited number of tokens represented as black circles. For instance, to analyze the security of a smart contract, the authors of [15], [25] used colored Petri nets and the cpntools toolkit. The analysis is carried out manually in [25] while it is carried out automatically in [15]. In the same context, the authors of [38] suggested a method for creating a smart contract from a model described as a PN process and automatically validating the model in accordance with common characteristics like the avoidance of dead transitions.[9]
- BIP[10]: BIP is a platform that generates executable code that can be used for simulating, executing, exploring, and debugging formal models. It includes execution engines, which act as runtime schedulers for the produced codes. Smart contract BIP models typically have two layers: the Behavior layer's components define each smart contract's logic as a finite state machine. In contrast, the Interaction layer specifies the communication rules between them. For instance, the authors of [27] developed a BIP-based framework called VeriSolid, which allows producing provably secure smart contracts.
- BPMN[11]: BPMN is a flow chart representing the steps of a planned business process from beginning to end. It is an essential component of Business Process Management because it graphically represents the detailed sequence of business activities and information flows required for completing a process. For instance, the authors of [13] suggested translating BPMN models, which outline collaborative business processes, to Solidity and deploying them on the blockchain. Translation to a formal language, such as PN, is one potential method of validating a BPMN model.
- UML[12]: UML is a graphical modeling language used in software engineering to offer a standardized method for visualizing system design. UML notations allow users to model control flow, access control, and asset management in the produced smart contracts. For instance, in [17], a UML state model was used to create a Solidity implementation of a cyber-physical system.

[8]PN=Petri Net.

[9]A dead transition is a transition that can never be executed.

[10]BIP= Behavior Interaction Priority.

[11]BPMN=Business Process Model and Notation.

[12]UML=Unified Modeling Language.

- PA[13]: A PA is a collection of mathematical techniques used to characterize the behaviors of parallel or distributed systems as interacting concurrent processes. A PA describes the interactions between one or more connected smart contracts and simultaneously acting users. By translating a smart contract into one of the process-algebraic formalisms, the validity of these interactions can be verified. For instance, the authors of [29] proposed a translation of Ethereum contract public functions from Solidity to process notations represented in an applied π-calculus SAPIC version.[14] Similarly, the authors of [33] proposed a translation of public functions from Solidity into CSP processes.[15] In addition, several domain-specific programming languages use process-algebraic models for a smart contract like BitML (Bitcoin Modelling Language) [6], [7] , Rholang (Reflective higher-order language) [16] and SCL [24].

Temporal logic is a branch of symbolic logic concerned with problems involving propositions with time-dependent truth values. Contract-level specifications are written in a variety of logics. The most common are Linear Temporal Logic (LTL) [34] and Computation Tree Logic (CTL) [26]. Other logics include Defeasible Logic [18] and Deontic Logic [4], which aid in the definition of smart contract parties' obligations and rights.

V. Limitations and Challenges

Below, we list the main limitations and challenges related to using smart contracts and formal methods.

A. *Limitations of Smart Contracts*

Smart contracts have five major characteristics that make them appealing targets for hackers:

- Low level of maturity: Smart contract development, deployment, testing, and security tools are still in their infancy. This creates an opportunity for developers to make mistakes, as well as opportunities for malicious actors.
- No strict legal consequences: Smart contracts are open to the public, and anyone can engage with them. This, paired with the anonymity of smart contracts, implies that hackers are less likely to face legal consequences for hacking them than for hacking other types of networks.
- Higher Payouts: Successful assaults on smart contracts frequently result in rewards of up to $10 million. This is especially true for assaults against wallets and exchanges that store a significant amount of a customer's money. Hackers are drawn to this because the possibility of obtaining even a portion of such big sums surpasses the rewards of a classical bank attack.

[13]PA=Process Algebra.

[14]SAPIC=Stateful Applied pi Calculus.

[15]CSP=Communicating Sequential Processes.

- Reusing existing tools: To cause harm, hackers do not need to be experts in smart contracts. They reuse existing tools to detect vulnerabilities and common banking Trojans to steal digital wallet secret keys.
- Difficult to identify: One is nothing more than an address on the blockchain. Identifying the person behind the address is difficult or impossible if the address is not associated with any other identifiable information.

B. Limitations of Formal Methods

In the software development lifecycle, formal approaches are critical. These approaches, however, have considerable limitations. These weaknesses hamper the usefulness of formal approaches for software products. The following are some of the shortcomings of formal methods:

- While rigorous descriptions promise to improve system reliability, design time, and comprehension; they come at the cost of a longer learning curve; the mathematical disciplines employed to formally describe computing systems are outside the scope of traditional engineering education. Furthermore, most formal methods' metamodels are often constrained to improve provability. There is a substantial compromise between the demand for rigor and the ability to simulate all behaviors.
- In general, actual user requirements may differ from what the user specifies, and they will most likely change over time. There is no way to guarantee the validity and completeness of a specification in terms of the user's everyday needs while utilizing formal methods. However, different ways to reduce the likelihood of inaccurate specifications exist in the literature, but all approaches must begin with an informal starting point. It is impossible to be certain that you have gathered all user needs correctly.
- To establish the program's correctness, a formal description of the program should include a description of how the program should work in conjunction with the used hardware and operating system. A formal description of the technical environment used in industrial software development is not accessible in most cases. The challenge is exacerbated by the fact that, depending on the formal approach chosen, such a formal description must have a highly specific shape.
- The nature of formal approaches is descriptive and analytical. They were not thought to be imaginative. There are only formal techniques for describing and analyzing designs in reality. No such thing as a formal design process exists. We must mix formal methods with other approaches to construct a practical system.
- Formal approaches are used to deal with software and its documentation. Other key aspects of software products, such as training, customer support, maintenance, and installation, must be handled independently. Together, these components and their quality create a high-grade product. Formal approaches have no bearing on the quality of software products. As a result, most successful software product providers must devote significant time and attention to addressing all essential components of a software product.

VI. Conclusion and Future Directions

A smart contract is an automated self-executing transaction code that conducts the agreement conditions of a contract in programmable blockchains. The main purpose of a smart contract is to eradicate the need for a human agent in business and trade between anonymous and identified participants. As smart contracts are computerized self-reliant techniques that are deployed to blockchains to implement the agreement between buyer and seller, it is automatically verified and executed via a computer network. Therefore, several formal techniques have been proposed in the literature to provide a means for verifying and validating the correctness of the evolving smart contracts. Hence, this paper systematically investigates the existing formal methods utilized for the emerging smart contract specification and verification field to minimize the risk of faults and bugs and avoid possible costs. As future recommendations, we recommend the following future research directions for the verification of smart contracts using formal methods are the following:

- Lowering the cost and the execution time of formal approaches in various verification phases [21].
- Integrating multiple mathematical concepts and modeling languages using abstraction techniques.
- Making formal methods easier and more accessible to software developers and testers.
- Possessing reusable and parameterized models and theories rather than defining models and theories from scratch each time.
- Using modern super computing, cloud computing, and fog computing resources to speed up calculation and shorten the duration of the entire verification process.

References

[1] Abdellatif, T., Brousmiche, K.L.: 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). pp. 1–5. IEEE (2018)

[2] Amani, S., Bégel, M., Bortin, M., Staples, M.: Towards verifying ethereum smart contract bytecode in isabelle/hol. In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 66–77. ACM (2018)

[3] Antonino, P., Roscoe, A.: Formalising and verifying smart contracts with solidifier: a bounded model checker for solidity. arXiv preprint arXiv:2002.02710 (2020)

[4] Azzopardi, S., Pace, G.J., Schapachnik, F.: On observing contracts: deontic contracts meet smart contracts. In: Legal Knowledge and Information Systems, pp. 21–30. IOS Press (2018)

[5] Bai, X., Cheng, Z., Duan, Z., Hu, K.: Formal modeling and verification of smart contracts. In: Proceedings of the 2018 7th International Conference on Software and Computer Applications. pp. 322–326. ACM (2018)

[6] Bartoletti, M., Zunino, R.: Bitml: a calculus for bitcoin smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 83–100 (2018)

[7] Bartoletti, M., Zunino, R.: Verifying liquidity of bitcoin contracts. In: International Conference on Principles of Security and Trust. pp. 222–247. Springer (2019)

[8] Batra, M.: Formal methods: Benefits, challenges and future direction. Journal of Global Research in Computer Science **4**(5), 21–25 (2013)

[9] Brent, L., Grech, N., Lagouvardos, S., Scholz, B., Smaragdakis, Y.: Ethainter: a smart contract security analyzer for composite vulnerabilities. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 454–469 (2020)

[10] Chatterjee, K., Goharshady, A.K., Velner, Y.: Quantitative analysis of smart contracts. In: European Symposium on Programming. pp. 739–767. Springer, Cham (2018)

[11] Chen, J., Xia, X., Lo, D., Grundy, J.: Why do smart contracts self-destruct? investigating the selfdestruct function on ethereum. ACM Transactions on Software Engineering and Methodology (TOSEM) **31**(2), 1–37 (2021)

[12] Coblenz, M., Oei, R., Etzel, T., Koronkevich, P., Baker, M., Bloem, Y., Myers, B.A., Sunshine, J., Aldrich, J.: Obsidian: Typestate and assets for safer blockchain programming. ACM Transactions on Programming Languages and Systems (TOPLAS) **42**(3), 1–82 (2020)

[13] Di Ciccio, C., Cecconi, A., Dumas, M., García-Bañuelos, L., López-Pintado, O., Lu, Q., Mendling, J., Ponomarev, A., Binh Tran, A., Weber, I.: Blockchain support for collaborative business processes. Informatik Spektrum **42**(3), 182–190 (2019)

[14] Dingman, W., Cohen, A., Ferrara, N., Lynch, A., Jasinski, P., Black, P.E., Deng, L.: Defects and vulnerabilities in smart contracts, a classification using the nist bugs framework. International Journal of Networked and Distributed Computing **7**(3), 121–132 (2019)

[15] Duo, W., Xin, H., Xiaofeng, M.: Formal analysis of smart contract based on colored petri nets. IEEE Intelligent Systems **35**(3), 19–30 (2020)

[16] Eykholt, E., Meredith, L.G., Denman, J.: Rchain architecture documentation. Retrieve. Jan **19**, 2019 (2017)

[17] Garamvölgyi, P., Kocsis, I., Gehl, B., Klenik, A.: Towards model-driven engineering of smart contracts for cyber-physical systems. In: 2018 48th annual IEEE/IFIP international conference on dependable systems and networks workshops (DSN-W). pp. 134–139. IEEE (2018)

[18] Governatori, G., Idelberger, F., Milosevic, Z., Riveret, R., Sartor, G., Xu, X.: On legal contracts, imperative and declarative smart contracts, and blockchain systems. Artificial Intelligence and Law **26**(4), 377–409 (2018)

[19] Hirai, Y.: Defining the ethereum virtual machine for interactive theorem provers. In: International Conference on Financial Cryptography and Data Security. pp. 520–535. Springer (2017)

[20] Krichen, M.: Contributions to model-based testing of dynamic and distributed real-time systems. Ph.D. thesis, École Nationale d'Ingénieurs de Sfax (Tunisie) (2018)

[21] Krichen, M.: Improving formal verification and testing techniques for internet of things and smart cities. Mobile Networks and Applications pp. 1–12 (2019)

[22] Krichen, M., Mechti, S., Alroobaea, R., Said, E., Singh, P., Khalaf, O.I., Masud, M.: A formal testing model for operating room control system using internet of things. Computers, Materials & Continua **66**(3), 2997–3011 (2021)

[23] Lahami, M., Maâlej, A.J., Krichen, M., Hammami, M.A.: A comprehensive review of testing blockchain oriented software. In: Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2022, Online Streaming, April 25-26, 2022. pp. 355–362. SCITEPRESS (2022)

[24] Laneve, C., Coen, C.S., Veschetti, A.: On the prediction of smart contracts' behaviours. In: From Software Engineering to Formal Methods and Tools, and Back, pp. 397–415. Springer (2019)

[25] Liu, Z., Liu, J.: 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, pp. 555–560. IEEE (2019)

[26] Madl, G., Bathen, L., Flores, G., Jadav, D.: Formal verification of smart contracts using interface automata. In: 2019 IEEE International Conference on Blockchain (Blockchain). pp. 556–563. IEEE (2019)

[27] Mavridou, A., Laszka, A., Stachtiari, E., Dubey, A.: Verisolid: Correct-by-design smart contracts for ethereum. In: International Conference on Financial Cryptography and Data Security. pp. 446–465. Springer (2019)

[28] van der Meyden, R.: On the specification and verification of atomic swap smart contracts. In: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 176–179. IEEE (2019)

[29] Mukhopadhyay, M.: Ethereum Smart Contract Development: Build blockchain-based decentralized applications using solidity. Packt Publishing Ltd (2018)

[30] Nelaturu, K., Mavridou, A., Veneris, A., Laszka, A.: Verified development and deployment of multiple interacting smart contracts with verisolid. In: Proc. of the 2nd IEEE International Conf. on Blockchain and Cryptocurrency (ICBC) (2020)

[31] Nikolić, I., Kolluri, A., Sergey, I., Saxena, P., Hobor, A.: Finding the greedy, prodigal, and suicidal contracts at scale. In: Proceedings of the 34th annual computer security applications conference. pp. 653–663 (2018)

[32] Petrucci, L., Seceleanu, C., Cavalcanti, A. (eds.): Critical Systems: Formal Methods and Automated Verification - Joint 22nd International Workshop on Formal Methods for Industrial Critical Systems - and - 17th International Workshop on Automated Verification of Critical Systems, FMICS-AVoCS 2017, Turin, Italy, September 18-20, 2017, Proceedings, Lecture Notes in Computer Science, vol. 10471. Springer (2017)

[33] Qu, M., Huang, X., Chen, X., Wang, Y., Ma, X., Liu, D.: Formal verification of smart contracts from the perspective of concurrency. In: International Conference on Smart Blockchain. pp. 32–43. Springer (2018)

[34] Ray, I.: Security Vulnerabilities in Smart Contracts as Specifications in Linear Temporal Logic. Master's thesis, University of Waterloo (2021)

[35] Wang, F., Cao, Z., Tan, L., Zong, H.: Survey on learning-based formal methods: Taxonomy, applications and possible future directions. IEEE Access **8**, 108561–108578 (2020)

[36] Weiss, K., Schütte, J.: Annotary: A concolic execution system for developing secure smart contracts. In: European Symposium on Research in Computer Security. pp. 747–766. Springer (2019)

[37] Yamashita, K., Nomura, Y., Zhou, E., Pi, B., Jun, S.: Potential risks of hyperledger fabric smart contracts. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE). pp. 1–10. IEEE (2019)

[38] Zupan, N., Kasinathan, P., Cuellar, J., Sauer, M.: Secure smart contract generation based on petri nets. In: Blockchain Technology for Industry 4.0, pp. 73–98. Springer (2020)