Computers & Security 88 (2020) 101654

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities

Amritraj Singh[a], Reza M. Parizi[a], Qi Zhang[b], Kim-Kwang Raymond Choo[c,*], Ali Dehghantanha[d]

[a] Department of Software Engineering and Game Development, Kennesaw State University, GA 30060, USA
[b] IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 10598, United States
[c] Department of Information Systems and Cyber Security, University of Texas at San Antonio, Texas, United States
[d] Cyber Science Lab, School of Computer Science, University of Guelph, Ontario, Canada

ARTICLE INFO

Article history:
Received 30 September 2018
Revised 20 October 2019
Accepted 21 October 2019
Available online 22 October 2019

Keywords:
Blockchain
Smart contracts
Formal methods
Verification
Systematic review

ABSTRACT

Blockchain as a distributed computing platform enables users to deploy pieces of software (known as smart contracts) for a wealth of next-generation decentralized applications without involving a trusted third-party. The advantages of smart contracts do, however, come at a price. As with most technologies, there are potential security threats, vulnerabilities and various other issues associated with smart contracts. Writing secure and safe smart contracts can be extremely difficult due to various business logics, as well as platform vulnerabilities and limitations. Formal methods have recently been advocated to mitigate these vulnerabilities. This paper aims to provide a first-time study on current formalization research on all smart contract-related platforms on blockchains, which is scarce in the literature. To this end, a timely and rigorous systematic review to examine the state-of-the-art research and achievements published between 2015 and July 2019 is provided. This study is based on a comprehensive review of a set of 35 research papers that have been extracted from eight major online digital databases. The results indicate that the most common formalization technique is theorem proving, which is most often used to verify security properties relating to smart contracts, while other techniques such as symbolic execution and model checking were also frequently used. These techniques were most commonly used to verify the functional correctness of smart contracts. From the language and automation point of views, there were 12 languages (domain specific/specification/general purpose) proposed or used for the formalization of smart contracts on blockchains, while 15 formal method-specific automated tools/frameworks were identified for mitigating various vulnerabilities of smart contracts. From the results of this work, we further highlight three open issues for future research in this emerging domain including: formal testing, automated verification of smart contracts, and domain specific languages (DSLs) for Ethereum. These issues suggest the need for additional, in-depth research. Our study also provides possible future research directions.

1. Introduction

In 2008, Satoshi Nakamoto introduced the formal idea of blockchain (Nakamoto, 2008) as an infrastructural technology powering Bitcoin[1] cryptocurrency. Since then, blockchain has made its mark on a wide range of industrial domains such as Security, Privacy, Finance, Cloud computing, Internet of Things (IoT) and many others. More recently, smart contracts (Cuccuru, 2017) have emerged as a new promising use case of the blockchain widening its horizons and turning it into distributed computing platform. Smart contracts are self-executing contracts where users can codify their agreements and trust relations, which are then stored on a hosting blockchain. Smart contracts can facilitate safe and trusted business activities by providing automated transactions without the supervision of an external financial system such as banks, courts, or notaries. These transactions are traceable, transparent, and irreversible. The smart contracts are generally written

* Corresponding author.
E-mail addresses: amritra@students.kennesaw.edu (A. Singh), rparizi1@kennesaw.edu (R.M. Parizi), q.zhang@ibm.com (Q. Zhang), raymond.choo@fulbrightmail.org (K.-K.R. Choo), adehghan@uoguelph.ca (A. Dehghantanha).

[1] https://bitcoin.org/en/.

https://doi.org/10.1016/j.cose.2019.101654
0167-4048/

with domain-specific languages, such as Solidity[2] on Ethereum,[3] Pact[4] on Kadena,[5] Liquidity[6] on the Tezos[7] platform to ease contract programming. In some instances, general-purpose languages such as Kotlin,[8] Go[9] and Java[10] are also used to write smart contracts, mainly because of the familiarity and usability reasons for the developers.

Although smart contracts seem simple to be implemented and understood, it is hardly ever the case in real-world situations as we move forward to more scalable smart contracts in blockchain 2.0. Parizi et al. (2018) performed an empirical evaluation of Solidity, Pact and Liquidity languages based on usability and security to a new smart contract developer. The results of their experiments suggested that, in terms of usability, Solidity is the most usable language for a new developer to write smart contracts, but, when it comes to security, new developers tend to write vulnerable contracts with Solidity which may be used by malicious entities to cause financial damages. One such infamous malicious attack took place in June 2016, when the DAO (Decentralized Autonomous Organization) smart contract was manipulated to steal around 2 Million (50 Million USD) Ether. The contract was vulnerable to reentrancy issues (Liu et al., 2018), owing to which a recursive call was made to the smart contract's 'splitDAO' function for withdrawing Ether when only one call was to be allowed according to the contract specification. In another work, Atzei et al. (2017) provided a summary and brief analysis of some infamous security vulnerabilities of Solidity and the Ethereum platform. They introduced and classified a taxonomy of causes of vulnerabilities on three levels: Solidity, EVM and Blockchain. Additionally, the authors accompanied this taxonomy with actual attacks which exploit these vulnerabilities. Similarly, Destefanis et al. (2018) provided a case study of the Parity wallet incident which led to the freezing of 500 K Ether[11] (150 M USD) in 2017. The results from their analysis showed that the incident was a consequence of poor programming practices rather than the imperfections of the Solidity language itself.

The above-mentioned studies and incidents show that both inexperienced and experienced smart contract developers can often write vulnerable smart contracts, which may be failure prone and vulnerable to attack from malicious entities. Recognizing this challenge of writing safe and secure smart contracts, researchers from both academic and industrial communities have recently started to focus their attention on the use of formal methods for the verification of smart contracts before they are deployed on the blockchain. The process of formal verification involves proving a contract code is correct for all inputs in its state space and hence, verifying that the contract behaves according to its specification. This process is generally performed using concrete specification languages to describe how the input and output of functions relate to each other. The formal verification process is nothing new to software engineering and is primarily used in designing safety-critical systems such as aircraft and medical systems. However, its use has been limited in other traditional industries due to the cost and time involved in the proving and verification process.

Therefore, to answer the obvious looming question, "*Why do we need to use formal methods in smart contracts verification?*". It is because (1) smart contracts are immutable, even though some platforms such as Hyperledger Fabric[12] allow the smart contract to be updated, one cannot just patch them easily if a bug or a vulnerability surfaces in the future, (2) many smart contracts store and operate on valuable assets, and (3) smart contracts on public blockchains are accessible from all over the world. This makes them very attractive for a malicious user to carry out attacks. Hence, formal verification is a strong approach that can reduce the risk of bugs and vulnerabilities in a contract and can help prevent malicious attacks in the future.

The number of studies in formalization of smart contracts has grown steadily in the recent years, but rigorous and formal studies such as systematic reviews in the same domain are still in a state of infancy. In fact, to the best of our knowledge, our work is the first systematic review in the field of smart contract formalization to summarize state-of-the-art research and practice and helps transfer results to both educators and practitioners. The study revolves around a specific set of research questions (presented in Section 3.1), which are then addressed via a thorough content analysis of a large set of academic and professional studies. The specific contributions of this study are as follows:

- Evaluation of the current formalization approaches for smart contracts while highlighting the innovative approaches and achievements in the domain.
- Analysis of the most commonly researched issues and vulnerabilities related to smart contracts on blockchains in formalization approaches.
- Review of Domain-Specific Languages (DSLs) and other formal languages proposed to date in literature for the mitigation of vulnerabilities in smart contracts.
- Overview of state-of-the-art tools and frameworks used for formalizing smart contracts.
- Identification of open issues, possible solutions to mitigate these issues and future directions to advance the state of research in the domain.

The remainder of this paper is structured as follows: Section 2 presents an overview of some of the related literature reviews / surveys in the field of blockchains and highlights the gap in blockchain research. Section 3 gives the research methodology and the whole process of paper exploration and selection as well as threats to validity and limitations. Section 4 presents the results and the discussion with respect to research questions. In Section 5, we discuss open issues and future directions to potentially solve these issues. Finally, Section 6 reports the conclusion.

2. Related literature reviews / surveys

This section provides a brief overview of the state-of-the-art secondary studies (i.e., those have review/survey nature) in the steadily growing domain, blockchain-based systems and highlights the gap in blockchain research, i.e. smart contract formalization (the summary of all related works in the area of blockchain smart contracts and formal methods are given in Appendix A).

Yli-huumo et al. (2016) conducted a systematic literature review (SLR) in order to determine what current research was published in relation to the general concept of blockchain technology. They excluded legal, economic and regulatory research from their review and focused on the technical blockchain papers; they found an 80% focus on Bitcoin projects and in particular a common theme of security and privacy. Since 2016 the applications for blockchain have diversified and as such our research looks to establish what research exists specifically in regard to cybersecurity and blockchain applications.

[2] https://solidity.readthedocs.io/.
[3] https://www.ethereum.org/.
[4] http://kadena.io/docs/Kadena-PactWhitepaper.pdf.
[5] http://kadena.io/.
[6] https://github.com/OCamlPro/liquidity/blob/master/docs/liquidity.md.
[7] https://tezos.com/.
[8] https://kotlinlang.org/.
[9] https://golang.org/.
[10] https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html.
[11] https://www.ethereum.org/ether.
[12] https://www.hyperledger.org/projects/fabric.

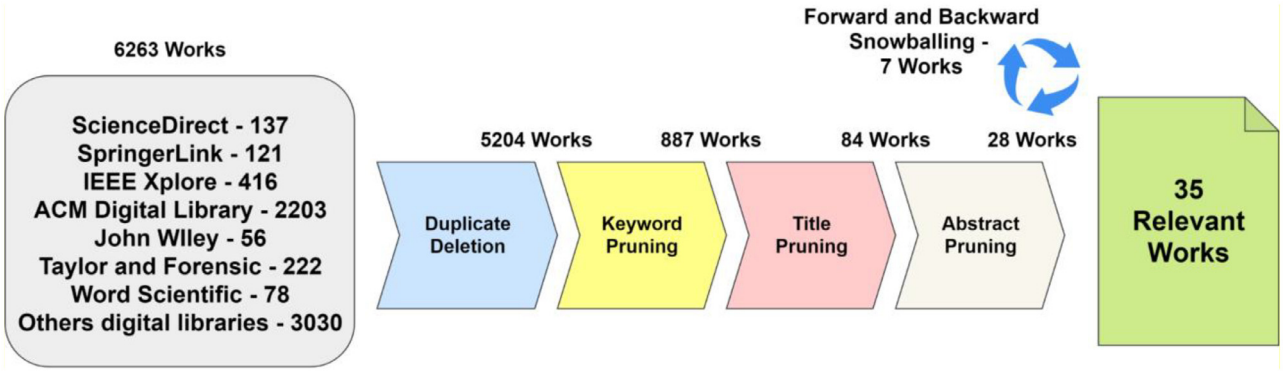


Fig. 1. Attrition of paper throughout the selection process.

Towards the end of 2016, Conoscenti et al. (2016) conducted an SLR concerning the use and adaptability of blockchain specifically in relation to IoT and other peer-to-peer devices. Interestingly, they highlighted that the blockchain could be used for data abuse detection without the need of a central reporting mechanism.

Seebacher and Schüritz (2017) provided an SLR in 2017 that highlighted blockchain was increasingly more impactive on service systems. The results of their SLR suggests that blockchain plays an integral role in the functioning of a service system. Blockchain facilitates co-creation of value, ensures information availability and offers coordination mechanism in such systems. The authors also recommended conducting large scale empirical study of real-world applications for the future of research in the area.

Most recently in 2018, Reyna et al. (2018) surveyed relevant works to identify current challenges and aspects of improvement in the integration of blockchain and IoT domains. Their survey suggests that the current state of integration of the two technologies face six challenges namely: scalability, security, privacy, smart contracts, legal issues and consensus. Additionally, they provided an evaluation to highlight the benefits of using blockchains with IoT devices.

All the previous studies mentioned above answer questions relating to various aspects of the blockchain technology and its integration with other fields, but do not look specifically in mitigating smart contracts' vulnerabilities and issues using formularization approaches. The field of research in relation to the blockchain has a relatively brief history, but smart contracts issues and security have received lesser attention. Hence, our work provides the first SLR that focuses on the formalization approaches of smart contracts for the improvement of correctness and vulnerabilities. Our work is based on a well-defined systematic protocol for the collection of all relevant works proposed in the literature since the inception of this new domain. We provide an analysis of the current research interests and areas in the academic and professional communities. Additionally, the work presented in this paper identifies current open issues and research gaps to shift the focus of researchers towards them as they require well-deserved attention.

3. Research methodology

A SLR is based on the identification, evaluation and interpretation of all available research relevant to one or more research questions or a topic of interest (Kitchenham, 2004; Kitchenham and Charters, 2007). Such studies are conducted based on a search and a review protocol, which describes the research questions to be addressed among other things such as the procedures for searching and identifying relevant research works and defining the process by which the data collected are synthesized to answer the research questions and achieve the goals of the review.

3.1. Research questions

Along with the motivation and the main stated objective, this study specifically aims to answer the research questions (referred to as RQ's) mentioned below:

- **RQ1**: What formal methods and techniques are used for the verification and improvement of smart contracts?
- **RQ2**: Which issues or vulnerability aspects of smart contracts do formalization approaches target?
- **RQ3**: How do formalization approaches mitigate issues and vulnerabilities in smart contracts?
- **RQ4**: What domain specific languages (DSL) or formal/specification/general-purpose languages are proposed/used for formalizing smart contracts?
- **RQ5**: What automated tools and frameworks are proposed in supporting state-of-the-art formalization approaches of smart contracts?

To answer these questions, we designed a systematic protocol to search and identify all the related works in the domain of formalization approaches of smart contracts published between 2015 and July 2019. We have defined the search string used to select the primary studies based on our RQ's. In the subsequent sub-sections, we describe the different steps that we took to search, identify and purify the relevant works for our study for which we have also defined the set of inclusion/exclusion criteria.

3.2. Inclusion and exclusion criteria

To ensure that the collected relevant works aligned with our systematic review, we defined several criteria to determine the works that would be considered and the ones that were outside the scope of this review. For a work to be considered for this systematic review, it must be a relevant work in the field of formalization of smart contracts on blockchains and must be present in one of the online databases mentioned in Fig. 1.

In addition to being published in one of the above-mentioned databases, each work had to satisfy all the inclusion criteria mentioned in Table 1. On the other hand, the exclusion criteria were designed to determine the works that were considered irrelevant for this review. These criteria are also mentioned in Table 1.

3.3. Search strategy and pruning process

After we identified the online digital libraries (See Fig. 1) to search for relevant works, we started the identification and collection of relevant works. We first identified what we called the "preliminary set of works". To identify and collect the preliminary set of works, we performed string and manual keyword searches

Table 1
Inclusion and exclusion criteria for relevant works.

Inclusion Criteria	Exclusion Criteria
Be published online from 2015 to July 2019.	white papers, editorial comments and book reviews
Studies are in the field of smart contracts and blockchains	Studies that present surveys and review papers
Studies offer technical quality method in the formalization of smart contracts	Studies that are not published in English
Studies that are available in technical archives or have a peer-review procedure	Studies not par with current technical quality aspects

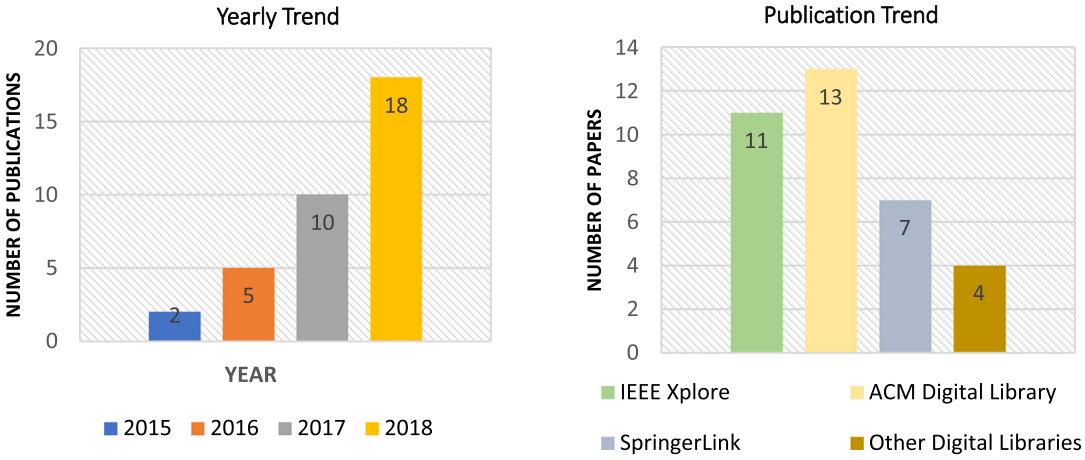


Fig. 2. Yearly and publication trend.

on the identified digital libraries. The results from these searches are summarized in Fig. 1, which shows the total number of preliminary studies acquired from each digital database.

Once we identified the initial set of 6263 research papers, we started the pruning process, i.e. discarding the works which were unrelated to the goals of this research. We applied four stages of pruning to the preliminary set of works to obtain the initial set of works for this review, these pruning stages are briefly described below:

1 **Duplicates deletion:** removal of duplicate studies
2 **Keyword pruning:** filtering studies based on a list of relevant keywords
3 **Title pruning:** filtering based on the title of the work
4 **Abstract pruning:** filtering based on abstract of the work

Fig. 1 highlights our pruning process and shows the number of papers selected at each stage the initial set of works. To obtain the final set of *relevant works*, we conducted a quality assessment of all the initial 28 works (which we obtained after applying all the pruning stages). At this stage, we carefully read all 28 papers gathered up to this point and then, performed a snowballing procedure. We paid keen attention on the bibliographical references and "Related Work" section of all the initial set of studies. This was done to collect new studies that might have been missed in the collection of the initial set of works. Here, our aim was to: 1) make sure that the new works assembled complied with the inclusion criteria defined in Section 3.2, and 2) the new works collected complimented the list of initial set of 28 works.

Hence, reading through all the initial set of works, we recognized an additional 7 papers to complement the initial set of works ultimately, providing us with the final set of 35 related works for this SLR (See references (Hirai, 2017; Amani et al., 2018; Le et al., 2018; Bhargavan et al., 2016; Bigi et al., 2015; Abdellatif and Brousmiche, 2018; Bai et al., 2018; Kalra et al., 2018; Chaudhary et al., 2015; Kim and Laskowski, 2017; Dennis et al., 2016; Breidenbach et al., 2017; Kosba et al., 2016; Matsuo, 2017; Luu et al., 2016; Tsankov et al., 2018; Mueller, 2018; Zhou et al., 2018; Nikolic et al., 2018; He et al., 2018; Scoca et al., 2017; Biryukov et al., 2017; Mavridou and Laszka, 2018; Ellul and Pace, 2018; Idelberger et al., 2016; Sergey and Hobor, 2017; Liao et al., 2017; Pîrlea and Sergey, 2018; Grishchenko et al., 2018; Cerezo Sánchez, 2017; Zhang et al., 2016; Hildenbrandt et al., 2018) and (O'Connor, 2017; Grossman et al., 2018; Sergey et al., 2018)). A brief summary of all the relevant studies (i.e. final set of works) has been provided in Appendix A.

Fig. 2 shows the number of studies published each year since 2015 concerning formalization of smart contracts. The figure suggests the domain is growing rapidly ever since the inception of the idea. This rapid growth in studies is a direct consequence of the recent financial damages caused by either malicious attacks on smart contracts or inadvertent bugs that froze funds by producing unexpected irreversible results. The figure also shows the relevant studies found in each online database.

3.4. Limitations and threats to validity

SLRs may suffer from some threats to the validity and well-known limitations that we discuss in this section, alongside the measures taken to minimize their impact on the results of this review.

Possibility of bias in relevant work search: The inclusion of all the relevant work for this systematic review is hard to guarantee. There exists a small chance that some studies that were relevant for this review might have been excluded inadvertently based on the search criteria defined in Section 3. Hence, to mitigate this threat, searches were performed in popular and reputable conferences, workshops, and journals both manually and automatically, keeping in check with the critical references listed in the initial set of works to make sure no additional studies were excluded from the final set of works.

Data extraction procedure limitations: We discovered a few difficulties in extracting data from the set of relevant works for this SLR. In some cases, the information provided in the relevant works had to be subjectively interpreted as not all of them provided explicit information that would help us in directly answering our research questions (RQs). For instance, some of the relevant works

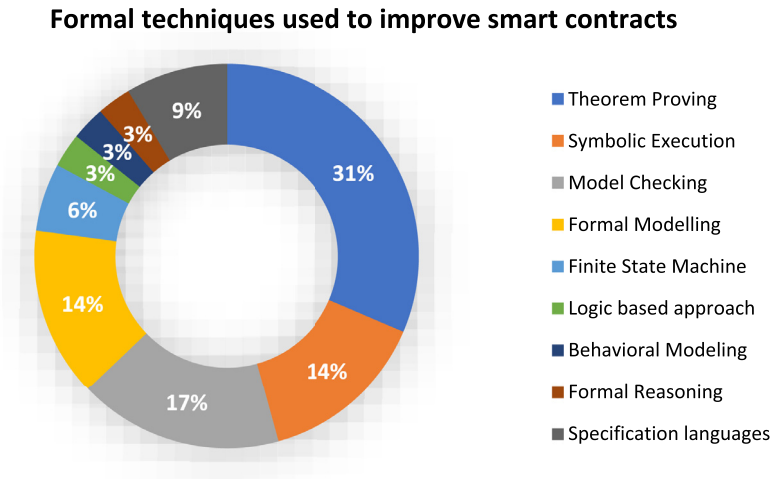


Fig. 3. The formal techniques used to improve smart contracts.

did not mention the technique, methodology, technology or the tool used for the formalization or improvement of smart contracts on blockchains.

4. Results and discussion

In this section, we present the results based on the analysis of all the relevant work to answer the research questions outlined in Section 3.1.

4.1. RQ1: What formal methods and techniques are used for verification and improvement of smart contracts?

It is important to emphasize that at least some degree of formalization takes place for the verification or improvement of smart contracts in all the studies selected and analyzed in this work. Several papers discuss more than one formalization technique. Keeping this in mind, our observations drew out 9 major formalization approaches which are discussed below:

Theorem Proving, is a formal method of providing proofs in symbolic logic utilizing deductive inference. In this approach, each step in the proof introduces an axiom or a premise and provides a statement which is a natural consequence of the previously established results using legitimate rules of inference. Theorem proving was the most typical way of formalizing smart contracts. Of all studies under evaluation, there were 11 (31%) papers dedicated to the theorem proving technique. Some of the studies that utilize theorem proving formalization approaches are (Hirai, 2017; Amani et al., 2018; Le et al., 2018). This formalization approach is used to prove various security properties that can greatly improve the reliability of smart contracts. Fig. 3 provides a breakdown of the formal techniques used for the improvement (i.e. outsmarting) of smart contracts.

Model checking (also known as *Property Checking*) is the process of checking whether a given finite-state model of a system behaves according to its formal specification or correctness properties. This technique was the second most used approach (mentioned in 6 (17%) relevant works), which is most commonly used in studies to verify important properties and correctness of smart contracts (Bhargavan et al., 2016; Bigi et al., 2015; Abdellatif and Brousmiche, 2018; Bai et al., 2018), and (Kalra et al., 2018). Model checking approach has also been used for the verification of blockchain consensus protocol correctness (Chaudhary et al., 2015). Interestingly, we found that all the papers that proposed this approach of formalization have used an automatic model checking tool to verify the correctness and important properties of smart contracts. Some of the used model checkers in the relevant works are SPIN,[13] UPPAAL,[14] and SMC (Legay et al., 2010) (Statistical Model Checking) tools.

Formal Modelling is the technique in which a system is designed using precise statements or components, these statements define each relationship between all components of a system resulting in 1) unambiguous communication, and 2) replicable results after following a series of steps. Formal Modelling was the third most common approach used for formalizing smart contracts and was mentioned in 5 (14%) relevant works (Kim and Laskowski, 2017; Dennis et al., 2016; Breidenbach et al., 2017; Kosba et al., 2016; Matsuo, 2017). Formal modelling technique is used to provide solutions to multiple challenges faced by smart contracts and blockchains such as scalability, privacy, providing bug bounties, etc. These aspects of smart contracts and blockchains are discussed in further detail in Section 4.2 (RQ2).

Symbolic execution is a software testing technique that aids test data generation and proofs regarding the quality of a program.[15] This technique was also mentioned in 5 (14%) relevant works, and it was observed as the most preferred approach in providing a backbone to studies that propose static analysis and verification tools [28, 29, 30, 31, and (Nikolic et al., 2018). These analysis results were achieved by the identification and verification of various dangerous patterns which may leave a smart contract vulnerable to failure or attack by malicious entities.

The remaining 8 (24%) papers reviewed and analyzed propose the following formalization approaches of smart contracts: 1) *formal specification languages* which are the programming languages based on formal specification (a technique to describe a behavior or system in terms of precise mathematical statements) such as SPESC (He et al., 2018), dSLAC (Scoca et al., 2017) and Findel (Biryukov et al., 2017) for developing smart contracts; 2) *Finite State Machine* (which is a technique in which a system is mathematically defined as an abstract machine that can be in one state out of a finite number of states at any given time. These machines can change state based on external inputs) based approach (FSM) that has been used to support automatic code generation tools and to provide security plugins and implement common design patterns for the safety and reliability of smart contracts (Mavridou and Laszka, 2018) and to verify the runtime safety

[13] http://spinroot.com/spin/whatispin.html.
[14] http://www.uppaal.org/.
[15] https://www.tutorialspoint.com/software_testing_dictionary/symbolic_execution.htm.

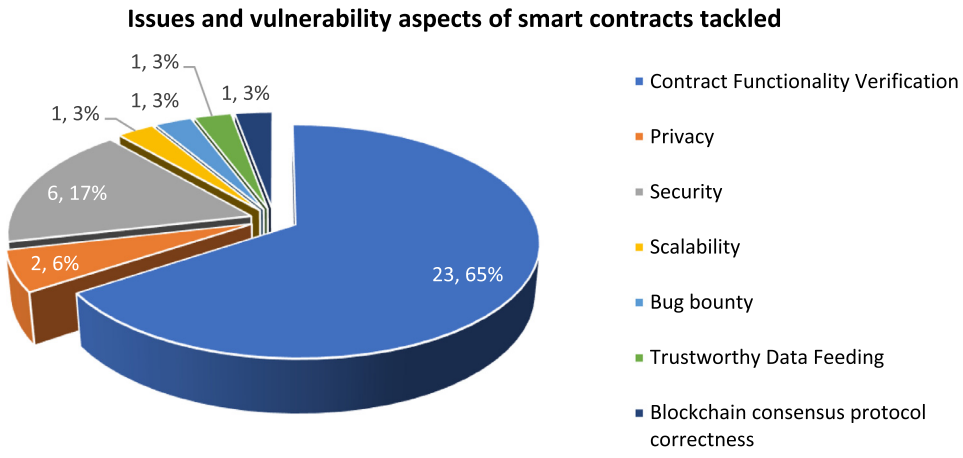


Fig. 4. The issues and vulnerability aspects of smart contracts tackled by formalization approaches.

and verification of Ethereum smart contracts (Ellul and Pace, 2018); 3) *Logic based approach* (Idelberger et al., 2016) (proving the robustness of a system based on formal logic) which was used to provide results suggesting that logic based languages have the potential to accompany common scripting languages for developing smart contracts. 4) *Formal reasoning* (which is an approach to draw out conclusions based on logical and deductive reasoning based on one or more premises) approach that was used to explore similarities between classical problems of shared-memory concurrency and multi-transactional behavior of Ethereum smart contracts (Sergey and Hobor, 2017). Finally, 5) *behavioral modeling* (Liao et al., 2017) ("*Behavioral models*[16] describe the internal dynamic aspects of an information system that supports the business processes in an organization. During analysis, behavioral models describe what the internal logic of the processes is without specifying how the processes are to be implemented") which was used in an approach to propose a platform that supports Behavior-Driven Development (BDD), deployment and testing of smart contracts for the Ethereum platform.

4.2. RQ2: Which issues or vulnerability aspects of smart contracts do formalization approaches target?

It is important to keep in mind that most of the papers selected for this review discuss and mitigate more than one issue or vulnerability aspect of smart contracts. Hence, we will answer this research question based on the primary vulnerability mitigation focus of each paper. Fig. 4 provides a breakdown of all the issues or vulnerability aspects of smart contracts tackled by formalization approaches. These issues and vulnerabilities are classified into seven categories namely: *Contract functionality verification, privacy, security, scalability, bug bounty, trustworthy data feeding, and blockchain consensus protocol correctness*.

The most common vulnerability aspect of smart contracts that are targeted by formal verification approaches was *contract functionality verification*, e.g., does the smart contract offer the exact functionality that it is designed or programmed for? Are there loopholes or dangerous boundary conditions that may produce unexpected results? We found that 23 (65%) of the studies had focused on contract functionality verification aspect of smart contracts. Some of these studies are (Mueller, 2018; Luu et al., 2016; Tsankov et al., 2018; Le et al., 2018; Mavridou and Laszka, 2018).

The second most discussed issue related to smart contracts is *security*. We found 6 (17%) relevant works that discuss various security-related issues with smart contracts on blockchains, these works are (Matsuo, 2017; Pîrlea and Sergey, 2018; Le et al., 2018; Zhou et al., 2018; Sergey and Hobor, 2017) and (Grishchenko et al., 2018). The security issues highlighted in these papers are described as follows: Matsuo (Matsuo, 2017) proposes an innovative approach for the application of formal analysis and verification by considering implementation, protocol, and language layers of a blockchain based system. Pirlea and Sergey (Pîrlea and Sergey, 2018) present a formal model of a distributed blockchain-based consensus protocol and mechanically prove the protocol's eventual consistency. Le et al. (Le et al., 2018) define the input conditions for which a smart contract terminates (or does not terminate) by proving conditional termination and non-termination statically. Zhou et al. (Zhou et al., 2018) introduce a method to detect potential security risks in smart contract programs with the help of Syntax topology analysis of smart contract invocation relationship and Detection and location of logical risks. Sergey and Hobor (Sergey and Hobor, 2017) explore similarities between classical problems of shared-memory concurrency and multi-transactional behavior of Ethereum smart contracts. They also introduce improved practices with the application of formal verification techniques for developing smart contracts. Finally, Grishchenko et al. (Grishchenko et al., 2018) give the formal definition of several security properties for smart contracts.

Privacy was the third most discussed issue related to smart contracts. There were 2 (6%) studies which discuss issues related to privacy. Kosba et al. (Kosba et al., 2016) highlight the issue of transactional privacy in existing blockchain based systems and in turn propose the 'Hawk' decentralized smart contract system, which does not store the financial transaction publicly on the blockchain hence, providing transactional privacy. Similarly, Sánchez (Cerezo Sánchez, 2017) proposes 'Raziel' which provides privacy, verifiability and correctness guarantees of smart contracts. The author introduces ways in which Zero-Knowledge Proofs of Proofs or Proof-Carrying program certificates can be used to prove the validity of smart contracts before execution to other parties in a private manner.

Other issues that were discussed in the relevant works are related to *scalability, bug bounties, trustworthy data feeding* and *blockchain consensus protocol correctness*. Each of these issues has only been addressed in 1 relevant work. Dennis et al. (Dennis et al., 2016) introduced and provided a solution for the scalability issue of blockchain based systems by proposing a constant fixed size blockchain called "rolling blockchain". They showed that the dele-

[16] https://www.oreilly.com/library/view/systems-analysis-and/9781118037423/11_chapter006.html.

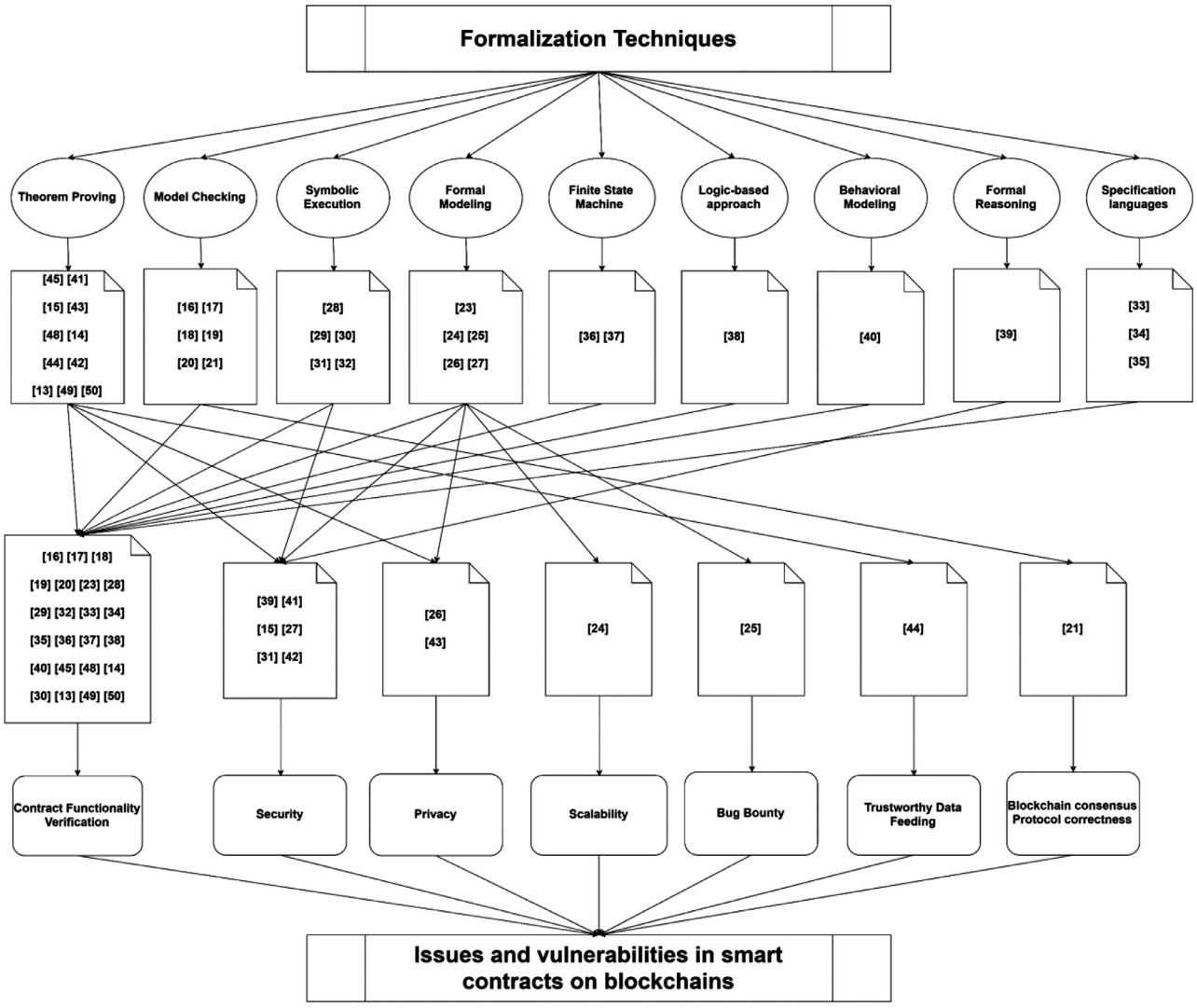


Fig. 5. Mapping of the relevant works based on RQ1 and RQ2.

tion of data from the proposed model of this blockchain does not affect its security as compared to traditional blockchains. When it comes to bug bounties, Breidenbach et al. (Breidenbach et al., 2017) introduced the Hydra Framework to incentivize honest disclosures of bugs and vulnerabilities in smart contracts. In case of trustworthy data feeding to smart contracts, Zhang et al. (Zhang et al., 2016) proposed Town Crier which acts as a link between existing commonly trusted non-blockchain based websites and smart contracts to provide authenticated data to smart contracts. And finally when it comes to the blockchain consensus protocol correctness issue, Chaudhary et al. (Chaudhary et al., 2015) introduced the potential problem of double spending in Bitcoin and the lack of a third party to guard against the problem.

4.3. RQ3: How do formalization approaches mitigate issues and vulnerabilities in smart contracts?

To provide a more taxonomic analysis, we further mapped out the results from RQ1 and RQ2. This particular analysis was aimed to provide a clearer perspective on all the studies based on the formalization approach they utilize and the issues or vulnerability aspects they mitigate. The results are shown in Fig. 5. As it can be seen from the figure, 1) all formalization techniques mentioned in RQ1 except for formal reasoning were used for the smart contract functionality verification. 2) Theorem proving, Symbolic Execution, formal modeling and formal reasoning were used to improve security aspect of smart contracts. 3) Theorem proving, and formal modeling were used to improve privacy of smart contracts. 4) formal modeling was used to improve scalability of blockchains and administer bug bounties. 5) Theorem proving was used to propose a way to provide trustworthy data from websites to smart contracts and finally, 6) Model checking was used to verify blockchain consensus protocol correctness.

We now discuss how each formalization technique (identified in RQ1) was used to mitigate each of issue and vulnerability (identified in RQ2) in the smart contract domain. This analysis is structured as follows, we examine each smart contract issue or vulnerability in its own separate and specific sub-section (i.e. sub-sections A - G). In each sub-section, we will discuss which formalization techniques were used in what manner to mitigate that issue or vulnerability.

4.3.1. Contract functionality verification

The functional correctness of software is defined as the ability of the software to produce an output for a specific input as specified by the functional requirements of that software. Hence, it is fairly intuitive to understand the meaning of functional correctness of smart contracts. It is the ability of a smart contract to produce

an output as governed by its specification for each input from a set of all possible inputs for that smart contract. In our analysis we found smart contract functionality verification to be tackled by the all formal techniques accept *formal reasoning*. We found contract functionality verification to be the most researched area in terms of smart contract issues and vulnerabilities as it attracted researchers from the most diverse background of formalization techniques.

Theorem proving is used by studies to 1) propose KEVM (Hildenbrandt et al., 2018), an executable formal specification of the EVM's bytecode stack-based language built with the K Framework. The KEVM was designed to provide a backbone for rigorous formal analysis of smart contracts. 2) Provide the definition of EVM in Lem (Hirai, 2017), which is a language that can be compiled for several known theorem provers. Using the provided definition, the author provided proofs for several safety properties of Ethereum smart contracts in Isabelle/HOL theorem prover.

Symbolic execution was used to build a backbone for tools such as 1) SASC, which is a static analysis tool that can find potential logic risks and generate topological charts of invocation relationships and 2) MAIAN, Oyente, Mythril and Securify to analyze smart contracts for potentially dangerous patterns which are exploitable by malicious users. The tool utilizes symbolic analysis and concrete validator to exhibit real vulnerabilities. *Model checking* have been used in studies such as (Bai et al., 2018) generating smart contract models. A model verification tool can be used to verify important properties and correctness of smart contracts. A case study was presented with the help of a famous model checking tool SPIN to illustrate the verification process and effects.

Formal modeling was used to (Kim and Laskowski, 2017) state principles regarding the possible evolution of smart contracts on blockchains, including: 1) Blockchains, whilst reducing the uncertainty of value exchange can also increase complexity from having to subsume work from third parties; 2) Smart contracts can be used to decrease the complexity arising from eliminating third parties with the use of blockchain; 3) The evaluation of smart contract models of several parties lowers uncertainty of value exchange whilst increasing transparency. Additionally, the study also emphasizes the use of formal models (mathematical, logical, or simulation-based) which can also increase transparency when evaluating or interpreting smart contracts of different parties.

Finite state machines were used to (Ellul and Pace, 2018) demonstrate that standard techniques for runtime verification of smart contracts, including a stake-based instrumentation technique which ensures that the violating party provides insurance for correct behavior. Idelberger et al. (Idelberger et al., 2016) outline the technical and legal disadvantages of *logic-based* smart contracts in regards of usual activities featuring ordinary contracts. The authors provide guidelines towards usage of such *logic based* smart contracts on blockchain based systems.

Scoca et al. (Scoca et al., 2017) introduce a *formal specification language* that can be used to specify interactions between requests and offers. The paper also presents an approach for the self-governed negotiation of smart contracts, which is used to analyze the cost and the necessary changes required to reach an agreement. Finally, Liao et al. (Liao et al., 2017) introduce a platform that supports *Behavior-Driven Development (BDD)*, deployment and testing of smart contracts for the Ethereum platform. The objective of the platform is to provide and resolve cross-cutting concerns across smart contract development life cycle.

4.3.2. Security

Security of a smart contract refers to its robustness against attacks from malicious users. It is important to keep in mind that the security of a smart contract tightly coupled to its correctness and blockchain consensus protocol. In our analysis we found that there were four formalization approaches used to improve the security guarantees of a smart contract.

Sergey and Hobor (Sergey and Hobor, 2017) explore similarities between classical problems of shared-memory concurrency and multi-transactional behavior of Ethereum smart contracts with the help of *formal reasoning*. They examine and analyze two examples from the Ethereum blockchain based on "how they are vulnerable to bugs that are closely reminiscent to those that often occur in traditional concurrent programs". This description of contracts-as-concurrent-objects provides a deeper knowledge of potential smart contract threats. The results also introduce improved practices with the application of formal verification techniques for developing smart contracts.

Formal modelling technique was utilized by Matsuo (Matsuo, 2017) for the application of formal analysis and verification by considering technology layers of blockchain-based systems and their security concerns. These layers are identified as, implementation, protocol, and language. *Symbolic Execution* was utilized by Zhou et al. (Zhou et al., 2018) to propose a static analysis tool called SASC for smart contract logic analysis and generation of topological invocation relationship diagrams.

Theorem proving was used by Pîrlea and Sergey (Pîrlea and Sergey, 2018) to introduce several theorems regarding pure functional implementation of block forests. The authors work is based on several primitive security features like, hash-functions, a notion of a proof object, a Validator Acceptance Function, and a Fork Choice Rule. Furthermore, the authors characterize their assumptions regarding these components to prove the consensus of the global system. Theorem proving was also utilized by Grishchenko et al. (Grishchenko et al., 2018) to provide initial semantics of EVM bytecode formalized in F* proof assistant. Le et al. (Le et al., 2018) determined the input conditions for which a smart contract terminates (or does not terminate) by proving conditional termination and non-termination statically. This is done by making sure that both, current state of the smart contract and the contract's input satisfy the termination condition to run on a proof carrying blockchain before the actual execution of the contract.

4.3.3. Privacy

Private smart contracts on blockchains allow disparate parties to transact amongst themselves without having to reveal the terms of their contract or their transactions on the blockchain to the public. In our analysis we found two papers that focused mainly on privacy. The way in which these papers improve privacy using formal techniques are discussed below:

Hawk (Kosba et al., 2016), which is a decentralized system for smart contracts is based on the *formal modeling* technique that provides transactional privacy by not storing financial transactions in the clear on the blockchain. Hawk allows intuitive development of private smart contracts without requiring the implementation of cryptography. The platform's compiler automatically generates cryptographic protocol wherever the parties bound by contract interact with the blockchain. This is achieved with the help of zero-knowledge proofs (Sah et al., 2016).

Raziel is backed by the *theorem proving* formalization technique (Cerezo Sánchez, 2017). It provides privacy, verifiability and correctness guarantees of smart contracts by combing proof carrying programs and secure multi-party computations. The paper also demonstrates the ways in which Zero-Knowledge Proofs of Proofs or Proof-Carrying program certificates can be used to prove the validity of smart contracts before execution to other parties in a private manner.

4.3.4. Scalability

Scalability for any system can be defined as its capability to grow in size and manage increased demands by its users. The core

problem surrounding blockchain networks is scalability in terms of its limitations on the amount of transactions that it can process. Public blockchains such as Ethereum and Bitcoin suffer from this problem as Ethereum network can process approximately 15 – 25 transactions per second and the Bitcoin network performs even worse with the capability of processing approximately 3 −7 transactions per second. Traditional centralized financial systems such as VISA can process over 1500 transactions per second which is significantly more than these public blockchains. Hence, in order to compete with such centralized financial systems blockchains need to be scalable both in terms of handling network load and processing transactions.

In our analysis we found only one relevant work which attempted to solve the blockchain scalability issue with the help of *formal modelling*. Dennis et al. (Dennis et al., 2016) proposed a solution for the improving the scalability blockchain networks. The authors propose a constant fixed size blockchain model called "rolling blockchain" which may solve the problem of exponential growth of blockchain networks by removing outdated information from the blockchain. They also presented a formal analysis of this proposed blockchain model, the results of which suggest that the deletion of data from the proposed model of the blockchain does not affect its security when compared to traditional blockchains. The provided solution might work for private blockchain networks but, it is hard ot visualize this proposal as a viable solution for public blockchains as deleting data from such blockchains might result in consensus failure and an eventual hard fork of the network.

4.3.5. Bug bounty

There have been several instances when vulnerabilities in smart contracts have been exploited by malicious users to gain financial advantage (Atzei et al., 2017). It is highly unlikely that a software, whether centralized, decentralized or distributed would be immune from any kind of security vulnerability. Recent research (Berger et al., 2016) and major centralized corporations such as Google and Facebook have shown that incentivizing the public for bug disclosure could be great way of reducing damages, costs and vulnerability patching times when a security vulnerability eventually surfaces. This process of incentivizing bug disclosure is known as *bounty hunting* or *bug bounty*.

In our analysis we found one relevant study that focused on *formally modelling* a framework (Hydra (Breidenbach et al., 2017)) for incentivizing bug and vulnerability disclosures in smart contracts. The framework is based on a program transformation that enables bug detection at runtime. The study also formally demonstrates that Hydra contracts incentivize bug disclosure, for bounties orders of magnitude below an exploit's value. Finally, he authors model strong bug-withholding attacks against on-chain bounties, and analyze Submarine Commitments, a generic defense to front-running that hides transactions in ordinary traffic.

4.3.6. Trustworthy data feeding

With *theorem proving* formalization technique as its backbone, Zhang et al. (Zhang et al., 2016) introduced Town Crier (TC) which is a tool that can act as a link between existing commonly trusted non-blockchain based websites and smart contracts. It utilizes a combination of a trusted hardware back-end and a blockchain front-end to provide trustworthy and authenticated data to smart contracts from HTTPS enabled websites.

4.3.7. Blockchain consensus protocol correctness

Being decentralized networks, blockchain nodes must reach an agreement regarding the state of a blockchain at any given time. This process of reaching an agreement is also known as consensus and is usually governed by sophisticated protocols such as Proof-of-Work in public blockchains such as Bitcoin and Ethereum. Given such complications, blockchain protocols may suffer from consensus attacks such as Byzantine General's attack, 51% attack etc. A blockchain consensus protocol must be resistant to such consensus attacks to maintain the integrity of the network.

Hence, as per our analysis only one study focused on verifying the correctness of blockchain consensus protocol with the help of *statistical model checking*. Chaudhary et al. (Chaudhary et al., 2015) investigated the Bitcoin protocol correctness and provide its formalization as a UPPAAL model. This study is inspired by the idea of double spending in Bitcoin and the lack of a third party to guard against the problem. The phenomena of double spending can take place if a user could prove to the majority that the blockchain without his previous payment is legitimate. The authors demonstrated that a malicious pool can overwhelm the computational power of the network and race against it to include a malicious network in the longest proof-of-work chain. The authors, with the help of *statistical model checking* also suggest that the probability of such attacks occurring on the Bitcoin network depends on the number of confirmations. A limitation of the paper could be the investigation of the double spending problem with different hash-rates.

4.4. RQ4: What domain specific languages (DSL) or formal/specification/general-purpose languages are proposed/used for formalizing smart contracts?

There were 13 (39%) relevant works that either propose a domain-specific/specification language or use a formal language for formalizing smart contracts. These studies are listed in Table 2 along with the language proposed/used for the formalization process.

12 out of the 13 papers presented in the table focus on contract functionality verification. Formal and specification languages such as Simplicity (O'Connor, 2017), SPESC (He et al., 2018), Findel (Biryukov et al., 2017), SCILLA (Sergey et al., 2018) and dSLAC (Scoca et al., 2017) were designed for formal verification and validation of smart contracts. These languages verify several functional and security properties that make certain bugs and vulnerabilities virtually impossible.

The F* programming language was mentioned in 2 relevant works. In fact, it is the only programming language that was mentioned more than once in all the related works collected for this review. F*, which is built for formal verification of programs, was used to verify security properties at the source code and Bytecode level (Bhargavan et al., 2016). Additionally, the F* proof assistant was also used to provide the initial semantics of the EVM (Grishchenko et al., 2018).

ETHERLITE (Luu et al., 2016) was proposed as a distilled conversion of Ethereum's EVM and Securify language (Tsankov et al., 2018) was used to power the Securify tool and define security patterns against which smart contracts were statically analyzed. PROMELA (Bai et al., 2018) was used for describing smart contracts which were later verified with SPIN model checking tool. Lem (Hirai, 2017), a language that can be compiled for several known theorem provers, was used to formally define all EVM instructions and prove safety properties of smart contracts. SMAC (Grossman et al., 2018) was used to provide theoretical development for contracts to detect Effective Callback Freedom.

Now moving on to the other side of the spectrum, the B language was the only one discussed in the relevant works that was not used to solve any code functionality verification issue. The B language was utilized to formally model a "rolling blockchain" to solve the scalability issues (Dennis et al., 2016) in traditional blockchains.

Table 2
Summary of the languages used/proposed in relevant works.

Ref.	Language proposed/used	Description of the language
(Dennis et al., 2016)	B language	B is an interpreted programming language which is an ancestor of the widely known C programming language and a descendent of BCPL (Basic/Bootstrap Combined Programming Language). B was built for system development instead of numeric computations. The only datatypes that B supports is a "word", which is used as both as integer and as memory address for dereferencing.
(Bai et al., 2018)	PROMELA	PROMELA (Process or Protocol Meta Language) is a modelling language meant for formal verification. PROMELA models are often analyzed with model checkers such as SPIN to verify correct system functionality.
(Bhargavan et al., 2016)[42]	F*	F* is a high-level general-purpose programming language built for the formal verification of its programs. F* programs can be converted to other high-level programming languages such as C, F# etc. after verification to provide functional correctness and security.
(Luu et al., 2016)	ETHERLITE	ETHERLITE is an extract of Ethereum's EVM. ETHERLITE is stack machine augmented with memory retaining some Ethereum-like features.
(Tsankov et al., 2018)	Securify language	Securify language is a Domain specific language for Securify platform. The language is used to express security patterns against which smart contracts are verified.
(O'Connor, 2017)	Simplicity	Simplicity is a functional language without loops or recursions for blockchain based systems. It has the following key features: • Provides upper bounds on amount of computation required with static analysis • Minimizes storage requirements and bandwidth. • Removes unused code at redemption time to improve privacy • Information outside the transaction are not accessible by programs. • Facilitates reasoning of programs with formal semantics
(Scoca et al., 2017)	dSLAC	dSLAC is a language which is used for the specification of smart contracts in the cloud computing domain.
(He et al., 2018)	SPESC	SPESC is a smart contract specification language. Smart contracts with SPESC can be defined in natural language like grammar which can make rights and obligations of parties involved in smart contracts unambiguous.
(Hirai, 2017)	Lem	Lem is a language that used for generating definitions from domain specific tools and porting them for proof in interactive theorem provers such as HOL Coq and Isabelle. Lem also provides features common to other functional programming languages such as logical constructs.
(Biryukov et al., 2017)	Findel	Findel is a financial domain specific language suited for blockchain applications that is purely declarative. Findel can be used to formalize smart contract clauses that makes them machine readable and unambiguous,
(Grossman et al., 2018)	SMAC	SMAC is an object-oriented programming language, which allows pass-by-value parameters with integer-typed local variables and data members. Every function in SMAC has a only one formal parameter namely "arg" and reruns a variable by assigning it a value with "ret".
(Sergey et al., 2018)	SCILLA	SCILLA is a translation target for high-level smart contract programming languages instead of being a high-level programming language irself. This translation is to be used for the application of formal verification of smart contracts before being further compiled to executable low-level code.

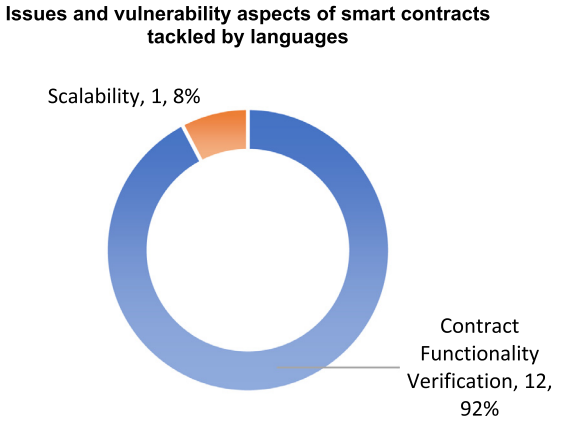


Fig. 6. The issues and vulnerability aspects of smart contracts focused by DSL/specification/general purpose languages in relevant works.

Fig. 6 provides a breakdown of the issue or vulnerability aspects of smart contract focused by the languages mentioned in the relevant works. The figure makes it clear that there was only one (B language) language that did not focus on smart contract functionality verification.

4.5. RQ5: What automated tools and frameworks are proposed in supporting state-of-the-art formalization approaches of smart contracts?

There were 15 formal code analyses, verification tools or frameworks presented in the relevant works. Most of these tools are based on symbolic execution and theorem proving methods. Other approaches such as formal modeling, model checking, and verification of certain properties of smart contracts with the usage of formal verification-based programming languages were also used to support these tools and frameworks. Table 3 presents the list of tools and frameworks and their descriptions as well as the formal techniques used to back these proposals.

The table suggests that there are several symbolic execution tools proposed in literature. All these tools are used to statically analyze the smart contracts for functional correctness and runtime safety. Hence, one could easily be misguided into thinking that all these tools are equally accurate and effective at smart contract vulnerability detection since, they all virtually perform the same tasks. But, this misconception was cleared by a recent empirical study performed by Parizi et al. (Parizi et al., 2018) which evaluated smart contracts security vulnerability detection tools (including Oyente, Mythril, Securify, and SmartCheck.[17]) from the detection accuracy and effectiveness points of view. Their results indicated that there are discrepancies between the tools on different security vulnerabilities.

5. Open issues and future directions

Blockchain has come a long way since its introduction in 2008 as a backbone to Bitcoin (Nakamoto, 2008). Over the last decade, blockchain has disrupted a wide range of industries such as Cloud computing, Finance, Internet of Things (IoT), Security and Privacy etc. But, this rise in popularity hasn't been without its fair share of problems such as security vulnerabilities, consensus protocol issues, privacy protection issues, scalability of blockchains etc. Re-

[17] https://tool.smartdec.net/.

Table 3
Automated tools and frameworks presented in the relevant works.

Tool/Framework Presented	Formal verification technique used	Tool description
FSolidM	Finite State Machine	The tool is built for designing smart contracts as Finite State Machines on a graphical interface. It can also generate automated Ethereum smart contract code.
Hydra	Formal Modelling	The Hydra framework is built to provide bug bounties for honest vulnerability disclosures.
Solidity* and EVM*	F* for program and bytecode verification	The framework analyzes and verifies both functional correctness and runtime safety of Solidity smart contracts with a functional programming language (F*).
Hawk	Formal Modelling	A smart contract platform which does not make financial transactions available publicly on the blockchain to maintain transactional privacy.
Oyente	Symbolic Execution	A tool to detect potential security vulnerabilities and bugs.
Framework for global system safety	Theorem Proving	The framework that implements several security primitives including: a notion of a proof object, hash-functions, a Fork Choice Rule and a Validator Acceptance Function.
Raziel	Theorem Proving	Raziel combines secure multi-party computation and proof-carrying code to provide privacy, correctness and verifiability guarantees for smart contracts on blockchains
Mythril	Symbolic Execution	A security analysis tool for Ethereum smart contracts, and its symbolic execution backend LASER-Ethereum
Securify	Symbolic Execution	A security analyzer for smart contracts that can prove contract behaviors as safe/unsafe with respect to a given property.
SASC	Symbolic Execution	SASC is a static analysis tool that can find potential logic risks and generate topological charts of invocation relationships.
Town Crier	Theorem Proving	Town Crier acts as a link between existing trusted non-blockchain based websites and smart contracts to provide authenticated data to smart contracts.
ZEUS	Theorem Proving, Model checking	ZEUS is a smart contract safety verifier that utilizes both symbolic model checking and abstract interpretation.
MAIAN	Symbolic analysis	MAIAN is a tool used to specify and reason about trace properties in smart contracts. The tool utilizes symbolic analysis and concrete validator to exhibit real vulnerabilities.
KEVM	Theorem Proving	Formal semantics of EVM in K
contractLarva	Finite State Machine	Runtime safety and verification tool for Solidity smart contracts on Ethereum

cently, formal methods have been proposed to mitigate many of these issues and vulnerabilities but, there is still long ways to go in the wide scale adoption of formal methods and blockchain itself. This section presents and discusses the challenges ahead in the intersects of formal methods and blockchain smart contracts:

Formal Testing – In the current age of software and technology, formal testing is an integral component of software development life cycle (SDLC). It is used in making sure that a software behaves and performs as per its specifications and requirements based on all possible inputs' conditions. But, when it comes to smart contract development, formal testing remains largely overlooked. This is particularly unfortunate as the process of formal testing has proven its worth when it comes to the development of safety critical systems and large financial systems such as Aircraft systems, medical systems, large banking software etc. Hence, it should not come as a surprise that formal testing can only bring fruitful results in making sure that a smart contract does what it is supposed to do based on its specification. Basic rules and guidelines of formal testing can easily be transitioned from safety critical systems to the smart contract domain for their verification. For instance, one can easily write formal test cases for Solidity smart contracts based on certain mathematical logic and rules by setting up a testing environment with truffle.[18] These test cases can be written in JavaScript and can be executed on a test network to check several properties of smart contracts. Additionally, we highly recommend exploring *Property-based testing* (Aichernig et al., 2017; Aichernig and Schumi, 2016) (pioneered by QuickCheck[19]) which is a technique for testing software that is aimed at verifying properties of a program that must hold true on every input, which is randomly generated from a wide range of possible inputs for that program. This technique can be used to check and verify several smart contract properties and behaviors based on a wide range of randomly generated inputs. For example, by feeding random input data to smart contracts developers can find bugs and edge cases that they would have missed otherwise, that would have resulted in failure or unexpected behavior of that particular smart contract.

Automated formal verification – Automated formal verification is a promising approach to detect bugs and other security vulnerabilities to guarantee the functional correctness of smart contracts. However, this solution adds two new challenges to the puzzle itself: 1) Limitation of memory and time of the machine that executes the contracts for formal verification. In several automated formalization approaches, bug and security issues are found by exploring as many execution states as possible. In this regard, for complex programs and protocols, the upper bound of execution time and runtime computer memory becomes a problematic limitation. Even though there are several techniques to reduce the number of states to be explored, these techniques are generally not sufficient for complex smart contracts. 2) Formalization correctness while using a formal verification tool. We often formalize the program, security patterns for correctness and environment of operation. The accuracy of formalization is crucial for the result of execution of the tool. However, the state-of-the-art work lacks an innovative tool that can check this accuracy.

From the above perspective, a good direction for the future of automatic formal analysis and verification could be the development of program patterns or templates and also tightly defining the language to limit the number of states. When it comes to implementation, the actual template for formalization is the protection profile. Moreover, for the verification of cryptographic protocols, evaluation reports can be used as templates which align with ISO/IEC 29128.[20]

Domain Specific Languages (DSLs) for Ethereum – Smart contracts written in Solidity on the Ethereum platform have suffered from multiple security vulnerabilities in the past few years (Atzei et al., 2017), which have resulted in both theft and gigantic financial losses. Most of these vulnerabilities could have been avoided with the help of formal analysis and verification of such smart contracts before deploying them on the blockchain. But, due to the fact that the state-of-the-art domain specific languages such

[18] https://truffleframework.com/.
[19] http://hackage.haskell.org/package/QuickCheck.
[20] https://www.iso.org/obp/ui/#iso:std:iso-iec:29128:ed-1:v1:en.

as Solidity is not built for formal verification itself makes matters worse as it is not a perfect DSL for writing safe smart contracts and is vulnerable to certain pitfalls.[21] Hence, even the most experienced smart contract developers can tend to leave behind security vulnerabilities or bugs in their code. This leaves developers and organization no choice but to turn to third party organizations, frameworks or tools for code analysis, reviews and audits which in itself may be expensive and can become a limitation for small organizations and individual developers. Thus, to mitigate this limitation, we propose introduction of a new domain specific language for developing smart contracts that fully supports formal analysis and verification. Additionally, defining such a domain specific language may also help in solving the limitations of state-of-the-art automated formal varication of smart contracts by reducing the number of states to be explored and providing a formalized template.

The issues related to smart contracts and blockchain-based systems is of great concern as these technologies have evolved into multibillion-dollar industries and hence, requires expertise from multiple research domains such as: networking, programming languages, formal methods and cryptography researchers etc. Real progress on these issues would be hard to achieve if the gaps between these research communities is not bridged. Hence, we hope this literature review brings these research gaps and issues under the limelight of these communities, so they can all coordinate for the improvement of smart contracts and blockchains in general.

6. Conclusions

In this systematic review, we presented and analyzed the state-of-the-art research and achievements concerning the formalization approaches in smart contracts. This was achieved with the help of a systematic paper selection protocol through which we identified 35 relevant works for this systematic review.

We analyzed the selected 35 studies based on four identified research questions (RQs): 1) **RQ1:** What formal methods and techniques are used in the verification and improvement of smart contracts? 2) **RQ2:** Which issues or vulnerability aspects of smart contracts do formalization approaches target? 3) **RQ3**: How do formalization approaches mitigate issues and vulnerabilities in smart contracts**?** 4) **RQ4**: What domain specific languages (DSL) or formal/specification/general-purpose languages are proposed/used for formalizing smart contracts? 5) **RQ5:** What automated tools and frameworks are proposed in supporting state-of-the-art formalization approaches of smart contracts?

The analysis conducted in this SLR indicates that theorem proving is the most commonly used formal technique for the verification and validation of smart contracts. Model checking, formal modelling and symbolic execution are other approaches which are also used for checking correctness and static analysis of smart contracts. Other formal approaches include the proposals of specification languages for writing and designing smart contracts such as Findel and SPESC. Specifically, there have been 12 languages (domain specific/specification/general purpose programming languages) proposed or used in 13 relevant works for the formalization of smart contracts on blockchains, while there were 15 automated formal verification tools/frameworks to provide support. Formal methods have most commonly been used to verify smart contract functionality. Security and privacy are other aspects of smart contracts that have been focused in the relevant works.

In addition to this analysis, we have also identified three major open issues currently looming in the smart contract domain: 1) Formal Testing, 2) Automated formal verification of smart contracts and 3) Domain Specific Languages (DSL) for the Ethereum platform. We have also provided possible solutions and future directions to mitigate these issues which would take the smart contract domain one step closer to being formally sound and robust.

Smart contracts on blockchains have faced several issues and vulnerabilities since the inception of the idea. Many of these vulnerabilities have caused significant financial damages. Formalization of smart contracts have helped in solving and mitigating many of these issues and vulnerabilities. But, as this infant domain matures, new vulnerabilities will surface which would require a combined effort from blockchain, cryptographic and formal methods research communities to solve. Hence, the motivation of our work was to bring these communities closer together by summarizing current achievements and highlighting open challenges for the improvement of smart contracts and blockchains in the future. We foresee this work to be an important piece of literature for conducting future empirical analysis of various smart contract formalization approaches, which could provide further valuable insight regarding the performance of these approaches as well as highlight key attributes such as their accuracy and computation cost.

Declaration of Competing Interest

The authors declared that there is no conflict of interest in this study.

Appendix A: Summary of the final set of works

Kim and Laskowski (Kim and Laskowski, 2017) stated principles regarding the possible evolution of smart contracts on blockchains, including: 1) Blockchains, whilst reducing the uncertainty of value exchange can also increase complexity from having to subsume work from third parties; 2) Smart contracts can be used to decrease the complexity arising from eliminating third parties with the use of blockchain; 3) The evaluation of smart contract models of several parties lowers uncertainty of value exchange whilst increasing transparency. Additionally, the authors also emphasize the use of formal models (mathematical, logical, or simulation-based) which can also increase transparency when evaluating or interpreting smart contracts of different parties.

Dennis et al. (Dennis et al., 2016) present a solution for the scalability limitation of blockchain based systems. The authors propose a constant fixed size blockchain called “rolling blockchain” which may solve the problem of exponential growth of state-of-the-art blockchain based systems. Additionally, they present a formal analysis of this proposed blockchain model, the results of which suggest that the deletion of data from the proposed model of the blockchain does not affect its security when compared to traditional blockchains.

Mavridou and Laszka (Mavridou and Laszka, 2018) present *FSolidM* framework for modeling smart contracts as Finite State Machine (FSM). Their proposed tool is rooted in rigorous semantics and provides an intuitive graphical editor, which supports automatic code generation. The authors also provide a set of plugins that developers can add to their contracts. These plugins are meant to implement: 1) security features for preventing known vulnerabilities such as unpredictable state and reentrancy; 2) common design patterns to facilitate functional correctness with complex functionality in smart contract development.

Breidenbach et al. (Breidenbach et al., 2017) introduce the Hydra Framework to incentivize honest disclosures of bugs and vulnerabilities in smart contracts. The framework is based on a program transformation that enables bug detection at runtime. The authors describe N-of-N-version programming (NNVP), which is

[21] https://solidity.readthedocs.io/en/v0.4.24/security-considerations.html#security-considerations.

a variant of N-version programming that detects divergences between multiple program instances. They also formally show that Hydra contracts incentivize bug disclosure, for bounties orders of magnitude below an exploit's value. Finally, they model strong bug-withholding attacks against on-chain bounties, and analyze Submarine Commitments, a generic defense to front-running that hides transactions in ordinary traffic.

Idelberger et al. (Idelberger et al., 2016) outline the technical and legal disadvantages of logic-based smart contracts in regards of usual activities featuring ordinary contracts. Additionally, the authors provide guidelines towards usage of such logic based smart contracts on blockchain based systems.

Bai et al. (Bai et al., 2018) introduce the application and critical issues in smart contracts. They propose the use of formal modelling and verification to produce smart contract models and verify smart contract properties respectively. A model verification tool can be used to verify important properties and correctness of smart contracts. A case study was presented with the help of a famous model checking tool SPIN to illustrate the verification process and effects.

Abdellatif and Brousmiche (Abdellatif and Brousmiche, 2018) propose a modeling formalism based on strong semantics for the verification of properties related to blockchain and smart contracts. They apply their formalism to a smart contract and model its interactions and behaviors with its execution environment. Also, they simulate these behaviors in the BIP framework and analyze the results with Statistical Model Checking (SMC), which shows the scenarios where smart contracts can be manipulated by malicious users.

Bhargavan et al. (Bhargavan et al., 2016) introduce a framework rooted on F* (a functional programming language for formal program verification) to verify and analyze the functional correctness and runtime safety of Solidity based smart contracts.

Kosba et al. (Kosba et al., 2016) introduce Hawk, which is a decentralized system for smart contracts that provides transactional privacy by not storing financial transactions in the clear on the blockchain. Hawk allows intuitive development of private smart contracts without requiring the implementation of cryptography. In fact, the compiler for the proposed solution automatically generates cryptographic protocol wherever the parties bound by contract interact with the blockchain. This is done using cryptographic technique known as zero-knowledge proofs (Sah et al., 2016).

Matsuo (Matsuo, 2017) proposes an innovative approach for the application of formal analysis and verification by considering technology layers of blockchain-based systems and their security concerns. These layers are identified as, implementation, protocol, and language. Moreover, the author proposes a framework for the application of formal analysis to these layers based on existing standards and results.

Hildenbrandt et al. (Hildenbrandt et al., 2018) propose KEVM, an executable formal specification of the EVM's bytecode stack-based language built with the K Framework. The KEVM is designed to provide a backbone for further rigorous formal analyses.

Luu et al. (Luu et al., 2016) outline several security issues where a malicious entity can gain profit by manipulating the execution of smart contracts. These issues highlight small gaps in the understanding of the distributed semantics of the underlying smart contract platform. Hence, as a counter measure, to make contracts less vulnerable they propose ways to enhance the operational semantics of the Ethereum platform. Additionally, the authors introduce Oyente which is a symbolic execution tool to detect security vulnerabilities.

Pîrlea and Sergey (Pîrlea and Sergey, 2018) present and implement a formal model of a distributed blockchain-based consensus protocol and its data structures (block forests). They also introduce several theorems regarding pure functional implementation of block forests. The authors work is based on several primitive security features like, hash-functions, a notion of a proof object, a Validator Acceptance Function, and a Fork Choice Rule. Furthermore, the authors characterize their assumptions regarding these components to prove the consensus of the global system.

Chaudhary et al. (Chaudhary et al., 2015) investigate the Bitcoin protocol correctness and provide its formalization as a UPPAAL model. This study is inspired by the idea of double spending in Bitcoin and the lack of a third party to guard against the problem. The phenomena of double spending can take place if a user could prove to the majority that the blockchain without his previous payment is legitimate.

Le et al. (Le et al., 2018) aim to determine the input conditions for which a smart contract terminates (or does not terminate) by proving conditional termination and non-termination statically. This is done by making sure that both, current state of the smart contract and the contract's input satisfy the termination condition to run on a proof carrying blockchain before the actual execution of the contract.

Raziel (Cerezo Sánchez, 2017) provides privacy, verifiability and correctness guarantees of smart contracts by combing proof carrying programs and secure multi-party computations. Additionally, the author introduces ways in which Zero-Knowledge Proofs of Proofs or Proof-Carrying program certificates can be used to prove the validity of smart contracts before execution to other parties in a private manner. Finally, the authors demonstrate ways in which miners can rewarded for producing pre-processed data for secure multi-party computations.

Ellul et al. (Ellul and Pace, 2018) demonstrate that standard techniques for runtime verification can be used for the verification of smart contracts, including a novel stake-based instrumentation technique which ensures that the violating party provides insurance for correct behavior. The author also discusses their partially implemented proof-of-concept tool called ContractLarva.

Tsankov et al. (Tsankov et al., 2018) propose Securify, which is a security vulnerability analyzer for smart contracts on the Ethereum platform. Securify is completely automated and can establish contract behaviors as safe or unsafe in regard to a given property. The underlying vulnerability analysis comprises of two steps: 1) the first step is the symbolic analysis of the contract dependency graph which is used to carve out precise semantic information from the program; 2) the second step is compliance checking and searching for violation patterns to capture sufficient conditions to prove if a property is valid or not.

Zhou et al. (Zhou et al., 2018) propose a method to detect potential security risks in smart contract programs. This is meant to perform two main functions: 1) Syntax topology analysis of smart contract invocation relationship, 2) Detection and location of logical risks, and presentation of results on a topological diagram. The authors also propose SASC which is a static analysis tool that is meant to find potential logic risks in smart contracts and generate topological diagram of invocation relationships.

O'Connor (O'Connor, 2017) introduces Simplicity, which is a combinator-based, typed, functional language meant to be used for blockchain based applications. Simplicity does not support loops and recursions. The objective of Simplicity is to build upon the current blockchain based languages such as Ethereum's EVM and Bitcoin Script by avoiding some critical problems that they face. Simplicity is rooted on formal denotational semantics which are defined in a general-purpose software proof assistant 'Coq'.

Scoca et al. (Scoca et al., 2017) introduce a formal language that can be used to specify interactions between requests and offers. The paper also presents an approach for the self-governed negotiation of smart contracts, which is used to analyze the cost and the necessary changes required to reach an agreement.

Mueller (Mueller, 2018) introduces Mythril, a security analysis tool based on symbolic execution backend for Ethereum smart contracts. The first part of the paper explains applications of symbolic execution and constraint solving in smart contract security analysis and verification. The second part showcases the use of symbolic analysis, static analysis, and control flow checking to discover real-world issues.

He et al. (He et al., 2018) introduce SPESC, which is a smart contract specification language. The specification of a smart contract can be defined in SPESC for collaborative design. This allows smart contracts to be specified like real world contracts written in natural language that can help in clearly defining the rights and obligations of all parties involved in the contract.

Liao et al. (Liao et al., 2017) introduce a platform that supports Behavior-Driven Development (BDD), deployment and testing of smart contracts for the Ethereum platform. The objective of the platform is to provide and resolve cross-cutting concerns across smart contract development life cycle.

Amani et al. (Amani et al., 2018) further extend EVM formalization in Isabelle/HOL with a strong program logic at the bytecode level. The authors structure bytecode sequences into blocks of straight-line code. Additionally, they create a program logic to reason about this formalization. The abstraction provided can help in controlling the complexity and cost of formal analysis and verification of Ethereum smart contracts.

Zhang et al. (Zhang et al., 2016) introduce Town Crier (TC). TC acts as a link between existing commonly trusted non-blockchain based websites and smart contracts. It utilizes a combination of a trusted hardware back-end and a blockchain front-end to provide authenticated data to smart contracts from HTTPS enabled websites.

Bigi et al. (Bigi et al., 2015) validate and analyze DSCP (Decentralized Smart Contract Protocol), a protocol for smart contracts idealized after BITHALO. Under the assumption of return maximization and perfect rationality, models such as Game theory can provide analysis of the behavior of the players in the game/protocol. These models also provide conditions for agreeing on the contract. Moreover, the simulation and modeling aspects of game theory presented in the paper are supported by formal methods for contract functionality verification purposes.

Kalra et al. (Kalra et al., 2018) introduce ZEUS, which is a framework that validates and verifies the fairness and correctness of smart contracts. ZEUS utilizes the power of symbolic model checking and abstract interpretation in addition to the usage of constrained horn clauses to verify contract safety. The authors have also built a prototype of their framework for both Fabric and Ethereum blockchain platforms.

Sergey and Hobor (Sergey and Hobor, 2017) explore similarities between classical problems of shared-memory concurrency and multi-transactional behavior of Ethereum smart contracts. They examine and analyze two examples from the Ethereum blockchain based on "how they are vulnerable to bugs that are closely reminiscent to those that often occur in traditional concurrent programs". This description of contracts-as-concurrent-objects provides a deeper knowledge of potential smart contract threats. The results also introduce improved practices with the application of formal verification techniques for developing smart contracts.

Grishchenko et al. (Grishchenko et al., 2018) provide initial semantics of EVM bytecode, which are then formalized in F* proof assistant. The authors also provide validation of the resulting executable code against the Ethereum test suite. Additionally, they provide formal definition of several security properties for smart contracts.

Hirai (Hirai, 2017) provides the definition of EVM in Lem, which is a language that can be compiled for several known theorem provers. Using the provided definition, the author provides proofs for several safety properties of Ethereum smart contracts in Isabelle/HOL theorem prover.

Biryukov et al. (Biryukov et al., 2017) propose Findel, which is a financial domain-specific language for blockchain networks. The authors implement an Ethereum smart contract that measures the cost of operation of a marketplace for Findel contracts. They also introduce the problems related to modeling and developing financial agreements on decentralized networks.

Nikolic et al. (Nikolic et al., 2018) propose and implement MAIAN, a tool which is based on symbolic analysis and concrete validator to exhibit various security exploits. Their proposal can also be used for precisely specifying and reasoning about trace properties.

Grossman et al. (Grossman et al., 2018) present Effective Callback Freedom, a general correctness condition for callbacks which enables modular reasoning in environments with local-only mutable states such as Ethereum. The authors also show that their work can be used to mitigate bugs without severely limiting programming style on the Ethereum platform. These bugs can be dynamically checked with low runtime overhead.

Finally, Sergey et al. (Sergey et al., 2018) introduce SCILLA that allwos extensive interaction patterns with seperation of communication aspect of smart contracts and a programming component principled with strong semantics open for formal verification. SCILLA is supposed to be a translation target for high-level smart contract programming languages instead of being a high-level programming language irself. This translation is to be used for the application of formal verification of smart contracts before being further compiled to executable low-level code.

References

Abdellatif, T., Brousmiche, K.-.L., 2018. 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.

Aichernig, B.K., Marcovic, S., Schumi, R., 2017. Property-Based testing with external test-case generators. In: 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 337–346.

Aichernig, B.K., Schumi, R., 2016. Property-Based testing with fscheck by deriving properties from business rule models. In: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 219–228.

Amani, S., Bégel, M., Bortin, M., Staples, M., 2018. 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.

Atzei, N., Bartoletti, M., Cimoli, T., 2017. A survey of attacks on ethereum smart contracts (SoK). In: International Conference on Principles of Security and Trust, pp. 1–24.

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

Berger, P., Hennig, P., Bocklisch, T., Herold, T., Meinel, C., 2016. A journey of bounty hunters: analyzing the influence of reward systems on stackoverflow question response times. In: 2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI), pp. 644–649.

Bhargavan, K., et al., 2016. Formal verification of smart contracts: short paper. In: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, pp. 91–96.

Bigi, G., Bracciali, A., Meacci, G., Tuosto, E., 2015. Validation of decentralised smart contracts through game theory and formal methods. In: Bodei, C., Ferrari, G., Priami, C. (Eds.), Programming Languages with Applications to Biology and Security: Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday, Cham. Springer International Publishing, pp. 142–161.

Biryukov, A., Khovratovich, D., Tikhomirov, S., 2017. Findel: secure derivative contracts for ethereum. In: Financial Cryptography and Data Security, pp. 453–467.

L. Breidenbach, P. Daian, F. Er, and A. Juels, "Enter the hydra: towards principled bug bounties and exploit-resistant smart contracts * the initiative for cryptocurrencies and contracts (IC3)," vol. 2017, 2017.

Cerezo Sánchez, D., 2017. Raziel: private and verifiable smart contracts on blockchains. IACR Cryptol. ePrint Arch. 1–56.

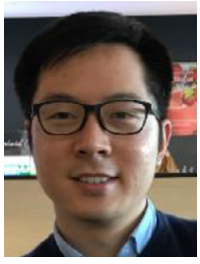
Chaudhary, K., Fehnker, A., van de Pol, J., Stoelinga, M., 2015. Modeling and verification of the bitcoin protocol. Electron. Proc. Theor. Comput. Sci. 196 (Mars), 46–60.

Conoscenti, M., Vetro, A., De Martin, J.C., 2016. Blockchain for the internet of things: a systematic literature review. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), pp. 1–6.

Cuccuru, P., 2017. Beyond bitcoin: an early overview on smart contracts. Int. J. Law Inf. Technol. 25 (3), 179–195.
Dennis, R., Owenson, G., Aziz, B., 2016. A temporal blockchain: a formal analysis. In: Proc. - 2016 Int. Conf. Collab. Technol. Syst. CTS 2016, pp. 430–437.
Destefanis, G., Marchesi, M., Ortu, M., Tonelli, R., Bracciali, A., Hierons, R., 2018. Smart contracts vulnerabilities: a call for blockchain software engineering? In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), pp. 19–25.
Ellul, J., Pace, G.J., 2018. Runtime verification of ethereum smart contracts. In: 2018 14th European Dependable Computing Conference (EDCC), pp. 158–163.
Grishchenko, I., Maffei, M., Schneidewind, C., 2018. A semantic framework for the security analysis of ethereum smart contracts. In: Principles of Security and Trust, pp. 243–269.
Grossman, S., et al., 2018. Online detection of effectively callback free objects with applications to smart contracts. In: Proc. ACM Program. Lang., 2.
He, X., Qin, B., Zhu, Y., Chen, X., Liu, Y., 2018. SPESC: a specification language for smart contracts. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 1, pp. 132–137.
Hildenbrandt, E., et al., 2018. KEVM: a complete formal semantics of the ethereum virtual machine. In: 2018 IEEE 31st Computer Security Foundations Symposium (CSF), pp. 204–217.
Hirai, Y., 2017. Defining the ethereum virtual machine for interactive theorem provers. Financ. Cryptogr. Data Secur. 520–535.
Idelberger, F., Governatori, G., Riveret, R., Sartor, G., 2016. Evaluation of logic - Based Smart contracts for blockchain systems. In: RuleML 2016, 1, pp. 1–17.
Kalra, S., Goel, S., Dhawan, M., Sharma, S., 2018. ZEUS: analyzing safety of smart contracts. NDSS Symp.
Kim, H., Laskowski, M., 2017. A perspective on blockchain smart contracts: reducing uncertainty and complexity in value exchange. In: 2017 26th International Conference on Computer Communication and Networks (ICCCN), pp. 1–6.
Kitchenham, B., 2004. Procedures For Performing Systematic Reviews 33, 28 no. TR/SE-0401.
Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering. Engineering 2, 1051.
Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C., 2016. Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 839–858.
Le, T.C., Xu, L., Chen, L., Shi, W., 2018. Proving conditional termination for smart contracts. In: Proceedings of the 2Nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts, pp. 57–59.
Legay, A., Delahaye, B., Bensalem, S., 2010. Statistical model checking: an overview. Int. Conf. Runtime Verif. 122–135.
Liao, C.F., Cheng, C.J., Chen, K., Lai, C.H., Chiu, T., Wu-Lee, C., 2017. Toward a service platform for developing smart contracts on blockchain in bdd and tdd styles. In: 2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA), pp. 133–140.
Liu, C., Liu, H., Cao, Z., Chen, Z., Chen, B., Roscoe, B., 2018. ReGuard: finding reentrancy bugs in smart contracts. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), pp. 65–68.
Luu, L., Chu, D.-.H., Olickel, H., Saxena, P., Hobor, A., 2016. Making smart contracts smarter. In: Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS'16, pp. 254–269.
Matsuo, S., 2017. How formal analysis and verification add security to blockchain-based systems. In: Proc. 17th Conf. Form. Methods Comput. Des. FMCAD 2017, pp. 1–4.
Mavridou, A., Laszka, A., 2018. Tool demonstration: fSolidM for designing secure ethereum smart contracts. In: Principles of Security and Trust, pp. 270–277.
Mueller, B., 2018. Smashing ethereum smart contracts for fun and real profit. In: Hacker in the Box (HITBAMS 2018), pp. 1–54.
Nakamoto, S., 2008. Bitcoin: a peer-to-peer electronic cash system. Www.Bitcoin.Org. [Online]. Available https://bitcoin.org/bitcoin.pdf . [Accessed: 30-Jan-2019].
Nikolic, I., Kolluri, A., Sergey, I., Saxena, P., Hobor, A., 2018. Finding the greedy, prodigal, and suicidal contracts at scale. [Online]. Available: http://arxiv.org/abs/1802.06038.
O'Connor, R., 2017. Simplicity: a new language for blockchains. In: Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security (PLAS '17), pp. 107–120.
Parizi, R.M., Amritraj, Dehghantanha, A., 2018. Smart contract programming languages on blockchains: an empirical evaluation of usability and security. In: Blockchain – ICBC 2018, pp. 75–91.
Parizi, R.M., Dehghantanha, A., Choo, K.K.R., Singh, A., 2018. Empirical vulnerability analysis of automated smart contracts security testing on blockchains. 28th Annual International Conference on Computer Science and Software Engineering (CASCON'18) .
Pîrlea, G., Sergey, I., 2018. Mechanising blockchain consensus. In: Proc. 7th ACM SIGPLAN Int. Conf. Certif. Programs Proofs - CPP 2018, pp. 78–90.
Reyna, A., Martín, C., Chen, J., Soler, E., Díaz, M., Nov. 2018. On blockchain and its integration with IOT. challenges and opportunities. Futur. Gener. Comput. Syst. 88, 173–190.
Sah, C.P., Jha, K., Nepal, S., 2016. Zero-knowledge proofs technique using integer factorization for analyzing robustness in cryptography. In: 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 638–642.
Scoca, V., Uriarte, R.B., Nicola, R.D., 2017. Smart contract negotiation in cloud computing. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), pp. 592–599.
Seebacher, S., Schüritz, R., 2017. Blockchain technology as an enabler of service systems: a structured literature review. Explor. Serv. Sci. 12–23.
Sergey, I., Hobor, A., 2017. A concurrent perspective on smart contracts. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 10323, 478–493 LNCS.
Sergey, I., Kumar, A., Hobor, A., 2018. Scilla: a smart contract intermediate-level LAnguage. [Online]. Available: http://arxiv.org/abs/1801.00687.
Tsankov, P., Dan, A., Cohen, D.D., Gervais, A., Buenzli, F., Vechev, M., 2018. Securify: practical security analysis of smart contracts. [Online]. Available: http://arxiv.org/abs/1806.01143.
Yli-huumo, J., Ko, D., Choi, S., Park, S., Smolander, K., 2016. Where is current research on blockchain technology ?— a systematic review. PLoS ONE.
Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E., 2016. Town crier: an authenticated data feed for smart contracts. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 270–282.
Zhou, E., et al., 2018. Security assurance for smart contract. In: 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pp. 1–5.

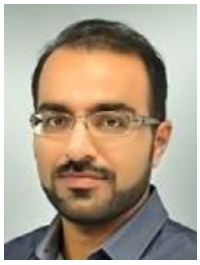
Amritraj Singh is a research assistant and a graduate student pursuing a master's degree in Software Engineering at the College of Computing and Software Engineering (CCSE) at Kennesaw State University. He received his bachelor's degree in Electrical and Electronics Engineering in 2016 from Visvesvaraya National Institute of Technology (VNIT), Nagpur, India. He is a member of blockchain community IEEE and his research interests include blockchain-based systems, decentralized applications, smart contracts programming and security, software development, and software quality engineering and assurance. He has previously published articles in conferences proceedings such as ICBC, FIE, and CASCON.

Reza M. Parizi is with Kennesaw State University. He is a consummate technologist and software security researcher with an entrepreneurial spirit. He is the member of Cyber Scientist - a community of cybersecurity researchers, as well as IEEE, IEEE Blockchain Community, IEEE Computer Society and ACM. Prior to joining KSU, he was an Associate Professor at New York Institute of Technology. He received a Ph.D. in Software Engineering in 2012. His research interests are R&D in blockchain, smart contracts, and emerging issues the practice of secure software-run world applications.



Qi Zhang received the Ph.D. degree in computer science from Georgia Institute of Technology, Atlanta, USA, in 2017. He is currently a Research Staff Member in IBM Thomas J. Watson Research Center. His-research interests include Blockchain systems, Cloud computing, big data processing, and distributed systems. He published research articles in referred journals and conference proceedings such as IEEE TC, IEEE TSC, ACM CSUR, VLDB, SC, HPDC, IEEE ICDCS, IEEE ICWS, IEEE CLOUD, ICBC. Dr. Zhang received the top 5 picks award in IEEE ICWS 2017. He served as a program committee member for IEEE Blockchain 2018.

Kim-Kwang Raymond Choo holds the Cloud Technology Endowed Professorship at University of Texas at San Antonio (UTSA). He was named the 2016 Cybersecurity Educator of the Year–APAC, and he and his team won Germany's University of Erlangen-Nuremberg's Digital Forensics Research Challenge 2015. He is the recipient of the 2019 IEEE Technical Committee on Scalable Computing (TCSC) Award for Excellence in Scalable Computing (Middle Career Researcher), 2018 UTSA CoB Col. Jean Piccione and Lt. Col. Philip Piccione Endowed Research Award, IEEE TrustCom 2018 and ESORICS 2015 Best Paper Awards, 2014 Australia New Zealand Policing Advisory Agency's Highly Commended Award, Fulbright Scholarship in 2009, 2008 Australia Day Achievement Medallion, and British Computer Society's Wilkes Award in 2008.



Ali Dehghantanha is the director of Cyber Science Lab in the School of Computer Science, University of Guelph (UofG). Prior to joining UofG, he has served as a Sr. Lecturer in the University of Sheffield, UK and as an EU Marie-Curie International Incoming Fellow at the University of Salford, UK. He has a PhD in Security in Computing and a number of professional certifications including CISSP and CISM. His-main research interests are malware analysis and digital forensics, IoT security and application of AI in the Cyber Security.