Reliable Smart Contracts: State-of-the-art, Applications, Challenges and Future Directions

César Sánchez[1], Gerardo Schneider[2], and Martin Leucker[3]

[1] IMDEA Software Institute, Madrid, Spain
cesar.sanchez@imdea.org
[2] University of Gothenburg, Sweeden
gerardo@cse.gu.se
[3] University of Luebeck, Germany
leucker@isp.uni-luebeck.de

Abstract. The popularization of blockchain technologies have brought a sudden interest in software that executes on top of blockchain, the so called smart contracts, with many potential applications, from financial contracts to unforgeable elections. Smart contracts are pieces of software that manipulate the shared data stored in the blockchain, with the promise that no central authority can forge or manipulate the execution or its results. This promise also involves an important risk, as well-intentioned users cannot easily roll-back undesired effects due to errors, or prevent other users from finding and exploiting loop-holes in deployed smart contracts. In this ISoLA track we seek to attract a variety of experts in the different aspects of smart contract reliability, discuss the state of the art and explore avenues for future research.

1 Blockchains and Smart-Contracts

Blockchain is a global distributed ledger, or database, running on millions of devices where not just information but anything of value (money, music, art, intellectual property, votes, etc.) can be moved and stored with a certain level of security and privacy. The blockchain trust is established through mass (distributed) collaboration. Blockchain has the potential to change in a fundamental way how we deal, not only with financial services, but also with more general applications, improving transparency and regulation. Many applications of blockchain have been proposed, starting with cryptocurrencies like *Bitcoin* [5], and more sophisticated programmable behaviors based on smart contracts, as introduced in Ethereum [1].

Smart contracts are software programs that, once deployed, execute autonomously on a blockchain. Smart contracts are openly stored in the blockchain (they can be read and used by anyone), and—as everything else in blockchains—they are permanent and cannot be altered, not even by their creator. The execution of smart contract is performed in the blockchain network by "workers" (commonly known as miners) that earn some crypto money in return for the execution of a smart contract. A smart contract typically offers several functions

which can be invoked by anyone via the Internet's API of the blockchain. As part of this functionality, users can transfer (crypto) value and any other kind of information, to other users via the contract. The contract will manage these invocations and execute the corresponding instructions that manipulate the local book-keeping of data (including the cryptocurrency) and can transfer data or value to the corresponding users. Underlying to the smart contract's idea is the description, and prescription, of an agreement between different parties in order to automate the regulated exchange of value and information over the internet. Given their implementation over a blockchain, these smart contracts are immutable and openly checkable.

The promise of smart contract technology is to diminish the costs of contracting, enforcing contractual agreements, and making payments, while at the same time ensuring trust and compliance, all in the absence of a central authority. It is not clear, however, whether this promise can be delivered given the current state-of-the-art and state-of-practice of smart contracts. In particular, some recent multi-million Ethereum bugs [3, 4, 6] just witness some the risks involved in any kind of software and that the community were afraid of. It is not clear what contracts mean and how to ensure that they are reliable and error free, which are incarnations in the smart contract world of classical issues in software reliability. This calls for better programming languages specifically for smart contracts with stronger security and privacy guarantees, or to develop mechanisms for the verification of smart contracts to guarantee reliability, security and privacy concerns.

In the track we have collected new results and discussions related to:

- Research on different languages for smart contracts (e.g., Solidity [2]), including their expressivity and reasoning methods.
- Research on the use of formal methods for specifying, validating and verifying smart contracts (both statically and at runtime).
- Surveys and state-of-knowledge about security and privacy issues related to smart contract technologies.
- New applications based on smart contracts.
- Description of challenges and research directions to future development for better smart contracts.

2 Summary of Selected Articles

In this section, we briefly summarize the articles invited to the track "Reliable Smart Contracts: State-of-the-art, Applications, Challenges and Future Directions".

- **Smart Contracts and Opportunities for Formal Methods**, by Andrew Miller, Zchicheng Cai and Somesh Jha, provides a background on smart contracts and surveys existing smart contract languages and verification tools. The paper also present some verification challenges for the formal methods community.

- **Contracts over Smart Contracts: Recovering from Violations Dynamically**, by Gordon Pace, Christian Colombo and Joshua Ellul, discuss the problem of checking and ensuring correctness of smart contracts, which is a challenging problem as smart contracts cannot easily be changed once in the blockchain. A variety of runtime monitoring, verification, recovery, enforcing as well as design patterns are discussed to achieve correct behaviour.
- **Security Analysis of Smart Contracts in Datalog**, by Petar Tsankov, briefly introduces *Securify*, a fully automated security analyzer for Ethereum smart contracts. Securify symbolically encodes relevant control- and data-flow dependencies in stratified Datalog and uses scalable Datalog solvers to derive relevant semantic facts about the smart contract. It allows the possibility to check compliance and violation patterns to capture sufficient conditions for proving if a given security property holds or not.
- **Temporal Properties of Smart Contracts**, by Ilya Sergey, Amrit Kumar and Aquinas Hobor attacks the static verification of smart contracts using theorem proving. The approach consists on using an intermediate representation language, called Scilla, specifically designed for verification. Scilla borrows well-known abstractions from static verification like communicating automata state-transition systems and temporal property templates, and separate the functional computation from the effects on the state of the contract and the underlying blockchain. Verification activities are ultimately carried out as proofs written in the Coq proof system.
- **Temporal Aspects of Smart Contracts for Financial Derivatives** by Christopher Clack and Gabriel Vanca, presents the problem of modeling over-the-counter financial derivative contracts. The paper first introduces terminology to differentiate the different uses of the term "contract" (smart legal contract vs smart contract code), and then argue that a formal language that handles over-the-counter financial derivatives must include temporal, deontic and operational aspects and sketches a potential direction for such a formalism.
- **Marlowe: financial contracts on blockchain**, by Simon Thompson and Pablo Lamela Seijas, explores the design of a DSL, called Marlowe, targeted at the execution of financial contracts on blockchains. Domain Specific Languages, compared to general languages, have the potential of being simpler for humans to comprehend programs, and to prevent ambiguities and incomprehensible behaviors. The paper presents an executable semantics of Marlowe implemented in Haskell, examples of Marlowe, and describe a tool that allows users to interact in-browser with simulations of Marlowe contracts.
- **SMT-based Verification of Solidity Smart Contracts**, by Leonardo Alt and Christian Reitwiessner presents a method to perform static analysis checks for Ethereum smart contracts written in Solidity. Since Solidity contracts are compiled into bytecode for the Ethereum Virtual Machine (EVM), the static analysis that the authors propose is integrated in the compiler. This technique is automatic and readily usable by developers, requiring no additional knowledge of an intermediate representation or language.

– **A Language-Independent Approach To Smart Contracts Verification**, by Xiaohong Chen, Daejun Park and Grigore Rosu, present an approach of using the so-called *language independent formal methods* for the verification of smart contracts. Language independent methods consists of using a sophisticated engine to formally encode operational semantics that can be used to formally derive interpreters, debuggers, symbolic executors, model checkers, etc. In this particular case, the system proposed uses the K-framework to encode the formal semantics of the Ethereum Virtual Machine.
– **Towards Adding Variety to Simplicity**, by Nachiappan Valliappan, Solène Mirliaz, Elisabet Lobo Vesga andAlejandro Russo, considers the smart contract language Spimplicity for the Etherum platform which allows fast analysis of resource consumption. It is argued that by using a categorical semantics, new combinators can easily be added to Simplicity enhancing the structure of corresponding contracts. Moreover, it is argued that the concept of functions should be added to Simplicity.
– **Fun with Bitcoin smart contracts**, by Massimo Bartoletti, Tiziana Cimoli and Roberto Zunino gives an introduction to BitML, a Domain Specific Language (DSL) for smart contracts based on process algebra, that compiles into Bitcoin. The computational soundness of the BitML compiler guarantees that the execution of the compiled contract is coherent with the semantics of the source specification, even in the presence of adversaries.
– **Computing Exact Worst-Case Gas Consumption for Smart Contracts** by Matteo Marescotti, Martin Blicha, Antti Hyvarinen, Sepideh Asadi and Natasha Sharygina, study the problem of calculating the resources needed to execute Ethereum smart contracts. In the context of Ethereum, the resource is called gas, to be paid to the miners that maintain the block-chain, which depends on the execution trace of the contract. This study presents two methods for determining the exact worst-case gas consumption of an Ethereum execution using methods borrowed by symbolic model checking. Additionally, they identify the challenges and sketch potential solutions.
– **Blockchains as Kripke Models: an Analysis of Atomic Cross-Chain Swap**, by Yoichi Hirai, considers the problem of proving the correctness of blockchain artefacts. To this end, the atomic cross-chain swap protocol is studied, a form of epistemic logic is introduced as proof vehicle, the protocol is analyzed and too weak and sufficient assumptions for the protocol to be correct are discussed.

References

1. Ethereum. `https://www.ethereum.org`.
2. Solidity. `http://solidity.readthedocs.io/en/develop/introduction-to-smart-contracts.html`.
3. A. Hern. $300m in cryptocurrency accidentally lost forever due to bug. Appeared at The Guardian `https://www.theguardian.com/technology/2017/nov/08/cryptocurrency-300m-dollars-stolen-bug-ether`, Nov 2017.

4. Mix. Ethereum bug causes integer overflow in numerous erc20 smart contracts (update). Appeared at HardFork https://thenextweb.com/hardfork/2018/04/25/ethereum-smart-contract-integer-overflow/, 2018.
5. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. White Paper https://bitcoin.org/bitcoin.pdf, 2009.
6. Haseeb Qureshi. A hacker stole $31M of Ethereum — how it happened, and what it means for Ethereum. Appeared at FreeCodeCamp https://medium.freecodecamp.org/a-hacker-stole-31m-of-ether-how-it-happened-and-what-it-means-for-ethereum-9e5dc29e33ce, 2017.