Towards the Automated Verification of Smart Contracts [*]

Chiara Braghin, Elvinia Riccobene, Simone Valentini
Università degli Studi di Milano

Ethereum [5] was the first blockchain implementing the the idea of smart contracts envisioned by Nick Szabo back in 1996 [9]. The combination of a distributed ledger with the ability to execute immutable code without the need of a trusted third party when predetermined conditions are met opened to a lot of potential new use cases. Nowadays, smart contracts are used to automate and enforce contractual terms between two or more parties entering the agreement, in scenarios ranging from financial trades and services, to insurance, credit authorization, legal processes, and even crowdfunding agreements.

As a consequence, both the number of blockchains supporting smart contracts and, unfortunately, the number of attacks have grown with expensive consequences. For example, the infamous DAO exploit [1] resulted in the loss of almost $60 million worth of Ether. Logic errors tend to be one of the most common types of blockchain smart contract vulnerabilities. They may include typographical errors, misinterpretation of specifications, or more serious programming errors that decrease the security of smart contracts. Since smart contracts cannot be deleted by default, and interactions with them are irreversible, the only remedy for this type of errors is to hard-fork the blockchain and revert one of the forks back to the state before the incorrect smart contract was executed. However, this remedy itself is devastating as it defeats the core values of blockchain, such as immutability and decentralized trust.

Thus, it is of paramount importance to guarantee the correctness of contracts at deployment time in order to avoid catastrophic events and huge loss of money. Formal verification, which uses formal methods for specifying, designing, and verifying programs, has been used for years to ensure correctness of critical hardware and software systems. When used for smart contracts, formal verification can prove that a contract's business logic meets a predefined specification. With respect to other techniques such as testing or code inspection, formal verification gives stronger guarantees that a smart contract is functionally correct. This comes at the expense of other challenges such as (i) a difficult modelling language, making the writing of the model error-prone as well; (ii) a verification process that might require user interaction and knowledge of the tool's internal; (iii) the verification results difficult to interpret and to bind to the original protocol; (iv) scalability.

Several studies have been performed to identify current topics and challenges of smart contracts and different approaches have been proposed, especially towards the formal verification of Solidity smart contracts [6]. However,

[*]This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NextGenerationEU.

the proposed approaches either target a limited number of vulnerabilities, or use complex notations that may lead to incorrect formal specifications [7, 8].

We are currently investigating the usage of the Abstract State Machine formal method [3, 2] for the specification and verification of smart contracts. ASMs offer several advantages as formal method: (1) due to their *pseudo-code format*, they can be easily understood by practitioners and can be used for *high-level programming*; (2) they allow for system specification at any desired *level of abstraction* and with different computational paradigms, from a *single* agent to distributed *multiple* agents; (3) they are *executable models*, so they are suitable also for lighter forms of model analysis such as simple simulation to check model consistency w.r.t. system requirements; (4) the ASMETA framework allows for an integrated usage of tools for different forms of model analysis (e.g., ASM models can be validated in terms of model simulation, animation, and scenarios execution. It is also possible to verify properties expressed in temporal logic by means of model checking).

We have been successfully using ASMs for the verification of security protocols [4], thus we expect to be able to apply the lessons learned also to the smart contract context, and prove their correctness also against a large number of different attacks and attackers.

References

[1] N. Alchemy. A short history of smart contract hacks on ethereum: A.k.a. why you need a smart contract security audit, 2019.

[2] E. Börger and A. Raschke. *Modeling Companion for Software Practitioners*. Springer, 2018.

[3] E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Verlag, 2003.

[4] C. Braghin, M. Lilli, and E. Riccobene. A model-based approach for vulnerability analysis of iot security protocols: The z-wave case study. *Computers Security*, 127:103037, 2023.

[5] E. Foundation. Ethereum, 2017.

[6] I. Garfatta, K. Klai, W. Gaaloul, and M. Graiet. A Survey on Formal Verification for Solidity Smart Contracts. In *Proceedings of the 2021 Australasian Computer Science Week Multiconference*, ACSW '21, New York, NY, USA, 2021. Association for Computing Machinery.

[7] G. Madl, L. Bathen, G. Flores, and D. Jadav. Formal Verification of Smart Contracts Using Interface Automata. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 556–563, 2019.

[8] A. Mavridou and A. Laszka. Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach. In S. Meiklejohn and K. Sako, editors, *Financial Cryptography and Data Security*, pages 523–540, Berlin, Heidelberg, 2018. Springer Berlin Heidelberg.

[9] N. Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 18(2):28, 1996.