Formal Verification of Smart Contracts: A Review

Stone Fang (Student ID: 19049045)

fnk7060@autuni.ac.nz

For Assessment 1 Question 3: Formal Methods in IT

of ENSE803 Formal Specification and Design

Computers and Information Sciences

Auckland University of Technology

Auckland, New Zealand

May 17, 2020

Blockchain enables the trade of crypto-currency without a trusted third party by peer-verification of transactions. Beyond that, smart contract is able to execute arbitrary programs written in Turing-complete languages on decentralised blockchain platforms (Tsankov et al., 2018). The most popular platform is Ethereum, which runs smart contracts on Ethereum Virtual Machine (EVM). However, due to the immutability and irreversibility of blockchain, unexpected operations in smart contracts can cause severe consequences, and crypto-currencies worth millions of USD have been stolen by attacks exploiting such vulnerability. Therefore, it is crucial to prove the correctness of a smart contract before its publishing by rigorous mathematical models, that is, formal verification (Murray & Anisi, 2019). However, the Turing-completeness of smart contract is a double-edged sword, posing the main challenge of verification of arbitrary behaviour and properties (Tsankov et al., 2018). A variety of studies and surveys on formal verification of smart contracts have been conducted, which can be divided into two categories: theorem proving and model checking (Harz & Knottenbelt, 2018; Murray & Anisi, 2019).

Theorem Proving methods express systems by mathematical formulas, and expected properties are also represented formally. Then the verification is performed by theorem prover (Murray & Anisi, 2019). In recent publications, formal verification tools including $\mathbb{K}$, Lem, Coq, and F^* are used. However, theorem proving approaches only reach partially automation (Harz & Knottenbelt, 2018). Amani, Bégel, Bortin, and Staples (2018) conducted a verification method for EVM bytecode based on Isabelle/HOL theorem prover. The EVM bytecode is first decompiled and split into basic blocks by extraction of Control Flow Graphs (CFG), followed by a Hoare-style program logic that proceeds these basic blocks with an EVM formal model (Hirai, 2017) defined in Lem. After the functional specification and correctness theorem have been created, the verification will be executed by the Isabelle theorem prover. In order to demonstrate the feasibility of this method, a case study was conducted and further accelerated by automatic verification condition generator with the help of Isabelle tactics (Amani et al., 2018).

Model Checking approach represents systems as models, mostly finite-state machines, and then verified by model checkers checking if the model satisfies the specification. If it is found not to meet the desired property in some scenario, a counter example will be provided (Murray & Anisi, 2019). Compared to theorem proving approach, an advantage of model checking is full automation (Harz & Knottenbelt, 2018).An example is SECURIFY (Tsankov et al., 2018), a scalable and fully-automated security analyzer for smart contracts at EVM bytecode level. The bytecode is decompiled and semantic facts specified by Datalog are automatically inferred from dependency graph. It also employs a specialised domain-specific language (DSL) to define a set of security patterns including both violation and compliance ones. Through verifications of semantic facts with security patterns, the behaviours of smart contracts are classified into violation, warnings and compliance. It was also shown that the coverage has been significantly improved compared to existing methods. Finally, the effectiveness of SECURIFY has been convincingly demonstrated by auditing over 18k contracts with a publicly available implementation (https://securify.ch) (Tsankov et al., 2018).

Formal verification of smart contracts is a young but fast-developing and active area. Currently, most methods are only able to handle simple contracts instead of full features (Murray & Anisi, 2019), for instance, supporting only recursion but not loops. In addition, improvement on coverage and automation is a key task for useful verification tools. Moreover, these methods are academic research rather than out-of-box tools for end-users, so it will be beneficial to create automatic verification tools integrated into the development environment of target blockchain platform.

References

Amani, S., Bégel, M., Bortin, M., & Staples, M. (2018, January 8). 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). CPP 2018. doi:10.1145/3167084

Harz, D., & Knottenbelt, W. (2018, October 31). Towards safer smart contracts: A survey of languages and verification methods. *arXiv:1809.09805 [cs]*. arXiv: 1809.09805

Hirai, Y. (2017). Defining the Ethereum virtual machine for interactive theorem provers. In M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. Ryan, V. Teague, . . . M. Jakobsson (Eds.), *Financial cryptography and data security* (Vol. 10323, pp. 520–535). doi:10.1007/978-3-319-70278-0_33

Murray, Y., & Anisi, D. A. (2019, June). Survey of formal verification methods for smart contracts on blockchain. In *2019 10th IFIP international conference on new technologies, mobility and security (NTMS)* (pp. 1–6). doi:10.1109/NTMS.2019.8763832

Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Bünzli, F., & Vechev, M. (2018, January 15). Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security* (pp. 67–82). CCS '18. doi:10.1145/3243734.3243780