Smart, and also Reliable and Gas-Efficient, Contracts

Elvira Albert*†, Jesús Correas†, Pablo Gordillo†, Guillermo Román-Díez‡ and Albert Rubio*†
*Instituto de Tecnología del Conocimiento
†Complutense University of Madrid, Spain
‡Universidad Politécnica de Madrid, Spain
{ealberta,jcorreas,pabgordi,alberu04}@ucm.es, guillermo.roman@upm.es

***Index Terms*—Ethereum, Smart contracts, Resource Analysis, Verification, Safety, Optimization**

A smart contract is a software program that runs on top of a blockchain. It contains a collection of public functions that can be invoked within the transactions launched over the contract by parties interacting with it. Being computer programs, well-studied formal verification techniques can be applied to them. Indeed, smart contracts are a very interesting application domain for validation, verification and optimization techniques since (1) they are relatively small in size, hence the application of these techniques scales better than when applied to larger industrial code, (2) they are valuable (in the corresponding blockchain cryptocurrency), hence software bugs or inefficiencies can cause economical losses and there is much interest in formally proving their safety and security, and (3) they require proving new specific properties to ensure their reliability and efficiency.

The concrete context of our work is the Ethereum blockchain, one of Bitcoin's most prominent successors that adds a quasi Turing-complete language to develop the smart contracts. In Ethereum, replicated execution is implemented by means of the Ethereum Virtual Machine (EVM), which is a stack-based operational formalism, enriched with a number of primitives that allow contracts to invoke each other, to refer to the global blockchain state, and even to create new contract instances dynamically. The EVM provides a convenient compilation target, known as EVM bytecode, that multiple high-level programming languages (e.g., Solidity or Vyper) compile to. Our methods for verification and optimization work directly on EVM bytecode. This has a number of advantages: (1) the source code is not always available (e.g., the blockchain only stores the bytecode), (2) the information to be gathered in the analysis might be only visible at the level of bytecode (e.g., gas consumption is specified at the level of EVM instructions), and (3) the analysis results may be affected by optimizations performed by the compiler (thus the analysis/optimization should be done ideally after compilation). However, analyzing bytecode requires the decompilation of the EVM bytecode into a high-level representation on which the analysis can be defined: we use an extended version of Oyente [1] to recover the CFG and the EthIR framework [4] to obtain a rule-based representation of the bytecode.

This work was funded partially by the Spanish MCIU, AEI and FEDER (EU) project RTI2018-094403-B-C31/C33, the MINECO project TIN2015-69175-C4-2/3-R, and by the CM project S2018/TCS-4314.

We will focus on the verification of gas-related and safety properties whose main goals are to save resources, prevent vulnerabilities and avoid potential attacks to Ethereum smart contracts. As regards safety, we will present SAFEVM [2], a verification framework for Ethereum smart contracts that makes use of state-of-the-art verification engines for C programs. The main observation within the SAFEVM framework is that the `INVALID` bytecode of EVM is key for verification, since `INVALID` is executed both in assertion violations and several sources of fatal operations (e.g., out-of-bounds access, division by zero). Our verification approach consists in decompiling the EVM bytecode into a C program in which the `INVALID` operations are translated into calls to an `error` function so that its unreachability can be proven by C verification tools. As regards gas-related properties, we will present our framework GASOL [3], for the inference and optimization of the gas usage of smart contracts. GASOL works on a wide variety of *cost models*, e.g., cost models to measure only storage opcodes, to measure a selected family of gas-consumption opcodes following the Ethereum's classification, to estimate the gas of a selected program line, etc. After choosing a cost model and a function of interest, GASOL returns to the user a gas upper bound of the cost for this function. As the gas consumption is often dominated by the instructions that access the storage, GASOL uses the gas analysis to detect under-optimized storage patterns, and includes an (optional) automatic optimization of the selected function.

References

[1] Oyente: An Analysis Tool for Smart Contracts, 2018. https://github.com/melonproject/oyente.

[2] E. Albert, J. Correas, P. Gordillo, G. Román-Díez, and A. Rubio. SAFEVM: A Safety Verifier for Ethereum Smart Contracts. In D. Zhang and A. Møller, editors, *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis 2019 (ISSTA'19)*, ACM, pages 386–389, 2019.

[3] E. Albert, J. Correas, P. Gordillo, G. Román-Díez, and A. Rubio. GASOL: Gas Analysis and Optimization for Ethereum Smart Contracts. In *26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2020. Proceedings*, Lecture Notes in Computer Science, 2020. To appear.

[4] E. Albert, P. Gordillo, B. Livshits, A. Rubio, and I. Sergey. EthIR: A Framework for High-Level Analysis of Ethereum Bytecode. In *ATVA*, volume 11138 of *LNCS*, pages 513–520. Springer, 2018.

978-1-7281-5778-8/20/$31.00 ©2020 IEEE
DOI 10.1109/ICST46399.2020.00010