Verification Techniques for Smart Contracts in Agda

Fahad F. Alhabardi[1], Arnold Beckmann[2], Bogdan Lazar[3], and Anton Setzer[4]

[1] Swansea University, Dept. of Computer Science
fahadalhabardi@gmail.com
[2] Swansea University, Dept. of Computer Science
a.beckmann@swansea.ac.uk
https://www.beckmann.pro/
[3] University of Bath
lazarbogdan90@yahoo.com
[4] Swansea University, Dept. of Computer Science
a.g.setzer@swansea.ac.uk
http://www.cs.swan.ac.uk/~csetzer/

Abstract

We have shown in [1] how some of the standard Bitcoin scripts can be verified using Hoare Logic in the interactive theorem prover Agda. Our framework was developed for standard instructions and in addition the multisig and the time delay instructions. We developed two ways of establishing human-readable weakest preconditions: (1) A step-by-step approach of working backwards in the program and (2) symbolic execution of the program and determining the accepting paths.

In this presentation, we investigate how these two approaches can be extended to Bitcoin scripts that use non-local instructions such as `OP_IF`, `OP_ELSE`, and `OP_ENDIF`. Our approach is based on a basic operational semantics [12], which added an additional stack called lfStack to the standard stack.

One of the most notable trustless and decentralised architectures is given by blockchain technology. It runs in a trustless environment using immutable peer-to-peer technology combined with a consensus protocol. Cryptocurrencies, which may be combined with smart contracts, are based on blockchain technology. Blockchain technology enables data to be shared universally in a secure, trusted, and synchronised way [15].

Satoshi Nakamoto [11] launched Bitcoin in 2008 as the first cryptocurrency. Bitcoin enables private, anonymous payments over a peer-to-peer network [9]. Since then, several cryptocurrencies have been introduced such as Ethereum [6]. This resulted in a new and emerging era of decentralised and digital monetary systems in which there is no need for any central entity such as central banks to manage and control users' transactions.

Smart contracts are another intriguing topic that has emerged as a result of the new era of blockchain [13, 10]. Smart contracts are defined as programs that are performed automatically when specific circumstances are met. Because smart contracts are capable of locking high monetary values, they are business-critical systems, hence techniques for evaluating their security and validity are required [10, 14]. In Bitcoin, smart contracts are written in the language Script. [5]. Other cryptocurrencies have their own languages [2]. One should note that the Ethereum Virtual Machine (EVM), into which smart contracts of Ethereum's smart contract languages are compiled, is as Bitcoin Script based on a stack machine[1], so we expect that techniques used for verifying Bitcoin scripts carry over when verifying smart contracts of the EVM.[2]

[1] [3, Ch 13]: "The EVM has a stack-based architecture, storing all in-memory values on a stack."

[2] There are as well many differences: The EVM is Turing complete and has jumps, has access to the state, and can make calls to other contracts. See the following blogposts/forum discussions comparing the two machines: [8, 7]

Hoare triples have been used to validate the correctness and capability of smart contract programs. In our approach we will verify these Hoare triples using Agda as our proof assistant for two reasons: (1) Agda is both a programming language and a theorem prover, which means we can both execute and verify smart contracts in Agda. (2) Proofs in Agda can be checked by hand. Smart contracts must be carefully verified because of high monetary values associated with them. It might not be sufficient to provide trust in the correctness of a smart contract to have a proof in a theorem prover, having to rely on its correct implementation – attackers might use an error in the theorem prover in order to create a false proof of the correctness of a compromised smart contract. If the amount of money being protected by a Bitcoin script is high, in addition to machine checking manual checking of correctness proofs might be appropriate.

In our previous article [1] we defined and formalised an operational semantics of a fragment of Bitcoin Script using instructions having only local behaviour. This included basic instructions and some more complex ones: the multi-signature instruction and an instruction enforcing a time delay. Then we verified those scripts using weakest preconditions for Hoare triples. The operational semantics was based on a state $\mathsf{StackState} := \mathsf{Time} \times \mathsf{Msg} \times \mathsf{Stack}$ (more precisely in Agda an explicit record type was used) formed from the normal stack Stack, a message Msg representing the message of the transaction to be signed, and the current time Time in Agda.

In this talk we will investigate the addition of conditional instructions to our fragment of Bitcoin Script, which have a non-local behaviour, and will discuss how to verify scripts written in this extended language. As discussed in [12], the use of control follow statements can be dealt with in the operational semantics by using a second stack IfStack, which keeps track of the nesting of conditionals. The new state is $\mathsf{Time} \times \mathsf{Msg} \times \mathsf{Stack} \times \mathsf{IfStack}$ defined as a record type State in Agda. Our approach is different from the usual approach of converting programs in Forth involving conditionals into programs with jumps. Instead, we work directly with the unstructured machine programs, and we adopted the usual techniques in Hoare logic for dealing with conditionals to such unparsed programs.

Conditionals can result in an exponential increase of the number of paths in a program. This is a well-known problem in Bitcoin: For instance, Antonopoulos [4, Ch. 7] writes: "Bitcoin Script flow control can be used to construct very complex scripts with hundreds or even thousands of possible execution paths. There is no limit to nesting, but consensus rules impose a limit on the maximum size, in bytes, of a script".

In [1] we argued that the correctness of Bitcoin scripts can be considered as verification of the security of access control systems. Access control is used to restrict access to a resource which in our case is access to the cryptocurrency in question, i.e. bitcoins. Hoare logic, which is based on preconditions and postconditions, is particularly well suited for safety-critical systems with a controllable set of inputs. Additionally, it enables interaction between multiple procedures within a program. When dealing with access control, weakest preconditions are more suitable: The precondition must not only be sufficient but as well necessary to ensure that the post-condition (which expresses access to the resource) is fulfilled after execution of the program.

As in [1] we use two methods for obtaining a human-readable weakest precondition that can be used to assist developers of smart contracts in Bitcoin. The first method is a step-by-step approach that works backwards through the program instruction by instruction. The second method is symbolic execution of the code and conversion to a nested case distinction, which enables reading off the weakest preconditions as the disjunction of the accepting paths. These methods have been formalised in Agda and as an example we apply them to the verification of P2PKH and P2MS, and combinations of them with conditionals. Since we obtain human readable weakest preconditions, these methods allow to reduce the validation gap between user requirements and formal specifications.

References

[1] Fahad F. Alhabardi, Arnold Beckmann, Bogdan Lazar, and Anton Setzer. Verification of Bitcoin Script in Agda using weakest preconditions for access control, 2022. URL: https://arxiv.org/abs/2203.03054, arXiv:arXiv:2203.03054, doi:10.48550/ARXIV.2203.03054.

[2] Mouhamad Almakhour, Layth Sliman, Abed Ellatif Samhat, and Abdelhamid Mellouk. Verification of smart contracts: A survey. *Pervasive and Mobile Computing*, 67:1–19, 2020. doi:10.1016/j.pmcj.2020.101227.

[3] Andreas Antonopoulos and Gavin Wood. *Mastering Ethereum. Building smart contracts and Dapps*. O'Reilly Media, November 2018.

[4] Andreas M Antonopoulos. *Mastering Bitcoin: Programming the open blockchain*. O'Reilly, 2nd edition, 2017.

[5] Harris Brakmić. Bitcoin Script. In *Bitcoin and Lightning Network on Raspberry Pi: Running Nodes on Pi3, Pi4 and Pi Zero*, pages 201–224, Berkeley, CA, 2019. Apress. doi:10.1007/978-1-4842-5522-3_7.

[6] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014. White paper. URL: https://ethereum.org/en/whitepaper.

[7] Differences/similarities of "Bitcoin script" and "Ethereum smart contract"?, Retrieved 9 March 2022. Forum discussion. URL: https://tinyurl.com/2p9xr9cc.

[8] From bitcoin script engine to Ethereum virtual machine, Retrieved 9 March 2022. Blogpost. URL: https://tinyurl.com/28yubhm3.

[9] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. Decentralization in Bitcoin and Ethereum Networks. In *Financial Cryptography and Data Security*, pages 439–457, Berlin, Heidelberg, 2018. Springer Berlin Heidelberg. doi:10.1007/978-3-662-58387-6_24.

[10] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 254–269, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2976749.2978309.

[11] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 2008. URL: https://www.debr.io/article/21260-bitcoin-a-peer-to-peer-electronic-cash-system.

[12] Anton Setzer and Bogdan Lazar. Modelling smart contracts of Bitcoin in Agda, June 2021. In Henning Basold (Ed): TYPES 2021 – Book of Abstracts, 27th International Conference on Types for Proofs and Programs on 14 - 18 June 2021. URL: https://types21.liacs.nl/download/modelling-smart-contracts-of-bitcoin-in-agda/.

[13] Nick Szabo. Smart contracts: Building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought*, 18(2), 1996. URL: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.

[14] Maximilian Wohrer and Uwe Zdun. Smart Contracts: Security patterns in the Ethereum ecosystem and Solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 2–8, 2018. doi:10.1109/IWBOSE.2018.8327565.

[15] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 557–564, 2017. doi:10.1109/BigDataCongress.2017.85.