Provably Correct Smart Contracts: An Approach using DeepSEA

Daniel Britten
db130@students.waikato.ac.nz
The University of Waikato
Hamilton, New Zealand

Vilhelm Sjöberg
vilhelm.sjoberg@certik.io
CertiK
USA

Steve Reeves
stever@waikato.ac.nz
The University of Waikato
Hamilton, New Zealand

Abstract

It is possible to download a piece of software over the internet and then verify its correctness locally using an appropriate trusted proof system. However, on a blockchain like Ethereum, smart contracts cannot be altered once deployed. This guarantee of immutability makes it possible for end users to interact collectively with a 'networked' piece of software, with the same opportunity to verify its correctness.

Formal verification of smart contracts on a blockchain therefore offers an unprecedented opportunity for end users to collectively interact with a deployed instance of software that they can verify while not relying on a central authority. All that is required to be trusted beyond the blockchain itself is an appropriate proof system, a component which always needs to be in the trusted computing base, and whose rules and definitions can be public knowledge. DeepSEA (Deep Simulation of Executable Abstractions) could serve as such a proof system.

CCS Concepts: • **Security and privacy** → **Logic and verification**; • **Computer systems organization** → *Distributed architectures.*

Keywords: smart contracts, formal methods, blockchain.

ACM Reference Format:
Daniel Britten, Vilhelm Sjöberg, and Steve Reeves. 2022. Provably Correct Smart Contracts: An Approach using DeepSEA. In *Companion Proceedings of the 2022 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '22), December 5–10, 2022, Auckland, New Zealand*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3563768.3564116

SPLASH Companion '22, December 5–10, 2022, Auckland, New Zealand

ACM ISBN 978-1-4503-9901-2/22/12…$15.00
https://doi.org/10.1145/3563768.3564116

1 Introduction

Theorem-proving is used for safety-critical software such as [5] and as discussed in [2]. However, by pairing theorem-proving with a blockchain, the need to trust a central authority can be removed and replaced by trust in an appropriate proof system and a certain level of expertise in the end users to analyse high-level specifications.

A critical advantage of DeepSEA over other systems is that in DeepSEA the specifications are written at a high-level, making understanding them within reach of a wider range of end users, and closing the usual problematic gap between problem statement and computational expression of the solution. Crucially, the high-level specifications will be satisfied by the executable bytecode due to stepwise refinement.

Here is an example of a specification written in DeepSEA for a crowdfunding contract. The intended definition of '*balance_backed*' here is that the funds the smart contract holds are greater than or equal to the sum of the donations recorded in the '*backers*' mapping, if the crowdfunding campaign has not yet been funded. *CF* abbreviates Crowdfunding, *bc_s* abbreviates blockchain state, *c_addr* abbreviates contract address, and *to_uint* corresponds to converting to *uint*.

```
Definition Safe P := ∀ bc_s, Reachable bc_s → P bc_s.
Definition balance_backed bc_s :=
  let s := (contract_state bc_s) in (CF_funded s) = false
    → sum (CF_backers s) ≤ to_uint (balance bc_s (c_addr))
      ∧ (∀ k v, get k (CF_backers s) = Some v → v ≥ 0).
Lemma sufficient_funds_safe := Safe balance_backed.
```

A reader of the above specification with sufficient expertise in logic can attest to the fact that it captures the notion of the '*balance_backed*' safety property, assuming the definitions used are reasonable. A diligent reader would want to be certain that the smart contract deployed on the blockchain is a refinement of the high-level version of the smart contract that this specification refers to (via the definition of *Reachable*). This is where DeepSEA is useful as it can bridge the gap between the high-level specifications and the executable bytecode, making use of Hirai's Lem model of the Ethereum Virtual Machine (EVM) [4]. The DeepSEA compiler is partly based upon the CompCert verified compiler [6].

There is ongoing work on DeepSEA, but we now focus on three aspects of DeepSEA which have been a focus lately.

2 Checks-Effects-Interactions Pattern

In previous work [1] we implemented a solution to the problem of reentrancy by following the standard approach of the Checks-Effects-Interactions pattern (CEIP). Reentrancy is when a smart contract A causes the execution of another contract B which calls a method in the original contract A before the original execution of A has finished. By structuring programs so that interactions with other contracts happen *after* any checks and side effects, Ethereum programmers can ensure that any reentrancy is benign. We define in Coq what it means for a contract to follow the CEIP and develop an Coq tactic that automatically proves that a given contract follows it (or signals an error if not). Insisting on the CEIP allows us to have the reassurance that the correctness proofs are not invalidated by the possibility of reentrancy. We can be confident of DeepSEA proofs without the complexity that fully modelling reentrancy would require. This simplifies the proofs without compromising on correctness.

3 Towards Verified Price Oracles

In previous work [3], Dave, Sjöberg, and Sun presented a verified DeepSEA Automated Market Maker (AMM) contract along with a theorem establishing that there is a lower bound on the cost of manipulation of the token price by any attacker. AMMs facilitate trading between tokens and can also serve as oracles for the prices of these tokens. This lower bound helps guarantee that it will not be worthwhile for attackers to influence the cost of the token in order to try make a profit by making related trades involving some contract that relies on the AMM as an oracle for token prices. The work of Dave et al. is a step towards a decentralised finance landscape which is provably safe from a range of potential exploits.

4 Modelling a Blockchain

In recent work[1], we implemented a model of a blockchain suitable for facilitating the specification of correctness properties at a high-level. This involved taking a snapshot approach and a successful-calls approach.

4.1 Snapshot Approach

Modelling smart contracts is complex because the blockchain is continuously being updated and it would be incorrect to assume that the smart contract had existed since the genesis of the blockchain. This implies the need for a "snapshot approach" that begins with an arbitrary state of the blockchain and models the execution from that point onwards. This approach is implemented using a mechanism in Coq that allows a *section* of code to refer to arbitrary variables.

The idea is that the model begins from a snapshot where every variable relevant to the snapshot is arbitrary - though potentially subject to constraints such as every balance being positive. Using this mechanism in Coq is equivalent to using universal introduction in logic. The snapshot approach is used in defining *Reachable* in the crowdfunding correctness proof. This ensures that the proof applies to the real-world Ethereum blockchain because we have proven the contract correct with respect to an arbitrary blockchain state.

[1] https://github.com/Coda-Coda/Crowdfunding/tree/splash-2022-poster.

4.2 The Successful-Calls Approach

When interacting with a smart contract on a blockchain, there is a range of situations where the smart contract will 'revert', i.e. cancel, the current execution and return to a previous state with no change other than the appropriate transaction fee (gas) being charged to the end user.

Given a snapshot, we model all possible further actions. That is, we consider the effect of all possible calls to the smart contract and then model the outcome of those calls. This is adequate; however, we can take a more elegant approach with Coq: we require a proof that the call will not result in a revert. The calls that do result in reverts are then handled as one exceptional case. This is the novel approach taken in this model. This results in proofs focusing on the cases where the function succeeds, avoiding repeatedly proving the trivial revert case, resulting in more elegant proofs.

5 Conclusion

DeepSEA has the potential to serve as a trusted proof system for developing smart contracts on the Ethereum blockchain. Work towards overcoming the obstacles of handling reentrancy and modelling the blockchain elegantly is discussed and shows that DeepSEA holds promise.

References

[1] Daniel Britten, Vilhelm Sjöberg, and Steve Reeves. 2021. Using Coq to Enforce the Checks-Effects-Interactions Pattern in DeepSEA Smart Contracts (Short Paper). In *3rd International Workshop on Formal Methods for Blockchains (FMBC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. https://doi.org/10.4230/OASIcs.FMBC.2021.3

[2] Stephen Chong, Joshua Guttman, Anupam Datta, Andrew Myers, Benjamin Pierce, Patrick Schaumont, Tim Sherwood, and Nickolai Zeldovich. 2016. Report on the NSF Workshop on Formal Methods for Security. https://dl.acm.org/citation.cfm?id=3040225.

[3] Kinnari Dave, Vilhelm Sjöberg, and Xinyuan Sun. 2021. Towards Verified Price Oracles for Decentralized Exchange Protocols. In *3rd International Workshop on Formal Methods for Blockchains (FMBC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. https://doi.org/10.4230/OASIcs.FMBC.2021.1

[4] Yoichi Hirai. 2017. Defining the Ethereum Virtual Machine for interactive theorem provers. In *International Conference on Financial Cryptography and Data Security*. Springer, 520–535. https://doi.org/10.1007/978-3-319-70278-0_33

[5] Sapna Jaidka, Steve Reeves, and Judy Bowen. 2017. Modelling safety-critical devices: coloured Petri nets and Z. *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (2017). https://doi.org/10.1145/3102113.3102125

[6] Xavier Leroy, Sandrine Blazy, Daniel Kästner, Bernhard Schommer, Markus Pister, and Christian Ferdinand. 2016. CompCert – A Formally Verified Optimizing Compiler. In *ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress*. SEE, Toulouse, France. https://hal.inria.fr/hal-01238879