Formalization of Blockchain Oracles in Coq

Mohammad Shaheer[1], Giselle Reis[1], Bruno Woltzenlogel Paleo[2], and Joachim Zahnentferner[2]*

[1] Carnegie Mellon University, Qatar – mshaheer@andrew.cmu.edu, giselle@cmu.edu
[2] Djed Alliance – bruno.wp@gmail.com

Abstract

Oracles are crucial components that bring external data to smart contracts deployed on blockchains. With the recent surge in popularity of decentralized finance (DeFi) applications, it is critical to provide assurances about the oracle implementations as these applications deal with high-value transactions and a small price discrepancy can lead to huge losses. Although there are many oracle implementations, there have not been many efforts to formally verify their behavior. We present a simple oracle implementation in Solidity and its formal model using the Coq interactive theorem prover. We also prove interesting trace-level properties that give us formal guarantees about the oracle's behavior at a high level. Our work can be a stepping stone for future oracle implementations and provide developers with a framework for formally verifying their implementations.

Smart contracts are programs that run on blockchains, maintain an internal state and provide functions whose execution may depend on the internal state and on calls to functions of other smart contracts. Due to the decentralized operation of blockchains, applications built through smart contracts may enjoy desirable qualities such as transparency, censorship resistance and interoperability. The flexible programmability of smart contracts coupled with the qualities of the blockchain environment formed a fertile ground for financial applications, where such qualities are highly desirable and at times absent in the traditional financial sector [1]. This ushered an era of so-called *Decentralized Finance* (DeFi). Thousands of digital assets have been implemented as ERC20 [2] smart contracts that maintain, as their internal states, the balances of all users of an asset and provide functions for transferring amounts from one user to another. Collateralized lending applications [3] have been developed to allow users to lend and borrow such assets. Exchange applications [4] have been developed to allow users to swap assets. And various stablecoin algorithms have been proposed to allow the price of a digital asset to track the price of another asset.

Blockchain applications often need access to information that is not readily available on the blockchain. For DeFi applications in particular, data from the external world can be crucial. For example: a stablecoin needs to know the relative price of the fiat currency to which it is pegged; a collateralized lending application needs to know the value of the collateral to know when to liquidate debts. This need is satiated through a special type of blockchain application known as *oracle*. An oracle has a smart contract that maintains the desired data in its internal state and implements an *oracle protocol* that establishes the conditions under which various entities may write or read data from the contract. Some of these entities are also responsible for operating off-chain components of the oracle to obtain the data to be written to the contract.

Oracle protocols differ widely in how frequently the data is updated, how data consumers are charged for reading the data, how data being written by multiple sources is aggregated, how misbehaviour is penalized and desirable behaviour incentivized.

Unfortunately, existing oracle implementations are ad hoc, lacking a formal definition, or even a precise description, of their protocol. Without a formal definition, it is harder to provide guarantees for the oracle's functioning. In the worst case, there might even be unidentified exploitable security vulnerabilities. This leads to uncertainty and compromises confidence on the applications that depend on such oracle implementations.

Our work tackles this issue by: (1) proposing a (non-exhaustive) set of desirable properties for oracle protocols; (2) formalizing a simple oracle protocol in Coq and formally proving that it satisfies these desirable properties; (3) implementing an oracle smart contract in Solidity closely following the formal oracle protocol.

Our work focuses on the long-term economic sustainability of the oracle. Since an oracle has off-chain components that are costly to operate (e.g. to obtain data from an external source and write it to the contract), the entities operating these components need to be economically incentivized to keep doing their work. This is done by charging fees from the data consumers who read data from the oracle. The oracle protocol automatically adjusts the fees for reading data based on the costs of the data provider and the frequency of data reads, aiming to ensure that: (A) the costs of the data provider are covered by the fee revenue from data reads; and (B) every data consumer is paying a price that is fair in the sense that they never have to pay twice for the same data point and the cost of data is distributed evenly between all data consumers.

Given the desired properties (A) and (B), we developed an oracle smart contract in Coq in parallel with its Solidity implementation assuming a single external data provider. The Coq formalization was designed such that properties could be proved fairly easily, but also that it faithfully represents the Solidity implementation. Striking this balance is not always straightforward. We went over a few different representations before establishing what follows.

Since Solidity is an object-oriented language, the formalization uses Coq's `Record`s to represent objects. The oracle contract itself is an object, which is implemented in Coq as a `Record` called `State`. This record consists of two sub-records: `OracleState` and `OracleParameters`; and a `Trace` list. `OracleState` contains contract attributes that change with time and `OracleParameters` encompasses immutable attributes set at initialization. `Trace` is a list of `Events` that keeps a record of the operations performed on the contract. Every time a contract function is called, its corresponding `Event` is added to the `Trace`. Finally, in order to account for side effects, contract functions implemented in Coq have explicit `States` in their input and output.

Our oracle protocol uses a subscription-based model where consumers deposit credit before reading the data. The cost of a data read is taken from a consumer's balance and it depends, among other things, on a *base fee*, which can be adjusted but may not exceed a *maximum fee*. These fees are part of the contract's parameters and can be adjusted by the data provider. Proper adjustments can, under certain assumptions, provably ensure that properties (A) and (B) are fulfilled.

Using the oracle formalization in Coq, we could prove two main theorems:

Thm 1. *For all consumers c, if* $\mathsf{credit}(c) \geq 0$*, then after any contract function call* $\mathsf{credit}(c) \geq 0$.

Thm 2. *Between two consecutive data writes, each consumer pays once to read the data.*

Both properties are proved using induction on the `Trace`. The proof of Theorem 1 uses two helper lemmas. The proof of Theorem 2 uses nine helper lemmas. Both proofs also use a number of auxiliary definitions for manipulating states and traces.

The implementation and formalization can be found at, respectively, `https://github.com/DjedAlliance/Oracle-Solidity/tree/cmu-qatar` and `https://github.com/DjedAlliance/Oracle-FormalMethods`.

For future work, our next step is to shift our focus from economic aspects to governance aspects around the whitelist of data providers, who can adjust the oracle's parameters and vote to add or remove data providers from the whitelist. We plan to prove theorems related to the security of such governance processes under circumstances where some data providers may have been compromised.

References

[1] Bowen Liu, Pawel Szalachowski, and Jianying Zhou. A First Look into DeFi Oracles, 2020.

[2] ERC20 White Paper. https://erc20.tech/erc20token-whitepaper, Accessed on Mar. 2023.

[3] Robert Leshner and Geoffrey Hayes. Compound: The money market protocol. https://eauapcddxck6wa2w2ufzqnjvg7u7bgw22fh6fkdkwooyeh7twypq.arweave.net/ICgHiGO4lesDVtULmDU1N-nwmtrRT-KoarOdgh_zth8/documents/Compound.Whitepaper.pdf, Accessed on Mar. 2023.

[4] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. https://berkeley-defi.github.io/assets/material/Uniswap%20v3%20Core.pdf, Accessed on Mar. 2023.

Notes

*. "Joachim Zahnentferner" is the pseudonym used by Bruno Woltzenlogel Paleo for papers on the topic of blockchains and cryptocurrencies. He continues to use his own name for papers on topics related to logic and formal methods.