Formal Investigation of the Extended UTxO Model (Extended Abstract)

Orestis Melkonian
Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
melkon.or@gmail.com

Wouter Swierstra
Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
w.s.swierstra@uu.nl

Manuel Chakravarty
Input Output HK
Hong Kong
manuel.chakravarty@iohk.io

1 Introduction

Distributed ledger technology has seen a myriad of applications during the past few years [3, 4, 6], but has also unveiled a new source of vulnerabilities[1] arising from the distributed execution of *smart contracts* (programs that run on the blockchain). Since most of these applications deal with transactions of significant funds, it is crucial that we can formally reason about their concurrent behaviour.

To this end, we attempt to lay the foundations for a mechanized formal framework, where one can verify that such scenarios are impossible. The grand vision is that blockchain developers will work alongside a proof assistant, writing code that carries formal proofs of the desired properties. We formulate an accounting model for ledgers based on *unspent transaction outputs* (UTxO) in Agda [8], exploiting its expressive dependent type system to mechanically enforce desired properties statically. An executable specification of our formal development is available on Github[2].

2 Formal Model

Our formalization closely follows the abstract accounting model for UTxO-based cryptocurrencies presented in [10], which leaves out details of other technical components of the blockchain such as cryptographic operations. We further extend the original formulation to cover the extensions employed by the Cardano blockchain platform [1]. Cardano extends Bitcoin's UTxO model [7] with data scripts on transaction outputs, bringing it on par with Ethereum's expressive account-based scripting model [5], as well as support for multiple cryptocurrencies on the same ledger [2].

Transactions & Ledgers For simplicity, we model monetary quantities and hashes as natural numbers. We treat the type of addresses as an abstract module parameter equipped with an injective hash function. Transactions consist of a list of outputs, transferring a monetary value to an address, and a list of inputs referring to previous outputs:

```
module UTxO (Address : Set) (_♯ : Address → ℕ) where
record OutputRef : Set where
  field id    : Address
        index : ℕ
record Input {R D : Set} : Set where
  field outRef    : OutputRef
        redeemer  : State → R
        validator : State → R → D → Bool
record Output {D : Set} : Set where
  field value   : Value
        address : Address
        data    : State → D
record Tx : Set where
  field inputs  : List Input
        outputs : List Output
        forge   : Value
        fee     : Value
```

Both inputs and outputs carry authorization scripts; for a transaction to consume an unspent output, the result of the validator script has to evaluate to *true*, given the current state of the ledger and additional information provided by the redeemer and data scripts:

```
authorize :: Input → List Tx → Bool
authorize i l = let s = getState l in
  validator i s (redeemer i s) (data (lookup l (outRef i)) s)
```

A ledger consists of a list of transactions, whose *unspent transaction outputs* we can recursively compute:

```
utxo : List Tx → List OutputRef
utxo []       = ∅
utxo (tx :: l) = (utxo l \ outRefs tx) ∪ outputs tx
```

Validity We are now ready to encode the validity of a transaction with respect to a given ledger as a dependent data type. For the sake of brevity, we present only two such conditions, namely that inputs refer to existing unspent outputs and all authorizations succeed:

[1] https://en.wikipedia.org/wiki/The_DAO_(organization)
[2] https://github.com/omelkonian/formal-utxo/

PL'18, January 01–03, 2018, New York, NY, USA
2018.

```
record IsValidTx (tx : Tx) (l : List Tx) : Set where
  field validOutputRefs :
          ∀i → i ∈ inputs tx → outRef i ∈ utxo l
        allInputsValidate :
          ∀i → i ∈ inputs tx → authorize i l ≡ true
        ...
```

Other validity conditions include that no output is spent twice ($\mathit{Unique}\ \ (\mathit{outRef}\ <\$>\ \mathit{inputs}\ \ \mathit{tx})$) and transactions preserve total values ($\mathit{forge} + \sum_{\mathit{in}} \equiv \mathit{fee} + \sum_{\mathit{out}}$).

It is now possible to characterize a well-formed *Ledger*, by requiring a validity proof along with each insertion to the list of transactions. Exposing only this type-safe interface to the user will ensure one can only construct valid ledgers.

Multi-currency In order to allow for user-issues currencies, we generalize the transferred values to maps from currencies to natural numbers. Fortunately, the validity conditions work for any (ordered) commutative group, so we can effortlessly replace the previous numeric representation with currency maps. *Monetary policies* are enforced by requiring a forging transaction to spend an output that carries the corresponding policy in its validator script:

```
forging : ∀c → c ∈ keys (forge tx) →
            ∃[ i ] ∃ λ (_ : i ∈ inputs tx) →
              (address $ lookupOutput l (outRef i)) ♯ ≡ c
```

3 Meta-theory

Apart from being able to define correct-by-construction ledgers, we can prove further meta-theoretical results over our existing formulation.

Weakening Given a suitable injection on addresses, we prove a weakening lemma, stating that a valid ledger parametrized over some addresses will remain valid even if more addresses become available:

$$\mathit{weakening}\ :\ \frac{(f : A \hookrightarrow B) \to \mathit{Ledger}\ l}{\to \mathit{Ledger}\ (\mathit{weaken}\ f\ l)}$$

Weakening consists of traversing the ledger's outputs and transporting all addresses via the supplied injection; in order to keep references intact, the injection has to also preserve the original hashes[3].

Combining Ideally, one would wish for a modular proof procedure, where the prover "zooms in" of interest and then weakens to the

We provide a ledger combinator that interleaves two *separate* ledgers. Due to lack of space, we eschew from giving the formal definition of the separation connective $\_*\_ \approx \_$. Briefly, two ledgers are separate if they do not share any common transaction and the produced interleaving does not break previous validator scripts (since they will now execute on a different ledger state). These conditions are necessary to transfer the validity of the two sub-ledgers to a proof of validity of the merged ledger.

$$\_\leftrightarrow\_\dashv\_\ :\ \frac{\mathit{Ledger}\ l \to \mathit{Ledger}\ l' \ \to l * l' \ \approx l''}{\to \mathit{Ledger}\ l''}$$

The notion of weakening we previously defined proves rather useful here, as it allows merging two ledgers acting on different addresses.

4 Future Work

Proof Automation Although we have made it possible to express desired ledger properties in the type system, users still need to manually discharge tedious proof obligations. In order to make the proof process more ergonomic, we can prove that the involved propositions are *decidable*, thus defining a decision procedure for closed formulas that do not contain any free variables [9]; we have already proven decidability of the validity conditions[4] and wish to also cover the propositions appearing in weakening and combining.

Furthermore, it would be convenient to utilize Agda's meta-programming facilities to automatically insert weakening transports as needed when combining ledgers of a different address space. This would allow the prover to focus only on the transactions of interest and afterwards combine the results, without having to worry about intermediate weakening steps that are necessary to plumb things together.

Comparison with Ethereum It would be interesting to conduct a more formal comparison between UTxO-based and account-based ledgers, relying on previous work on *chimeric ledgers* [11] that gives a translation between these two approaches. Note that implementing this translation on our inherently-typed representation would guarantee that we only ever compile down to **valid** UTxO ledgers.

Acknowledgments

We would like to thank Philip Wadler and Michael Peyton Jones for helpful discussion and IOHK for financial support.

References

[1] 2019. The Extended UTxO Model. Retrieved 5/2019 from https://github.com/input-output-hk/plutus/blob/master/docs/extended-utxo/README.md

[3] A practical case of such weakening is migrating from a 32-bit word address space to a 64-bit one.

[4] There is an example construction of a valid ledger in the code repository, where our decision procedure automatically discharges all required proofs.

[2] 2019. Multi-Currency. Retrieved 5/2019 from https://github.com/input-output-hk/plutus/blob/master/docs/multi-currency/multi-currency.md

[3] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2014. Secure multiparty computations on bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 443–458.

[4] Iddo Bentov and Ranjit Kumaresan. 2014. How to use bitcoin to design fair protocols. In *International Cryptology Conference*. Springer, 421–439.

[5] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* (2014).

[6] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1991. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM (JACM)* 38, 3 (1991), 690–728.

[7] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[8] Ulf Norell. 2008. Dependently typed programming in Agda. In *International School on Advanced Functional Programming*. Springer, 230–266.

[9] Paul Van Der Walt and Wouter Swierstra. 2012. Engineering proof by reflection in Agda. In *Symposium on Implementation and Application of Functional Languages*. Springer, 157–173.

[10] Joachim Zahnentferner. 2018. An Abstract Model of UTxO-based Cryptocurrencies with Scripts. *IACR Cryptology ePrint Archive* 2018 (2018), 469.

[11] Joachim Zahnentferner. 2018. Chimeric Ledgers: Translating and Unifying UTXO-based and Account-based Cryptocurrencies. *IACR Cryptology ePrint Archive* 2018 (2018), 262.