Program logics for ledgers

Orestis Melkonian[1,2]*, Wouter Swierstra[3], and James Chapman[2]

[1] University of Edinburgh, Scotland
[2] Input Output, Global
[3] Utrecht University, The Netherlands

Introduction Distributed ledgers nowadays manage substantial monetary funds in the form of cryptocurrencies such as Bitcoin, Ethereum, and Cardano. For such ledgers to be safe, operations that add new entries must be cryptographically sound—but it is less clear how to reason effectively about such ever-growing linear data structures.

We view distributed ledgers as *computer programs*, that, when executed, transfer funds between various parties. As a result, familiar program logics, such as Hoare logic and separation logic, can be defined in this novel setting. Borrowing ideas from concurrent separation logic, this enables modular reasoning principles over arbitrary fragments of any ledger. Our results have been mechanised in the Agda proof assistant [3] and are publicly available:

https://omelkonian.github.io/hoare-ledgers

A simple linear ledger We start by studying the simplest form of ledger; assuming an abstract set of participants $\mathcal{P}$, programs are linear sequences of transactions and states S keeps track of everyone's account balance in a finite map.

$$\begin{aligned} T &:= \mathcal{P} \xrightarrow{n} \mathcal{P} \\ L &:= \epsilon \mid T; L \\ S &:= \mathcal{P} \mapsto \mathbb{Z} \end{aligned}$$

```
Alice pays Bob 5;
Alice pays Carroll 10;
Dana pays Alice 2;
...
```

It's straightforward to define denotational, operational, and axiomatic semantics, as well as prove them equivalent to one another. Transactions and ledgers take denotations in the same domain, namely state transition functions $d(t) : S \to S$. For the sake of brevity, we refer to the Agda development for the full definitions and only present the essential Hoare rules:

$$\frac{}{\{P\}\ \epsilon\ \{P\}}\ \text{STOP} \qquad \frac{\{P\}\ l\ \{Q\}}{\{P \circ d(t)\}\ t; l\ \{Q\}}\ \text{STEP}$$

$$\frac{\{P\}\ l_1\ \{Q\} \qquad \{Q\}\ l_2\ \{R\}}{\{P\}\ l_1 +\!\!+ l_2\ \{R\}}\ \text{APP} \qquad \frac{}{\{p_1 \mapsto n\}\ p_1 \xrightarrow{n} p_2\ \{p_2 \mapsto n\}}\ \text{SEND}$$

These should remind you of the corresponding rules for assignment and sequencing in imperative programs from traditional Hoare logic. It is natural to form chains of these Hoare-style state predicates like so: $\{\lambda\sigma.\ \sigma(A) = 2\}\ A \xrightarrow{1} B\ \{\lambda\sigma.\ \sigma(A) = 1\}\ A \xrightarrow{1} C\ \{\lambda\sigma.\ \sigma(A) = 0\}$. However, each step operates on the whole state which is not modular and would not scale to ever-growing blockchain ledgers.

Towards separation We can remedy this by exploiting the monoidal structure of the state space (i.e. pointwise addition of maps $\oplus$), leading to the following notion of *separating conjunction* [4] and the usual Hoare rules of (concurrent) separation logic:

$$(P * Q)(\sigma) := \exists\sigma_1.\ \exists\sigma_2.\ P(\sigma_1)\ \wedge\ Q(\sigma_2)\ \wedge\ \sigma = \sigma_1 \oplus \sigma_2$$

$$\frac{\{P\}\ l\ \{Q\}}{\{P * R\}\ l\ \{Q * R\}}\ \text{FRAME} \qquad \frac{\{P_1\}\ l_1\ \{Q_1\} \qquad \{P_2\}\ l_2\ \{Q_2\}}{\{P_1 * P_2\}\ l_1 \,||\, l_2\ \{Q_1 * Q_2\}}\ \text{PAR}$$

*Funded by Input Output (iohk.io) through the Edinburgh Blockchain Technology Lab.

Notice the lack of the usual freshness side-conditions, rendering our logic compositional, i.e. a proof about a large ledger can be obtained from proofs about its individual parts.

Extending to the UTxO case If we now try to extend our technique to previous formalisations of UTxO-based blockchain ledgers [1, 2], we are forced to introduce freshness side-conditions when composing *disjoint* ($\uplus$) sub-ledgers and end up with non-compositional rules:

$$\frac{\{P\}\ l\ \{Q\} \qquad l\ \#\ R}{\{P * R\}\ l\ \{Q * R\}}\ \textsc{frame} \qquad \frac{\{P_1\}\ l_1\ \{Q_1\} \qquad \{P_2\}\ l_2\ \{Q_2\} \qquad l_1\ \#\ P_2 \qquad l_2\ \#\ P_1}{\{P_1 * P_2\}\ l_1\ ||\ l_2\ \{Q_1 * Q_2\}}\ \textsc{par}$$

This is due to the fact that, in contrast to the previous *by-value* formulation, UTxO transactions reference previous unspent outputs *by name* (requiring the transaction's hash and an index into its outputs), so we need to explicitly track names which enforces a coarse level of modularity.

Abstract UTxO We propose a novel UTxO model that embraces the initial *value-centric* perspective, where previous unspent outputs are only referred to by value (i.e. the monetary value and the script that locks it). This means we now hold bags rather than maps in our state space, abstracting away from the explicit names and hash references of the concrete UTxO model. By exploiting the monoidal nature induced by pointwise bag addition/inclusion, we regain our compositional Hoare rules free of side-effects, as demonstrated in the example below that contrasts a monolithic proof using FRAME (left) to a modular proof combining two smaller proofs using PAR (right):

$$\begin{array}{l}
\{A \mapsto 1 * B \mapsto 0 * C \mapsto 0 * D \mapsto 1\} \\
\quad t_1 \quad \dashv \textsc{frame}(C \mapsto 0 * D \mapsto 1, \textsc{send}) \\
\{A \mapsto 0 * B \mapsto 1 * C \mapsto 0 * D \mapsto 1\} \quad \approx \\
\{C \mapsto 0 * D \mapsto 1 * A \mapsto 0 * B \mapsto 1\} \\
\quad t_2 \quad \dashv \textsc{frame}(A \mapsto 0 * B \mapsto 1, \textsc{send}) \\
\{C \mapsto 1 * D \mapsto 0 * A \mapsto 0 * B \mapsto 1\} \quad \approx \\
\{A \mapsto 0 * B \mapsto 1 * C \mapsto 1 * D \mapsto 0\} \\
\quad t_3 \quad \dashv \textsc{frame}(C \mapsto 1 * D \mapsto 0, \textsc{send}) \\
\{A \mapsto 1 * B \mapsto 0 * C \mapsto 1 * D \mapsto 0\} \quad \approx \\
\{C \mapsto 1 * D \mapsto 0 * A \mapsto 1 * B \mapsto 0\} \\
\quad t_4 \quad \dashv \textsc{frame}(A \mapsto 1 * B \mapsto 0, \textsc{send}) \\
\{C \mapsto 0 * D \mapsto 1 * A \mapsto 1 * B \mapsto 0\} \quad \approx \\
\{A \mapsto 1 * B \mapsto 0 * C \mapsto 0 * D \mapsto 1\} \quad \square
\end{array}$$

$$\begin{array}{c}
\{A \mapsto 1 * B \mapsto 0 * C \mapsto 0 * D \mapsto 1\} \\
\begin{array}{|l|l|}
\hline
\{A \mapsto 1 * B \mapsto 0\} & \{C \mapsto 0 * D \mapsto 1\} \\
\quad t_1 \quad \dashv \textsc{send} & \quad t_2 \quad \dashv \textsc{send} \\
\{A \mapsto 0 * B \mapsto 1\} & \{C \mapsto 1 * D \mapsto 0\} \\
\quad t_3 \quad \dashv \textsc{send} & \quad t_4 \quad \dashv \textsc{send} \\
\{A \mapsto 1 * B \mapsto 0\} \quad \square & \{C \mapsto 0 * D \mapsto 1\} \quad \square \\
\hline
\end{array} \dashv \textsc{par} \\
\{A \mapsto 1 * B \mapsto 0 * C \mapsto 0 * D \mapsto 1\} \quad \square
\end{array}$$

Sound abstraction Even though we believe our abstract UTxO model might be a better foundation for a next-generation blockchain, we still wish to guarantee that it is sound to reason at this higher level in order to prove properties of *concrete* UTxO ledgers that currently exist. To formulate soundness, we start by relating concrete ($\mathbb{C}$) and abstract ($\mathbb{A}$) states, i.e. collect all values of a key-value map in a bag: $abs^S(\sigma) = \{\sigma(k) | k \in \sigma$. A similar construction is defined for ledgers (abs^L). After proving a crucial lemma that connects the concrete and abstract denotational semantics (left), we can finally prove the *soundness theorem* (right):

$$\frac{\mathbb{C}[\![\ l\ ]\!](\sigma) = \tau}{\mathbb{A}[\![\ abs^L(l)\ ]\!](abs^S(\sigma)) = abs^S(\tau)} \qquad \frac{\mathbb{A}\{P\}\ abs^L(l)\ \{Q\} \qquad l\ \text{valid in}\ \sigma}{\mathbb{C}\{P \circ abs^S\}\ l\ \{Q \circ abs^S\}}\ \textsc{soundness}$$

Conclusion The use of a proof assistant was instrumental in navigating various points in the design space; we are now confident that our approach lays robust foundations at the ledger level and is able to culminate into larger-scale verification of actual smart-contracts.

References

[1] Nicola Atzei, Massimo Bartoletti, Stefano Lande, and Roberto Zunino. A formal model of Bitcoin transactions. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 - March 2, 2018, Revised Selected Papers*, volume 10957 of *Lecture Notes in Computer Science*, pages 541–560. Springer, 2018.

[2] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, and Philip Wadler. The Extended UTXO model. In Matthew Bernhard, Andrea Bracciali, L. Jean Camp, Shin'ichiro Matsuo, Alana Maurushat, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers*, volume 12063 of *Lecture Notes in Computer Science*, pages 525–539. Springer, 2020.

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

[4] John C. Reynolds. Separation Logic: A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 55–74. IEEE Computer Society, 2002.