INPUT | OUTPUT

Determinism of Ledger Updates

Authors : **Polina Vinogradova**, James Chapman, Andre Knispel,

and Orestis Melkonian

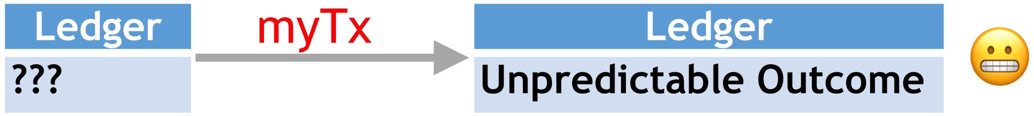
The Problem

- We want to be able to **predict what changes a transaction makes to the ledger** to which it is applied (or to its parts)
- This ability to predict is colloquially referred to as **ledger determinism**
- How can we **design ledgers** in a way that guarantees we can **make this prediction correctly**, taking the **onus off the users and contract designers** to ensure this?

The Problem

Why is this prediction hard?

- It should be **easy**, though - **blockchains rely on deterministic computation**!
- In practice, however, a user **cannot predict what ledger** their transaction/block will be applied to



- There are a number of **reasons** for this :
    - unpredictable propagation of transactions over the network, rollbacks, delays in getting the most current ledger, malicious actors, etc.

Motivating Example

Selling a Sword

Ledger
Offer1 : 1 Sword for 5 Coins paid to **PK1**
PK1 has 0 Coins
PK2 has 20 Coins
…

Transaction1 :

PK2 accepts **Offer1**

Ledger
PK1 has 5 Coins
PK2 has 15 Coins
PK2 has 1 Sword
…

Motivating Example

Selling a Sword

Ledger
Offer1 : 1 Swords for 5 Coins paid to **PK1**
PK1 has 0 Coins
PK2 has 20 Coins
…

Transaction2 :

Change Sword price in **Offer1** to 10 Coins

Ledger
Offer1 : 1 Swords for 10 Coins paid to **PK1**
PK1 has 0 Coins
PK2 has 20 Coins
…

Transaction1 :

PK2 accepts **Offer1**

Ledger
PK1 has 10 Coins
PK2 has 10 Coins
PK2 has 1 Sword
…

PK1 and **PK2** both care which transaction gets processed **first**!

Our approach

Make it formal, and forget the state (sort of)

- **Formally specify ledgers as state transition systems**, with valid transactions as the only transitions
- Ask “what if **the changes a transaction makes did not depend on the ledger state** to which it is being applied?”
- Formulate this in the language of our specification, and use mathematical tools to look for the answer

Ledger (Specification) *L*

- Specifies what it means for something to be a ledger
- Captures the structure shared by most ledgers
  - eg. Cardano, Bitcoin, Ethereum, Tezos, Zilliqa
- Is a tool for comparing different ledgers across a formally stated property, eg. determinism

$$\texttt{initState} \xrightarrow{\text{tx1}} \texttt{s1} \xrightarrow{\text{tx2}} \texttt{s2} \xrightarrow{\text{tx3}} \ldots$$

`State : Type`
ledger state type

`Tx : Type`
transaction type

`initState : State`
initial state

`updateState : Tx → State → State⊥`
output new state with a given transaction applied,
or throw an error

Valid States

From here on, we only talk about valid states

- A ledger state `s ∈ State` is valid whenever there exists a trace `[tx1; tx2; … txn] ∈ [Tx]` from the initial state to `s`, such that

```
(… (updateState (updateState initState tx1) tx2) … txn) = s
```

- We denote the set of **all valid states** (ie. the dependent pairs of a state and a trace proving its validity), plus the **error state**, by

```
ValSt⊥
```

- We use shorthand `(s l)` for updating `s` with the list of transactions `l`

Expressing Determinism

What do we want to state formally?

- Given two ledger states `s` and `s′`, and a transaction `tx`, the **changes** `tx` makes to both states are the same when `(s tx) ≠ ⊥ ≠ (s′ tx)`:

$$\Delta(s,\ (s\ tx)) = \Delta(s',\ (s'\ tx))$$

- But how do we define, and what can we say about

$$\Delta\ :\ (State,\ State) \to ???$$

Order-Determinism (OD)

Transaction commutativity with errors

$$\forall\ l \in [Tx],\ l' \in \text{Permutation}\ l,$$
$$(\text{initState}\ l) \neq \perp \neq (\text{initState}\ l')$$

$$\Rightarrow\ \text{initState}\ \overset{\text{lstx}}{\underset{\text{lstx}'}{\rightrightarrows}}\ s = (\text{initState}\ l) = (\text{initState}\ l')$$

This is a desired constraint in a system where users may have no control over the **order** in which their transactions will be applied

- Blockchains are such systems

Key Points

- Determinism is a property of a **ledger**, not specific transaction sets
- This model processes **all transactions in any order**, but the “bad” transactions produce a **special state** $\perp$, which we must correctly account for in all logical analysis
- **Swapping transactions pairwise is insufficient** to show ledger determinism — permutations of lists are needed
- In general, data structure-agnostic types of change sets are very hard — our ledger specification is uniquely suited for **reasoning about transactions as change sets** while remaining agnostic underlying types

Application to Cardano

Pointer addresses

- **Pointer addresses** are memory-saving pointers to full-length addresses, used in the Cardano platform specification (and implementation)
- Pointer addresses are **generated at the time of transaction application**
- We use our formal specification, and the order-determinism definition, to analyze the **address-generation procedure** and show that it **is not order-deterministic**
    - Example of pairwise swapping not affecting the ledger state being insufficient

Research Directions

We are working on :

- Adapting **data structure derivatives** from the Theory of Changes to formalize $\boldsymbol{\Delta}$
    - We need to **account for** $\perp$ in the adjusted definitions
    - **Spoiler** : for deterministic ledgers, we want to say that ledger change sets to correspond exactly to lists of transactions (ie. constant derivatives)

- Studying **ledger components** (**threads**)
    - **Conjecture** : Non-error output of a thread update should depend only on the data in that thread and the transaction updating it, and not the global state
    - Analyze Cardano's smart contracts as threads

Research Directions

We are working on :

- **Category-theoretic treatment** of ledgers
    - What can we say about ledgers using categorical tools?
    - Study the category where ledgers are objects, and morphisms preserve the initial state and update function (related to categories of deterministic automata)

- **Blocks** (not transactions) are the actual **atomic units of ledger updating**
    - How do we formalize the relation between blocks and transactions?