The Extended UTXO Model

Manuel M.T. Chakravarty, James Chapman, Kenneth MacKenzie,
Orestis Melkonian, Michael Peyton Jones, Philip Wadler

(presented by **Alexander Nemish**)

February 14, 2020

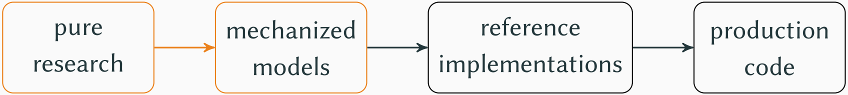
INPUT | OUTPUT

Introduction

Blockchain	Model	Turing-complete	Deterministic
Bitcoin	UTXO	×	×
Ethereum	Accounts	✓	×
Cardano (IOHK)	EUTXO	✓	✓

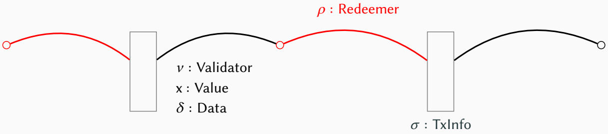
Methodology

- Focus on validating the relevant meta-theory
    - In contrast to validating individual contracts
- Fully mechanized approach, utilizing Agda's rich type system
- Fits well with IOHK's research-oriented approach



Contributions

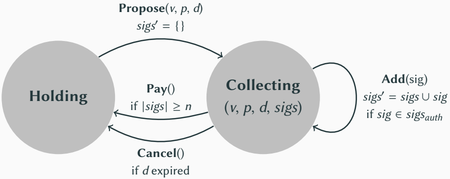
- **Detailed description of the Extended UTXO model (EUTXO)**



$$\nu(\rho,\ x,\ \delta,\ \sigma) \stackrel{?}{=} \text{True}$$

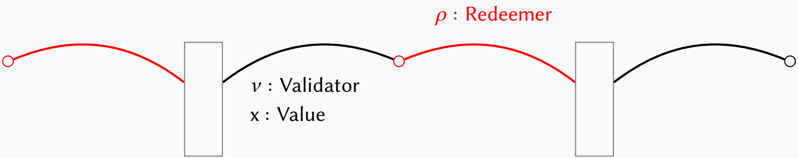
- **Formalization in** Agda

- **Proof of bisimulation with a specific form of state machines**

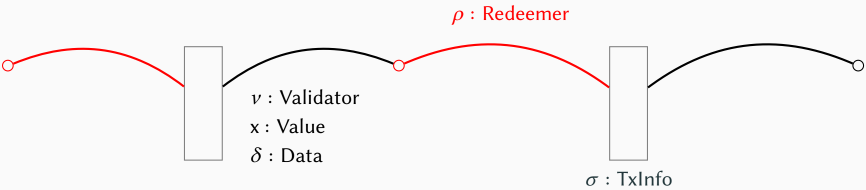


EUTXO, Informally…

UTXO vs EUTXO

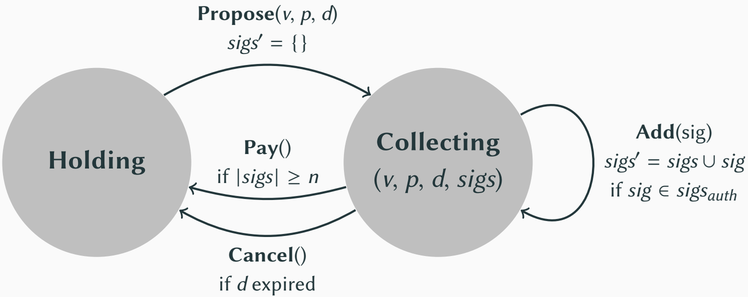


$$\nu(\rho) \stackrel{?}{=} \text{True}$$

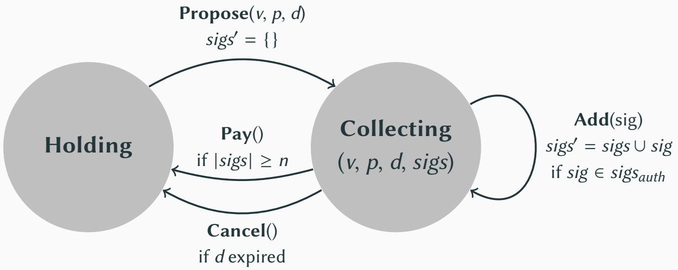


$$\nu(\rho,\ x,\ \delta,\ \sigma) \stackrel{?}{=} \text{True}$$

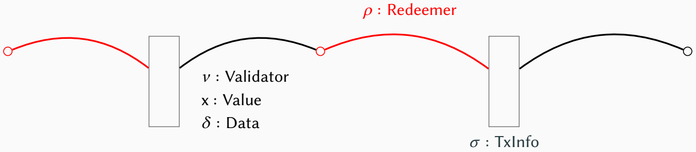
Example: Asynchronous Multi-signature Contract



Pay value (v) to payee (p) until deadline (d)

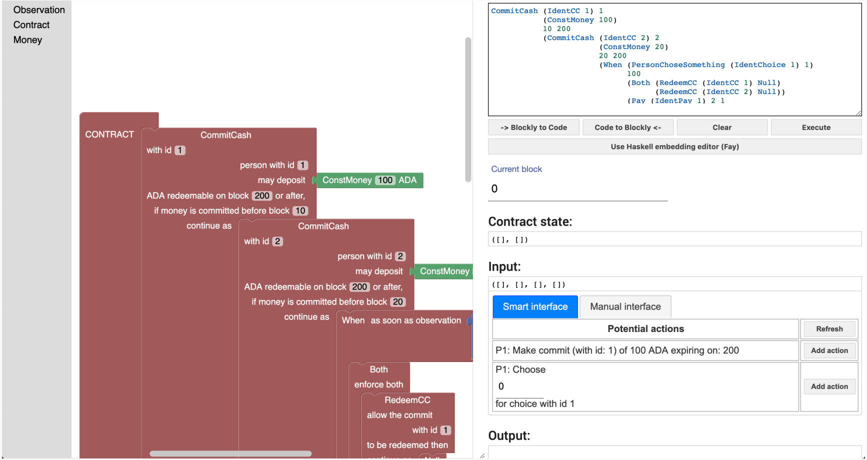


- $\delta \in \{\text{Holding}, \text{Collecting}\}$
- $\rho \in \{\text{Propose}, \text{Add}, \text{Cancel}, \text{Pay}\}$



$$v(\rho,\ x,\ \delta,\ \sigma) \stackrel{?}{=} \text{True}$$

Example in the Wild: Marlowe



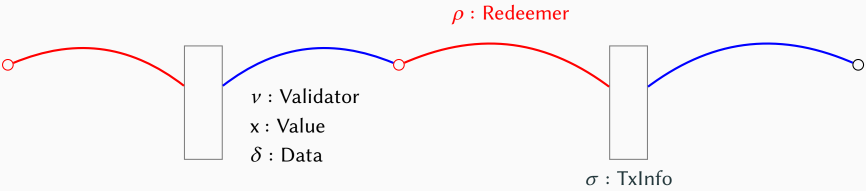
EUTXO, Formally...

Enhanced Scripting: Data Values

1. Data values additionally carried by outputs
    - allows a contract to carry data without changing its code
    - otherwise we could not identify a contract by its code's hash

Enhanced Scripting: Contract Continuity

2. More information about the transaction available to the validator
    - allows inspection of the transaction's outputs, thus supporting contract continuity (i.e. outputs use the expected validator)



$$\nu(\rho,\ x,\ \delta,\ \sigma) \stackrel{?}{=} \text{True}$$

Enhanced Scripting: Validity Intervals & Determinism

3. Transactions have (restricted) access to time
    - addition transaction field: validity interval
    - specifies a time interval, in which the transaction must be processed
    - in contrast to allowing access to the current time
        - allows for deterministic script execution
        - we can pre-calculate consumed resources/time
        - a user can simulate execution locally

Validity of EUTXO Transactions (I)

1. **The current tick is within the validity interval**

$$\mathrm{currentTick} \in t.\mathit{validityInterval}$$

2. **All outputs have non-negative values**

$$\forall o \in t.\mathit{outputs},\ \ o.\mathit{value} \geq 0$$

3. **All inputs refer to unspent outputs**

$$\{i.\mathit{outputRef} : i \in t.\mathit{inputs}\} \subseteq \mathrm{unspentOutputs}(l)$$

4. **Value is preserved (ignoring fees)**

$$\sum_{i \in t.\mathit{inputs}} \mathrm{getSpentOutput}(i, l).\mathit{value} = \sum_{o \in t.\mathit{outputs}} o.\mathit{value}$$

Validity of EUTXO Transactions (II)

5. **No output is double spent**

$$\text{If } i_1, i_2 \in t.\mathit{inputs} \text{ and } i_1.\mathit{outputRef} = i_2.\mathit{outputRef} \text{ then } i_1 = i_2$$

6. **All inputs validate**

$$\forall i \in t.\mathit{inputs},\ [\![i.\mathit{validator}]\!](i.\mathit{data},\ i.\mathit{redeemer},\ \mathrm{toTxInfo}(t, i)) = \mathrm{true}$$

7. **Validator scripts match output addresses**

$$\forall i \in t.\mathit{inputs},\ \mathrm{scriptAddr}(i.\mathit{validator}) = \mathrm{getSpentOutput}(i, l).\mathit{addr}$$

8. **Data values match output hashes**

$$\forall i \in t.\mathit{inputs},\ \mathrm{dataHash}(i.\mathit{data}) = \mathrm{getSpentOutput}(i, l).\mathit{dataHash}$$

Expressiveness of EUTXO

Constraint Emitting Machines (CEM)

To formally reason about the expressiveness of EUTXO, we introduce a specific form of state machines:

- $\textbf{step} : \textbf{State} \rightarrow \textbf{Input} \rightarrow \textbf{Maybe}\ (\textbf{State} \times \textbf{TxConstraints})$

Similar to Mealy machines (FSM + output), but differ in some aspects:

1. No notion of initial states
2. Cannot transition out of a final state
3. Blockchain-specific output values (TxConstraints)
    - e.g. for the **Pay** move of Multisig, $p \in \{o.address : o \in tx.outputs\}$

Behavioural Equivalence: Notation

- A ledger l corresponds to a CEM state s:

$$l \sim s$$

- New (valid) transaction submitted to ledger l:

$$l \xrightarrow{tx} l'$$

- Valid CEM transition from source state s to target state s', using input symbol i and emitting constraints $tx^{\equiv}$:

$$s \xrightarrow{i} (s', tx^{\equiv})$$

Given a smart contract, expressed as a CEM C, we can derive the validator script that disallows any invalid transitions:

$$\mathrm{validator}_C(s, i, \mathit{txInfo}) = \begin{cases} \mathrm{true} & \mathit{if}\ s \xrightarrow{i} (s', tx^{\equiv}) \\ & \mathit{and}\ \mathrm{satisfies}(\mathit{txInfo}, tx^{\equiv}) \\ & \mathit{and}\ \mathrm{checkOutputs}(s', \mathit{txInfo}) \\ \mathrm{false} & \mathit{otherwise} \end{cases}$$

Proposition 1 (Soundness)

Given a valid CEM transition, we can construct a new valid transaction, such that the resulting ledger corresponds to the target CEM state:

$$\frac{s \xrightarrow{i} (s', tx^{\equiv}) \quad l \sim s}{\exists tx\, l' \,.\, l \xrightarrow{tx} l' \;\wedge l' \sim s'}\ \textsc{sound}$$

Proposition 2 (Completeness)

Given a new valid transaction on the ledger, it is either irrelevant to the state machine or corresponds to a valid CEM transition:

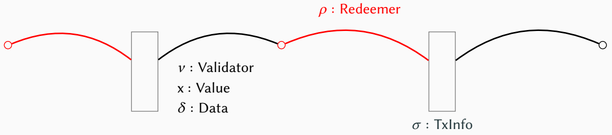
$$\frac{l \xrightarrow{tx} l' \quad l \sim s}{l' \sim s \;\vee\; \exists i\, s'\; tx^{\equiv} \,.\, s \xrightarrow{i} (s', tx^{\equiv})}\ \textsc{complete}$$

Related Work

- Bitcoin Covenants [Möser et al. @ FC'16]
  - Allows restricting how output values will be used in the future
  - Major inspiration for our introduction of *data values*
- Bitcoin Modelling Language (BitML) [Bartoletti et al. @ CCS'18]
  - Process-calculus with automata-based operational semantics
  - Compiles down to standard Bitcoin transactions
  - Complicated translation and requires off-chain communication
- Scilla [Sergey et al. @ OOPSLA'19]
  - For Ethereum-like contracts, using *communicating automata*
  - Embedded in Coq, allows proving temporal (hyper-)properties
- Timed Automata [Andrychowicz et al. @ FORMATS'14]
  - Pragmatic model checking of Bitcoin contracts using UPPAAL
  - Does not come with formal guarantees though

Contributions

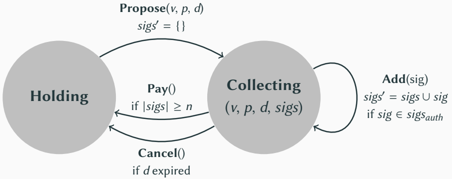
- **Detailed description of the Extended UTXO model (EUTXO)**



$$v(\rho,\ x,\ \delta,\ \sigma) \stackrel{?}{=} \text{True}$$

- **Formalization in** Agda

- **Proof of bisimulation with a specific form of state machines**



Questions?

Ledger Primitives

Quantity	an amount of currency
Tick	a tick
Address	an "address" in the blockchain
Data	a type of structured data
DataHash	the hash of a value of type Data
TxId	the identifier of a transaction
txId : Tx → TxId	get a transaction's identifier
Script	the (opaque) type of scripts
scriptAddr : Script → Address	the address of a script (i.e. its hash)
dataHash : Data → DataHash	the hash of a data value
$[\![\_]\!]$: Script → Args → Bool	applying a script to its arguments

Defined Types

$$
\begin{aligned}
\textbf{Output} \quad &= \quad (\mathit{value} : \mathrm{Quantity}, \mathit{addr} : \mathrm{Address}, \mathit{dataHash} : \mathrm{DataHash}) \\
\textbf{OutputRef} \quad &= \quad (\mathit{id} : \mathrm{TxId}, \mathit{index} : \mathbb{N}) \\
\textbf{Input} \quad &= \quad (\mathit{outputRef} : \mathrm{OutputRef}, \mathit{validator} : \mathrm{Script}, \\
&\qquad\ \mathit{data} : \mathrm{Data}, \mathit{redeemer} : \mathrm{Data}) \\
\textbf{Tx} \quad &= \quad (\mathit{inputs} : \mathrm{Set}[\mathrm{Input}], \mathit{outputs} : \mathrm{List}[\mathrm{Output}], \\
&\qquad\ \mathit{validityInterval} : \mathrm{Interval}[\mathrm{Tick}]) \\
\textbf{Ledger} \quad &= \quad \mathrm{List}[\mathrm{Tx}]
\end{aligned}
$$