# 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

## Bitcoin

- Based on *unspent transaction outputs* (UTXO)
- Simple, non Turing-complete smart contracts in SCRIPT

## Ethereum

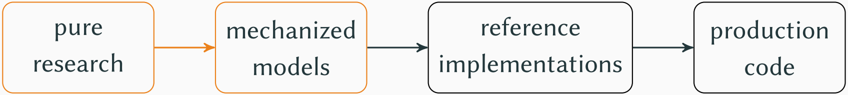
- Account-based
- Turing-complete smart contracts in Solidity/EVM

## Cardano (IOHK)

- UTXO-based + extensions
- Turing-complete smart contracts in Plutus

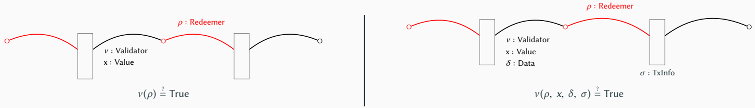
# Methodology

- 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



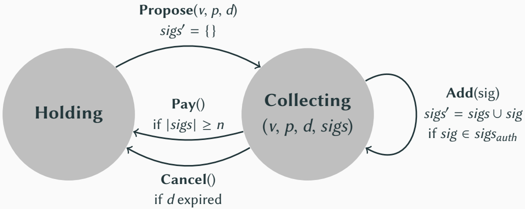
# Contributions

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



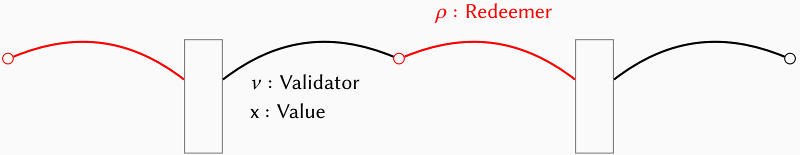

- **Formalization in** Agda

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

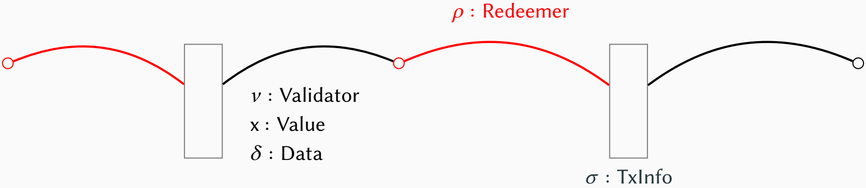


# EUTXO, Informally...

# UTXO vs EUTXO

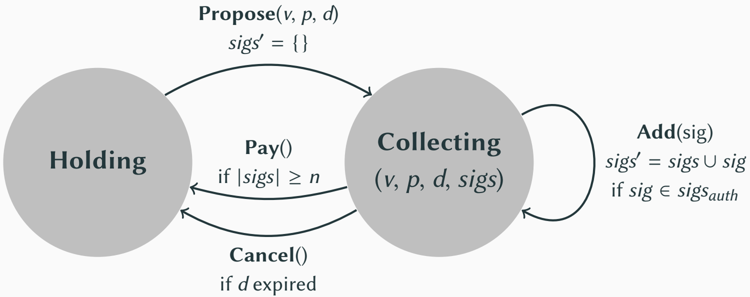



$$\nu(\rho) \stackrel{?}{=} \mathsf{True}$$

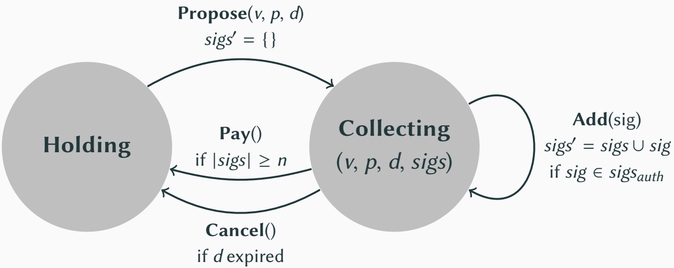



$$\nu(\rho,\ x,\ \delta,\ \sigma) \stackrel{?}{=} \mathsf{True}$$

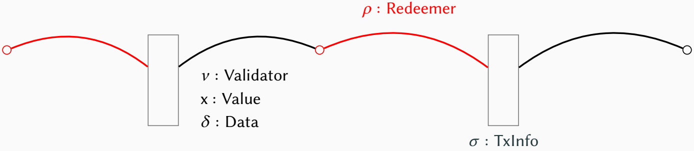
# Example: Asynchronous Multi-signature Contract




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

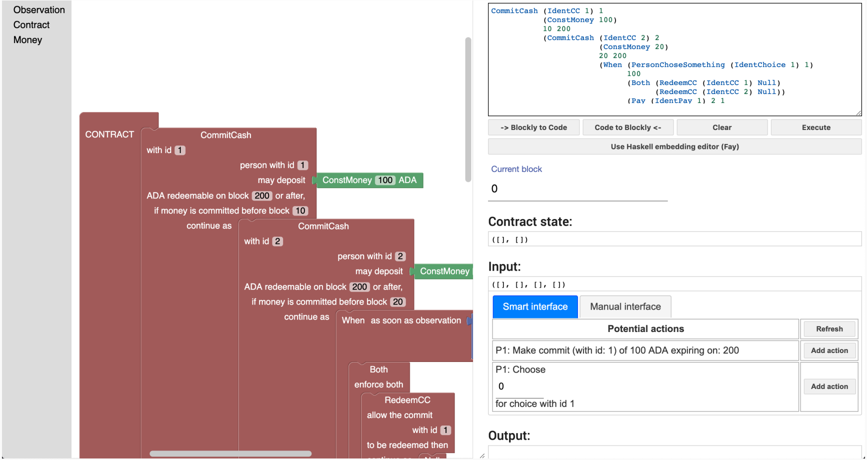



- $\delta \in \{\text{Holding}, \text{Collecting}\}$
- $\rho \in \{\text{Propose}, \text{Add}, \text{Cancel}, \text{Pay}\}$




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

# Example in the Wild: Marlowe



# EUTXO, Formally...

# Enhanced Scripting: Data Values & Contract Continuity

1. Data values additionally carried by outputs
   - passed as extra argument of type Data during validation
   - allows a contract to carry data without changing its code
   - otherwise we could not identify a contract by its code's hash
2. More information about the transaction available to the validator
   - passed as extra argument of type TxInfo during validation
   - allows inspection of the transaction's outputs, thus supporting contract continuity (i.e. outputs use the expected validator)

# 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

# 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

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

# 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**

$$t.\mathit{inputs} \subseteq \mathrm{unspentOutputs}(l)$$

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**

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

6. **All inputs validate**

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

7. **Validator scripts match output addresses**

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

8. **Data values match output hashes**

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

# Expressiveness of EUTXO

# Constraint Emitting Machines (CEM)

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

- Type of states S, type of inputs I
- final : $S \rightarrow$ Bool
- step : $S \rightarrow I \rightarrow$ Maybe ($S \times$ TxConstraints)

These are similar to Mealy machines, 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)
    - these typically are first-order equality constraints on the fields of Tx

# 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:

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

# Behavioural Equivalence: Weak Bisimulation

**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:

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

**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:

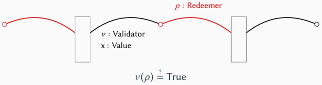
$$\frac{l \xrightarrow{tx} l' \qquad l \sim s}{l' \sim s \ \vee\ \exists i\, s'\ tx^{\equiv} \,.\, s \xrightarrow{i} (s', tx^{\equiv})}\ \textsc{complete}$$

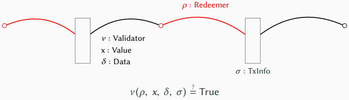
# Related Work

- 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
  - Quite 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

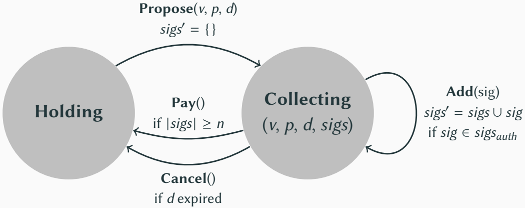
- **Detailed description of the Extended UTXO model (EUTXO)**




- **Formalization in** Agda

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



# Questions?