

Formal specification of the Cardano ledger, mechanized in Agda

Andre Knispel, Orestis Melkonian, James Chapman, Alasdair Hill, Joosep Jääger, William DeMeo, Ulf Norell

7 April 2024, FMBC, Luxembourg

Introduction

Much (formally-verified) meta-theoretical work on EUTxO

- ...but all on simplified and idealized settings
- ...none cover the full feature set of a realistic blockchain like Cardano

Much (formally-verified) meta-theoretical work on EUTxO

- ...but all on simplified and idealized settings
- ...none cover the full feature set of a realistic blockchain like Cardano

Previous iterations of Cardano's ledger specification written informally on paper

- Lack the rigor of a mechanized formal artifact
- Are not executable and thus require a separate prototype to be implemented

Separation of Concerns

- Networking: deals with sending messages across the internet
- Consensus: establishes a common order of valid blocks
- **Ledger**: decides whether a sequence of blocks is valid

Separation of Concerns

- Networking: deals with sending messages across the internet
- Consensus: establishes a common order of valid blocks
- **Ledger**: decides whether a sequence of blocks is valid

Stay as close to the previous (LaTeX) specification as possible

- Use of **set theory**
- System evolution formulated as **state machines**

Separation of Concerns

- Networking: deals with sending messages across the internet
- Consensus: establishes a common order of valid blocks
- **Ledger**: decides whether a sequence of blocks is valid

Stay as close to the previous (LaTeX) specification as possible

- Use of **set theory**
- System evolution formulated as **state machines**

Use the Agda proof assistant to produce a **readable** mechanized specification.

Agda Preliminaries

Easy to define **common operations**:

$$_ \subseteq _ _ \equiv^e _ : \{A : \text{Type}\} \rightarrow \mathcal{P} A \rightarrow \mathcal{P} A \rightarrow \text{Type}$$

$$X \subseteq Y = \forall \{x\} \rightarrow x \in X \rightarrow x \in Y$$

$$X \equiv^e Y = X \subseteq Y \times Y \subseteq X$$

Easy to define **common operations**:

$$_ \subseteq _ \equiv^e _ : \{A : \text{Type}\} \rightarrow \mathcal{P} A \rightarrow \mathcal{P} A \rightarrow \text{Type}$$

$$X \subseteq Y = \forall \{x\} \rightarrow x \in X \rightarrow x \in Y$$

$$X \equiv^e Y = X \subseteq Y \times Y \subseteq X$$

Finite **maps** as set of tuples:

$$_ \rightarrow _ : \text{Type} \rightarrow \text{Type} \rightarrow \text{Type}$$

$$A \rightarrow B = \exists \lambda (\mathcal{R} : \mathcal{P} (A \times B)) \rightarrow$$

$$\forall \{a \ b \ b'\} \rightarrow (a \ , \ b) \in \mathcal{R} \rightarrow (a \ , \ b') \in \mathcal{R} \rightarrow b \equiv b'$$

$$\Gamma \vdash s \xrightarrow[X]{b} s'$$

$$\Gamma \vdash s \xrightarrow[X]{b} s'$$

$_ \vdash _ \rightarrow (_ _) _ : Env \rightarrow State \rightarrow Signal \rightarrow State \rightarrow \text{Type}$

Depicting transitions: Triptychs

Environments
(Signals)

States

Possible transitions

Sequencing transitions: Reflexive-transitive closure

`data _ \vdash _ \rightarrow ⟦_⟧*_ : Env \rightarrow State \rightarrow List Signal \rightarrow State \rightarrow Type where`

`step :`

`base :`

$$\frac{}{\Gamma \vdash s \rightarrow \llbracket [] \rrbracket * s}$$

- $\Gamma \vdash s \rightarrow \llbracket b \quad \rrbracket s'$
- $\Gamma \vdash s' \rightarrow \llbracket bs \quad \rrbracket * s''$

$$\frac{}{\Gamma \vdash s \rightarrow \llbracket b :: bs \rrbracket * s''}$$

Formalization

Basic entities / assumptions

- Crypto:

`VKey Sig Ser : Type`

`isSigned : VKey → Ser → Sig → Type`

Basic entities / assumptions

- Crypto:

VKey Sig Ser : Type

isSigned : VKey → Ser → Sig → Type

- Addresses:

Credential = KeyHash ∪ ScriptHash

record BaseAddr : Type where

pay : Credential

stake : Credential

Addr = BaseAddr ∪ ...

record RwdAddr : Type where

stake : Credential

Basic entities / assumptions

- Crypto:

VKey Sig Ser : Type

isSigned : VKey → Ser → Sig → Type

- Addresses:

Credential = KeyHash ∪ ScriptHash

record BaseAddr : Type where

pay : Credential

stake : Credential

Addr = BaseAddr ∪ ...

record RwdAddr : Type where

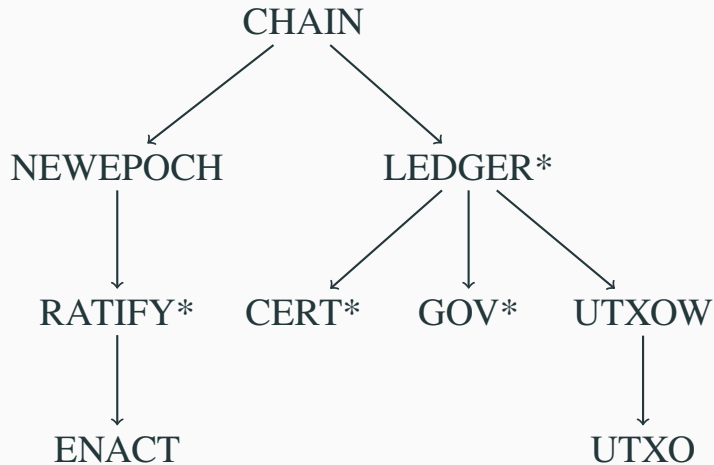
stake : Credential

- Tunable parameters:

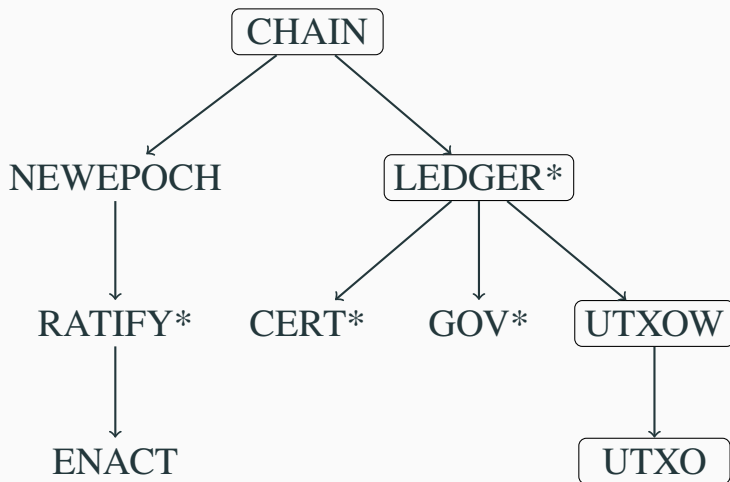
record PParams : Type where

maxBlockSize maxTxSize a b : ℕ

The hierarchy of transitions



The hierarchy of transitions



CHAIN: processing blocks

```
record Block : Type where
  field ts    : List Tx
        slot : Slot
```

```
record NewEpochState : Type where
  field lastEpoch : Epoch; acnt : Acnt
        ls : LState; es : EnactState
        fut : RatifyState
```

```
record ChainState : Type where
  field newEpochState : NewEpochState
```

CHAIN :

- $\text{mkNewEpochEnv } s \vdash s . \text{newEpochState} \rightarrow \langle \text{epoch slot}, \text{NEWEPOCH} \rangle \text{ nes}$
- $\llbracket \text{slot} \otimes \text{constitution} . \text{proj}_1 . \text{proj}_2 \otimes \text{pparams} . \text{proj}_1 \otimes \text{es} \rrbracket$
 $\vdash \text{nes} . \text{ls} \rightarrow \langle \text{ts}, \text{LEDGER*} \rangle \text{ ls'}$

$_ \vdash s \rightarrow \langle b, \text{CHAIN} \rangle \text{updateChainState } s \text{ nes}$

LEDGER: processing transactions

```
record LEnv : Type where
```

```
  slot      : Slot
```

```
  ppolicy   : Maybe ScriptHash
```

```
  pparams   : PParams
```

```
  enactState : EnactState
```

```
record LState : Type where
```

```
  utxoSt     : UTxOState
```

```
  govSt      : GovState
```

```
  certState  : CertState
```

LEDGER :

- $\text{mkUTxOEnv } \Gamma \vdash \text{utxoSt} \rightarrow \langle \text{tx}, \text{UTXOW} \rangle \text{ utxoSt}'$
- $\llbracket \text{epoch slot} \otimes \text{pparams} \otimes \text{txvote} \otimes \text{txwdrls} \rrbracket \vdash \text{certState} \rightarrow \langle \text{txcerts}, \text{CERT*} \rangle \text{ certState}'$
- $\llbracket \text{txid} \otimes \text{epoch slot} \otimes \text{pparams} \otimes \text{enactState} \rrbracket \vdash \text{govSt} \rightarrow \langle \text{txgov txb}, \text{GOV*} \rangle \text{ govSt}'$

$$\Gamma \vdash s \rightarrow \langle \text{tx}, \text{LEDGER} \rangle \llbracket \text{utxoSt}' \otimes \text{govSt}' \otimes \text{certState}' \rrbracket$$

LEDGER: The transaction type

Ix $TxId$: Type

$TxIn$ = $TxId \times Ix$

$TxOut$ = $Addr \times Value \times Maybe\ DataHash$

$UTxO$ = $TxIn \rightarrow TxOut$

record $TxBody$: Type where

$txins$: $P\ TxIn$

$txouts$: $Ix \rightarrow TxOut$

$txfee$: Coin

$txvote$: List GovVote

$txprop$: List GovProposal

$txsize$: \mathbb{N}

$txid$: $TxId$

record $TxWitnesses$: Type where

$vkSigs$: $VKey \rightarrow Sig$

$scripts$: $P\ Script$

record Tx : Type where

$body$: $TxBody$

$wits$: $TxWitnesses$

UTXOW: witnesses & scripts

UTXOW-inductive :

- $\text{witsVKeyNeeded ppolicy utxo txb} \subseteq \text{witsKeyHashes}$
- $\text{scriptsNeeded ppolicy utxo txb} \equiv^e \text{witsScriptHashes}$
- $\forall [(vk, \sigma) \in vkSigs] \text{ isSigned } vk (\text{txidBytes txid}) \sigma$
- $\forall [s \in \text{scriptsP1}] \text{ validP1Script witsKeyHashes txvldt } s$
- $\Gamma \vdash s \rightarrow \langle tx, \text{UTXO} \rangle s'$

$$\Gamma \vdash s \rightarrow \langle tx, \text{UTXOW} \rangle s'$$

UTXO: the “core” transition

```
record UTxOEnv : Type where
  slot      : Slot
  pparams   : PParams
  Deposits = DepositPurpose → Coin
```

```
record UTxOState : Type where
  utxo      : UTxO
  deposits  : Deposits
  fees donations : Coin
```

UTXO-inductive :

- $\text{txins} \neq \emptyset$
- $\text{minfee pp tx} \leq \text{txfee}$
- $\text{consumed pp s txb} \equiv \text{produced pp s txb}$
- $\text{txins} \subseteq \text{dom utxo}$
- $\text{txsize} \leq \text{maxTxSize pp}$
- $\text{coin mint} \equiv 0$

```
[ (utxo | txins ) U outs txb
  , updateDeposits pp txb deposits
  , fees + txfee
  , donations + txdonation ]
```

$\Gamma \vdash s \rightarrow \langle tx, \text{UTXO} \rangle$

UTXO: the property of Value Preservation

Property (Value preservation)

- $tx.body.txid \notin \text{map } proj_1 (\text{dom } (s.utxo))$
- $\Gamma \vdash s \rightarrow \langle tx, UTXO \rangle s'$

$$getCoin\ s \equiv getCoin\ s'$$

Compiling to executable Haskell

Proving transitions are computational

```
record Computational ( $\_ \vdash \_ \dashv \_ , X \triangleright \_ : C \rightarrow S \rightarrow \text{Sig} \rightarrow S \rightarrow \text{Type}$ ) : Type where
  field compute          :  $C \rightarrow S \rightarrow \text{Sig} \rightarrow \text{Maybe } S$ 
      compute-correct :  $\text{compute } \Gamma s b \equiv \text{just } s' \Leftrightarrow \Gamma \vdash s \dashv b , X \triangleright s'$ 
```

Example: compiling the UTXOW transition (Agda source)

```
Computational-UTXOW : Computational  $\vdash \rightarrow \Downarrow$  _, UTXOW  $\Downarrow$  _
Computational-UTXOW = let H ,  $\Downarrow$  H? = UTXOW-inductive-premises {tx}{s}{ $\Gamma$ } where
  computeProof : Maybe $  $\exists$  ( $\Gamma \vdash s \rightarrow \Downarrow tx$  , UTXOW  $\Downarrow$  _)
  computeProof = case H? of  $\lambda$  where
    (yes (p1 , p2 , p3 , p4 , p5))  $\rightarrow$ 
      map2' (UTXOW-inductive... p1 p2 p3 p4 p5) <$> computeProof'  $\Gamma$  s tx
    (no _)  $\rightarrow$  nothing

  completeness :  $\forall$  s'  $\rightarrow$   $\Gamma \vdash s \rightarrow \Downarrow tx$  , UTXOW  $\Downarrow$  s'  $\rightarrow$  M.map proj1 computeProof  $\equiv$  just s'
  completeness s' (UTXOW-inductive... p1 p2 p3 p4 p5 h)
    rewrite dec-yes H? (p1 , p2 , p3 , p4 , p5) .proj2
    with computeProof'  $\Gamma$  s tx | completeness' _ _ _ _ h
    ... | just _ | refl = refl

utxowStep : UTxOEnv  $\rightarrow$  UTxOState  $\rightarrow$  Tx  $\rightarrow$  Maybe UTxOState
utxowStep = compute Computational-UTXOW
{-# COMPILER GHC utxowStep #-}
```

Example: running the UTXOW transition (Haskell target)

```
import Lib (utxowStep)
```

```
utxowSteps :: UTxOEnv -> UTxOState -> [Tx] -> Maybe UTxOState
```

```
utxowSteps = foldM . utxowStep
```

Example: running the UTXOW transition (Haskell target)

```
spec :: Spec
spec = describe "utxowSteps" $ it "results in the expected state" $
  utxowSteps initEnv initState [testTx1, testTx2] @?= Just (MkUTxOState
    { utxo = [ (1,0) .-> (a0, (890, Nothing))
              , (2,0) .-> (a2, (10,  Nothing))
              , (2,1) .-> (a1, (80,  Nothing)) ]
    , fees = 20 })
  where
    testTx1, testTx2 :: Tx

    initEnv :: UTxOEnv
    initEnv = MkUTxOEnv {slot = 0, pparams = ...}

    initUTxO :: UTxO
    initUTxO = [ (0, 0) .-> (a0, (1000, Nothing)) ]

    initState :: UTxOState
    initState = MkUTxOState {utxo = initUTxO, fees = 0}
```

Conclusion

Compilation issues:

- Automate away **boilerplate**
- Finalize **conformance-testing** integration
- Randomly test (proven) Agda statements by translating to **Quickcheck** properties
- Optimizations in implementation → **refinements** in formalization

Compilation issues:

- Automate away **boilerplate**
- Finalize **conformance-testing** integration
- Randomly test (proven) Agda statements by translating to **Quickcheck** properties
- Optimizations in implementation → **refinements** in formalization

Expand the scope of the formalization

- Prove more interesting meta-theoretical **properties**
- Cover previous **eras**: “keeping up with the past”
- Towards verifying **smart contracts**

Questions?

[https://intersectmbo.github.io/
formal-ledger-specifications/](https://intersectmbo.github.io/formal-ledger-specifications/)

