The UTxO Model and the BitML Calculus

A Formal Investigation in Agda

Orestis Melkonian

March 28, 2019

Utrecht University, The Netherlands

Overview

1. UTxO (github.com/omelkonian/formal-utxo)
   - Transactions/Ledgers
   - Validity
   - Example
   - Weakening Lemma
2. BitML (github.com/omelkonian/formal-bitml)
   - Contracts
   - Small-step Semantics
   - Example
3. Next Steps

UTxO

Basic Types

$$
\begin{aligned}
&\mathit{Address} = \mathbb{N} \\
&\mathit{Value} = \mathbb{N}
\end{aligned}
$$

$\textbf{record}\ \mathit{State} : \mathit{Set}\ \textbf{where}$

$\quad \textbf{field}\ \mathit{height} : \mathbb{N}$

$\qquad\qquad \vdots$

$\textbf{postulate}$

$\quad \_\# : \forall\ \{A : \mathit{Set}\} \to A \to \mathit{Address}$

$\quad \#\text{-}\mathit{injective} : \forall\ \{x\ y : A\} \to x\,\# \equiv y\,\# \to x \equiv y$

Inputs and Output References

```
record TxOutputRef : Set where
  constructor _@_
  field id    : Address
        index : ℕ

record TxInput {R D : Set} : Set where
  field outputRef : TxOutputRef
        redeemer  : State → R
        validator : State → Value → R → D → Bool
```

Transactions

```
module UTxO (addresses : List Address) where

record TxOutput {D : Set} : Set where
  field value      : Value
        address    : Index addresses
        dataScript : State → D

record Tx : Set where
  field inputs  : Set⟨ TxInput ⟩
        outputs : List TxOutput
        forge   : Value
        fee     : Value

Ledger : Set
Ledger = List Tx
```

Unspent Outputs

$$
\begin{aligned}
&\mathit{unspentOutputs} : \mathit{Ledger} \to \mathit{Set}\langle\ \mathit{TxOutputRef}\ \rangle \\
&\mathit{unspentOutputs}\ [\,] \qquad\qquad = \varnothing \\
&\mathit{unspentOutputs}\ (\mathit{tx} :: \mathit{txs}) = (\mathit{unspentOutputs}\ \mathit{txs} \setminus \mathit{spentOutputsTx}\ \mathit{tx}) \\
&\qquad\qquad\qquad\qquad\qquad\quad \cup\ \mathit{unspentOutputsTx}\ \mathit{tx} \\
&\quad \mathbf{where} \\
&\qquad \mathit{spentOutputsTx}\ ,\ \mathit{unspentOutputsTx} : \mathit{Tx} \to \mathit{Set}\langle\ \mathit{TxOutputRef}\ \rangle \\
&\qquad \mathit{spentOutputsTx} \qquad\quad = (\mathit{outputRef} <\$> \_)\mathit{inputs} \\
&\qquad \mathit{unspentOutputsTx}\ \mathit{tx} = ((\mathit{tx}\ \#)\ @\_)\ <\$>\ (\mathit{indices}\ (\mathit{outputs}\ \mathit{tx}))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{runValidation}\ \ :\ \ (i : \mathit{TxInput}) \to (o : \mathit{TxOutput}) \\
&\qquad\qquad\qquad \to D\ i \equiv D\ o \to \mathit{State} \to \mathit{Bool} \\
&\mathit{runValidation}\ i\ o\ \mathit{refl}\ \mathit{st} = \\
&\quad \mathit{validator}\ i\ \mathit{st}\ (\mathit{value}\ o)\ (\mathit{redeemer}\ i\ \mathit{st})\ (\mathit{dataScript}\ o\ \mathit{st})
\end{aligned}
$$

Validity I

```
record IsValidTx (tx : Tx) (l : Ledger) : Set where
field
  validTxRefs : ∀ i → i ∈ inputs tx →
    Any (λ t → t # ≡ id (outputRef i)) l

  validOutputIndices : ∀ i → (i∈ : i ∈ inputs tx) →
    index (outputRef i) <
      length (outputs (lookupTx l (outputRef i) (validTxRefs i i∈)))

  validOutputRefs : ∀ i → i ∈ inputs tx →
    outputRef i ∈ unspentOutputs l

  validDataScriptTypes : ∀ i → (i∈ : i ∈ inputs tx) →
    D i ≡ D (lookupOutput l (outputRef i) (validTxRefs i i∈)
                                          (validOutputIndices i i∈))
```

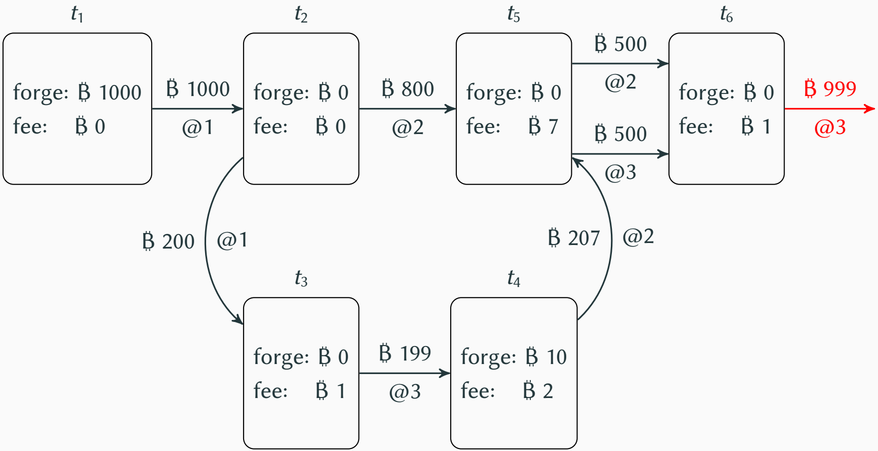
Validity II

$$
\begin{aligned}
&\mathit{preservesValues} :\\
&\quad \mathit{forge}\ \mathit{tx} + \mathit{sum}\ (\mathit{mapWith}{\in}\ (\mathit{inputs}\ \mathit{tx})\ \lambda\ \{i\}\ i{\in} \rightarrow\\
&\qquad \mathit{lookupValue}\ l\ i\ (\mathit{validTxRefs}\ i\ i{\in})\ (\mathit{validOutputIndices}\ i\ i{\in}))\\
&\qquad \equiv\\
&\quad \mathit{fee}\ \mathit{tx} + \mathit{sum}\ (\mathit{value}\ {<}\${>}\ \mathit{outputs}\ \mathit{tx})\\
&\mathit{noDoubleSpending} :\\
&\quad \mathit{noDuplicates}\ (\mathit{outputRef}\ {<}\${>}\ \mathit{inputs}\ \mathit{tx})\\
&\mathit{allInputsValidate} : \forall\ i \rightarrow (i{\in} : i \in \mathit{inputs}\ \mathit{tx}) \rightarrow\\
&\quad \mathbf{let}\ \mathit{out} = \mathit{lookupOutput}\ l\ (\mathit{outputRef}\ i)\ (\mathit{validTxRefs}\ i\ i{\in})\\
&\qquad\qquad (\mathit{validOutputIndices}\ i\ i{\in})\\
&\quad \mathbf{in}\qquad \forall\ (\mathit{st} : \mathit{State}) \rightarrow\\
&\qquad\qquad\quad T\ (\mathit{runValidation}\ i\ \mathit{out}\ (\mathit{validDataScriptTypes}\ i\ i{\in})\ \mathit{st})\\
&\qquad \times\ \mathit{to}\mathbb{N}\ (\mathit{address}\ \mathit{out}) \equiv (\mathit{validator}\ i)\ \#
\end{aligned}
$$

Valid Ledgers

```
data ValidLedger : Ledger → Set where
  • _⊢_ : (t : Tx)
        → IsValidTx t []
        → ValidLedger [ t ]
  _⊕_⊢_ : ValidLedger l
        → (t : Tx)
        → IsValidTx t l
        → ValidLedger (t :: l)
```

Example: Transaction Graph



Example: Setting Up

$$addresses : List\ Address$$
$$addresses = 1 :: 2 :: 3 :: [\,]$$

open import *UTxO addresses*

$$withScripts : TxOutputRef \rightarrow TxInput$$
$$\begin{aligned} withScripts\ tin = \mathbf{record}\ \{ & outputRef = tin \\ ; & redeemer \ \ = \lambda\ \_ \rightarrow 0 \\ ; & validator \ \ = \lambda\ \_\ \_\ \_\ \_ \rightarrow true \} \end{aligned}$$

$$\text{฿}\ \_@\_ : Value \rightarrow Index\ addresses \rightarrow TxOutput$$
$$\begin{aligned} \text{฿}\ v\ @\ addr = \mathbf{record}\ \{ & value \qquad = v \\ ; & address \quad = addr \\ ; & dataScript = \lambda\ \_ \rightarrow 0 \} \end{aligned}$$

Example: Definitions of Transactions I

$$
\begin{aligned}
& t_1 \,,\, t_2 \,,\, t_3 \,,\, t_4 \,,\, t_5 \,,\, t_6 : \mathit{Tx} \\
& t_1 = \mathbf{record}\,\{\, \mathit{inputs} = [\,] \\
& \qquad\qquad ;\, \mathit{outputs} = [\,\text{฿}\ 1000\ @\ 0\,] \\
& \qquad\qquad ;\, \mathit{forge} = \text{฿}\ 1000 \\
& \qquad\qquad ;\, \mathit{fee} = \text{฿}\ 0 \,\} \\
& t_2 = \mathbf{record}\,\{\, \mathit{inputs} = [\,\mathit{withScripts}\ t_{10}\,] \\
& \qquad\qquad ;\, \mathit{outputs} = \text{฿}\ 800\ @\ 1 :: \text{฿}\ 200\ @\ 0 :: [\,] \\
& \qquad\qquad ;\, \mathit{forge} = \text{฿}\ 0 \\
& \qquad\qquad ;\, \mathit{fee} = \text{฿}\ 0 \,\} \\
& t_3 = \mathbf{record}\,\{\, \mathit{inputs} = [\,\mathit{withScripts}\ t_{21}\,] \\
& \qquad\qquad ;\, \mathit{outputs} = [\,\text{฿}\ 199\ @\ 2\,] \\
& \qquad\qquad ;\, \mathit{forge} = \text{฿}\ 0 \\
& \qquad\qquad ;\, \mathit{fee} = \text{฿}\ 1 \,\}
\end{aligned}
$$

Example: Definitions of Transactions II

$$
\begin{aligned}
t_4 = \textbf{record}\ \{ & \mathit{inputs} &&= [\mathit{withScripts}\ t_{30}] \\
; & \mathit{outputs} &&= [฿\ 207\ @\ 1] \\
; & \mathit{forge} &&= ฿\ 10 \\
; & \mathit{fee} &&= ฿\ 2\} \\
t_5 = \textbf{record}\ \{ & \mathit{inputs} &&= \mathit{withScripts}\ t_{20} :: \mathit{withScripts}\ t_{40} :: [\,] \\
; & \mathit{outputs} &&= ฿\ 500\ @\ 1 :: ฿\ 500\ @\ 2 :: [\,] \\
; & \mathit{forge} &&= ฿\ 0 \\
; & \mathit{fee} &&= ฿\ 7\} \\
t_6 = \textbf{record}\ \{ & \mathit{inputs} &&= \mathit{withScripts}\ t_{50} :: \mathit{withScripts}\ t_{51} :: [\,] \\
; & \mathit{outputs} &&= [฿\ 999\ @\ 2] \\
; & \mathit{forge} &&= ฿\ 0 \\
; & \mathit{fee} &&= ฿\ 1\}
\end{aligned}
$$

Example: Correct-by-construction Ledger

$$
\begin{aligned}
&\mathit{ex\text{-}ledger} : \mathit{Ledger}\\
&\mathit{ex\text{-}ledger} = \;\bullet\; t_1 \vdash \textbf{record}\; \{\, \mathit{validTxRefs} = \lambda\, i\, ()\\
&\qquad\qquad\qquad\qquad ;\, \mathit{validOutputIndices} = \lambda\, i\, ()\\
&\qquad\qquad\qquad\qquad ;\, \mathit{validOutputRefs} = \lambda\, i\, ()\\
&\qquad\qquad\qquad\qquad ;\, \mathit{validDataScriptTypes} = \lambda\, i\, ()\\
&\qquad\qquad\qquad\qquad ;\, \mathit{preservesValues} = \mathit{refl}\\
&\qquad\qquad\qquad\qquad ;\, \mathit{noDoubleSpending} = \mathit{tt}\\
&\qquad\qquad\qquad\qquad ;\, \mathit{allInputsValidate} = \lambda\, i\, () \,\}\\
&\qquad\qquad\; \oplus\; t_2 \vdash \textbf{record}\; \{\ldots\}\\
&\qquad\qquad\qquad \vdots\\
&\qquad\qquad\; \oplus\; t_6 \vdash \textbf{record}\; \{\ldots\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{utxo} : \mathit{list}\;(\mathit{unspentOutputs}\;\mathit{ex\text{-}ledger}) \equiv [\, t_{60} \,]\\
&\mathit{utxo} = \mathit{refl}
\end{aligned}
$$

Weakening Lemma

$$\mathit{Ledger}' : \mathit{List}\ \mathit{Address} \to \mathit{Set}$$
$$\mathit{Ledger}'\ \mathit{as} = \mathit{Ledger}\ \textbf{where open import}\ \mathit{UTxO}\ \mathit{as}$$

$$\vdots$$

$$\mathit{weakenTxOutput} : \mathit{Prefix}\ \mathit{as}\ \mathit{bs} \to \mathit{TxOutput}'\ \mathit{as} \to \mathit{TxOutput}'\ \mathit{bs}$$
$$\mathit{weakenTxOutput}\ \mathit{pr}\ \mathit{txOut} = \mathit{txOut}\ \{\mathit{address} = \mathit{inject} \leqslant \ldots\}$$
$$\quad \textbf{where open import}\ \mathit{UTxO}\ \mathit{bs}$$

$$\mathit{weakening} : \forall\ \{\mathit{as}\ \mathit{bs} : \mathit{List}\ \mathit{Address}\}\ \{\mathit{tx} : \mathit{Tx}'\ \mathit{as}\}\ \{l : \mathit{Ledger}'\ \mathit{as}\}$$
$$\quad \to (\mathit{pr} : \mathit{Prefix}\ \mathit{as}\ \mathit{bs})$$
$$\quad \to \mathit{IsValidTx}'\ \mathit{as}\ \mathit{tx}\ l$$

$$\quad \to \mathit{IsValidTx}'\ \mathit{bs}\ (\mathit{weakenTx}\ \mathit{pr}\ \mathit{tx})\ (\mathit{weakenLedger}\ \mathit{pr}\ l)$$
$$\mathit{weakening} = \ldots$$

BitML

Basic Types

module *Types* (*Participant* : *Set*) (*Honest* : *List*$^+$ *Participant*) **where**

$$\begin{aligned}
\mathit{Time} &= \mathbb{N}\\
\mathit{Value} &= \mathbb{N}\\
\mathit{Secret} &= \mathit{String}\\
\mathit{Deposit} &= \mathit{Participant} \times \mathit{Value}
\end{aligned}$$

data *Arith* : *List Secret* $\rightarrow$ *Set* **where** ...

$[\![\_]\!]_n : \mathit{Arith}\ s \rightarrow \mathbb{N}$

$[\![\_]\!]_n = \ldots$

data *Predicate* : *List Secret* $\rightarrow$ *Set* **where** ...

$[\![\_]\!]_b : \mathit{Predicate}\ s \rightarrow \mathit{Bool}$

$[\![\_]\!]_b = \ldots$

Contract Preconditions

```
data Precondition : List Value -- volatile deposits
                  → List Value -- persistent deposits
                  → Set where
  -- volatile deposit
  _?_ : Participant → (v : Value) → Precondition [ v ] [ ]
  -- persistent deposit
  _!_ : Participant → (v : Value) → Precondition [ ] [ v ]
  -- committed secret
  _#_ : Participant → Secret → Precondition [ ] [ ]
  -- conjunction
  _∧_ : Precondition vsᵥ vsₚ → Precondition vsᵥ′ vsₚ′
      → Precondition (vsᵥ ++ vsᵥ′) (vsₚ ++ vsₚ′)
```

Contracts I

$$
\begin{aligned}
&\textbf{data } \mathit{Contract} : \mathit{Value} \quad \text{-- the monetary value it carries}\\
&\qquad\qquad\qquad \to \mathit{Values} \quad \text{-- the (volatile) deposits it presumes}\\
&\qquad\qquad\qquad \to \mathit{Set}\ \textbf{where}
\end{aligned}
$$

$$
\begin{aligned}
&\quad \text{-- collect deposits and secrets}\\
&\quad \mathit{put\ \_\ reveal\ \_\ if\ \_} \Rightarrow \_ \vdash \_ :\\
&\qquad (\mathit{vs} : \mathit{List\ Value}) \to (s : \mathit{Secrets}) \to \mathit{Predicate}\ s'\\
&\qquad \to \mathit{Contract}\ (v + \mathit{sum}\ \mathit{vs})\ \mathit{vs}' \to s' \subseteq s\\
&\qquad \to \mathit{Contract}\ v\ (\mathit{Interleave}\ \mathit{vs}'\ \mathit{vs})
\end{aligned}
$$

$$
\begin{aligned}
&\quad \text{-- transfer the remaining balance to a participant}\\
&\quad \mathit{withdraw} : \mathit{Participant} \to \mathit{Contract}\ v\ \mathit{vs}
\end{aligned}
$$

Contracts II

```
-- split the balance across different branches
split : (cs : List (∃[ v ] Contract v vs))
  → Contract (sum (proj₁ <$> cs)) vs

-- wait for participant's authorization
_ : _ : Participant → Contract v vs → Contract v vs

-- wait until some time passes
after _ : _ : Time → Contract v vs → Contract v vs
```

Advertisements

```
record Advertisement (v : Value) (vsᶜ vsᵛ vsᵖ : List Value) : Set where
  constructor _⟨_⟩⊢_
  field G     : Precondition vsᵛ vsᵖ
        C     : List (Contract v vsᶜ)
        valid : length vsᶜ ≤ length vsᵛ
              × participantsᵍ G ++ participantsᶜ C
                    ⊆
                (participant <$> persistentDepositsᵖ G)
              × v ≡ sum vsᵖ
```

Example Advertisement

open *BitML* $(A \mid B)$ $[A]^{+}$

$$
\begin{aligned}
&\mathit{ex\text{-}ad} : \mathit{Advertisement}\ 5\ [200]\ [200]\ (3 :: 2 :: [\,])\\
&\mathit{ex\text{-}ad} = \langle\ B\ !\ 3 \wedge A\ !\ 2 \wedge A\ ?\ 200\ \rangle\\
&\qquad\qquad \mathit{split}\ \ (\ 2 \multimap \mathit{withdraw}\ B\\
&\qquad\qquad\qquad\quad \oplus\ 2 \multimap \mathit{after}\ 42 : \mathit{withdraw}\ A\\
&\qquad\qquad\qquad\quad \oplus\ 1 \multimap \mathit{put}\ [200] \Rightarrow B : \mathit{withdraw}\ \{201\}\ A\\
&\qquad\qquad\qquad\quad)\\
&\qquad\qquad \vdash \ldots
\end{aligned}
$$

Small-step Semantics: Actions I

$$AdvertisedContracts = List\ (\exists[\ v\ , \ldots, vs^{p}\ ]\ Advertisement\ v\ \ldots\ vs^{p})$$
$$ActiveContracts = List\ (\exists[\ v\ , vs\ ]\ List\ (Contract\ v\ vs))$$

```
data Action (p : Participant) -- the participant that authorises this action
    : AdvertisedContracts      -- the contract advertisments it requires
    → ActiveContracts          -- the active contracts it requires
    → Values                   -- the deposits it requires from this participant
    → List Deposit             -- the deposits it produces
    → Set where
```

Small-step Semantics: Actions II

```
-- commit secrets to stipulate an advertisement
#▷_ : (ad : Advertisement v vsᶜ vsᵛ vsᵖ)
     → Action p [v , vsᶜ , vsᵛ , vsᵖ , ad] [] [] []

-- spend x to stipulate an advertisement
_▷ˢ_ : (ad : Advertisement v vsᶜ vsᵛ vsᵖ)
      → (i : Index vsᵖ)
      → Action p [v , vsᶜ , vsᵛ , vsᵖ , ad] [] [vsᵖ !! i] []

-- pick a branch
_▷ᵇ_ : (c : List (Contract v vs))
      → (i : Index c)
      → Action p [] [v , vs , c] [] []

⋮
```

Small-step Semantics: Actions Example

$$\textit{ex-spend} : \textit{Action } A \; [5 \, , [200] \, , [200] \, , 3 :: 2 :: [\,] \, , \textit{ex-ad}] \; [\,] \; [2] \; [\,]$$
$$\textit{ex-spend} = \textit{ex-ad} \rhd^{s} 1$$

Small-step Semantics: Configurations I

$$
\begin{aligned}
&\textbf{data}\ \mathit{Configuration}' : \quad \text{-- } \quad \mathit{current} \quad \times \quad \mathit{required}\\
&\qquad \mathit{AdvertisedContracts} \times \mathit{AdvertisedContracts}\\
&\qquad \to \mathit{ActiveContracts} \times \mathit{ActiveContracts}\\
&\qquad \to \mathit{List\ Deposit} \times \mathit{List\ Deposit}\\
&\qquad \to \mathit{Set}\ \textbf{where}
\end{aligned}
$$

-- *empty*

$$\varnothing : \mathit{Configuration}'\ ([\,]\,,[\,])\ ([\,]\,,[\,])\ ([\,]\,,[\,])$$

-- *contract advertisement*

$$
\begin{aligned}
&`\_ \ : \ (ad : \mathit{Advertisement}\ v\ vs^{c}\ vs^{v}\ vs^{p})\\
&\quad \to \mathit{Configuration}'\ ([v\,,vs^{c}\,,vs^{v}\,,vs^{p}\,,ad]\,,[\,])\ ([\,]\,,[\,])\ ([\,]\,,[\,])
\end{aligned}
$$

-- *active contract*

$$
\begin{aligned}
&\langle\ \_\,,\_\ \rangle^{c} \ : \ (c : \mathit{List}\ (\mathit{Contract}\ v\ vs)) \to \mathit{Value}\\
&\qquad \to \mathit{Configuration}'\ ([\,]\,,[\,])\ ([v\,,vs\,,c]\,,[\,])\ ([\,]\,,[\,])
\end{aligned}
$$

Small-step Semantics: Configurations II

-- deposit redeemable by a participant

$$\langle \_ , \_ \rangle^{\mathrm{d}} \;:\; (p : \mathit{Participant}) \to (v : \mathit{Value}) \to \mathit{Configuration}'\ ([\,] , [\,])\ ([\,] , [\,])\ ([p\ \mathit{has}\ v] , [\,])$$

-- authorization to perform an action

$$\_[\_] \;:\; (p : \mathit{Participant}) \to \mathit{Action}\ p\ \mathit{ads}\ \mathit{cs}\ \mathit{vs}\ \mathit{ds} \to \mathit{Configuration}'\ ([\,] , \mathit{ads})\ ([\,] , \mathit{cs})\ (\mathit{ds} , ((p\ \mathit{has}\ \_)\ \text{<\$>}\ \mathit{vs}))$$

-- committed secret

$$\langle\ \_ : \_ \# \_\ \rangle \;:\; \mathit{Participant} \to \mathit{Secret} \to \mathbb{N} \uplus \bot \to \mathit{Configuration}'\ ([\,] , [\,])\ ([\,] , [\,])\ ([\,] , [\,])$$

-- revealed secret

$$\_ : \_ \# \_ \;:\; \mathit{Participant} \to \mathit{Secret} \to \mathbb{N} \to \mathit{Configuration}'\ ([\,] , [\,])\ ([\,] , [\,])\ ([\,] , [\,])$$

Small-step Semantics: Configurations III

-- parallel composition

$$
\begin{aligned}
\_ \mid \_ \ :\ & \mathit{Configuration}' \ (\mathit{ads}^{\mathsf{l}} , \mathit{rads}^{\mathsf{l}}) \ (\mathit{cs}^{\mathsf{l}} , \mathit{rcs}^{\mathsf{l}}) \ (\mathit{ds}^{\mathsf{l}} , \mathit{rds}^{\mathsf{l}}) \\
\to\ & \mathit{Configuration}' \ (\mathit{ads}^{\mathsf{r}} , \mathit{rads}^{\mathsf{r}}) \ (\mathit{cs}^{\mathsf{r}} , \mathit{rcs}^{\mathsf{r}}) \ (\mathit{ds}^{\mathsf{r}} , \mathit{rds}^{\mathsf{r}}) \\
\to\ & \mathit{Configuration}' \ (\mathit{ads}^{\mathsf{l}} +\!\!+\ \mathit{ads}^{\mathsf{r}} \quad , \mathit{rads}^{\mathsf{l}} +\!\!+\ (\mathit{rads}^{\mathsf{r}} \setminus \mathit{ads}^{\mathsf{l}})) \\
& \qquad (\mathit{cs}^{\mathsf{l}} +\!\!+\ \mathit{cs}^{\mathsf{r}} \quad , \mathit{rcs}^{\mathsf{l}} +\!\!+\ (\mathit{rcs}^{\mathsf{r}} \setminus \mathit{cs}^{\mathsf{l}})) \\
& \qquad ((\mathit{ds}^{\mathsf{l}} \setminus \mathit{rds}^{\mathsf{r}}) +\!\!+\ \mathit{ds}^{\mathsf{r}} , \mathit{rds}^{\mathsf{l}} +\!\!+\ (\mathit{rds}^{\mathsf{r}} \setminus \mathit{ds}^{\mathsf{l}}))
\end{aligned}
$$

Small-step Semantics: Closed Configurations

$$\mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} = \mathit{Configuration}'\ (\mathit{ads}\ , [\,])\ (\mathit{cs}\ , [\,])\ (\mathit{ds}\ , [\,])$$

Small-step Semantics: Inference Rules I

$$\textbf{data}\ \_ \longrightarrow \_ \ :\ \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} \to \mathit{Configuration}\ \mathit{ads}'\ \mathit{cs}'\ \mathit{ds}' \to \mathit{Set}\ \textbf{where}$$

DEP-AuthJoin :

$$\langle\, A\,,\, v\,\rangle^{\mathrm{d}} \mid \langle\, A\,,\, v'\,\rangle^{\mathrm{d}} \mid \Gamma \longrightarrow \langle\, A\,,\, v\,\rangle^{\mathrm{d}} \mid \langle\, A\,,\, v'\,\rangle^{\mathrm{d}} \mid A\,[0 \leftrightarrow 1] \mid \Gamma$$

DEP-Join :

$$\langle\, A\,,\, v\,\rangle^{\mathrm{d}} \mid \langle\, A\,,\, v'\,\rangle^{\mathrm{d}} \mid A\,[0 \leftrightarrow 1] \mid \Gamma \longrightarrow \langle\, A\,,\, v + v'\,\rangle^{\mathrm{d}} \mid \Gamma$$

C-Advertise : $\forall\ \{\Gamma\ \mathit{ad}\}$

$$\to \exists[\ p \in \mathit{participants}^{\mathrm{g}}\ (G\ \mathit{ad})\ ]\ p \in \mathit{Hon}$$

$$\overline{\to \Gamma \longrightarrow {}`\mathit{ad} \mid \Gamma}$$

Small-step Semantics: Inference Rules II

$$\textit{C-AuthCommit} : \forall\ \{ A\ ad\ \Gamma \}$$
$$\rightarrow \textit{secrets}\ (G\ ad) \equiv a_1\ \ldots\ a_n$$
$$\rightarrow (A \in \textit{Hon} \rightarrow \forall\ [i \in 1\ \ldots\ n]\ a_i \not\equiv \bot)$$
$$\overline{\rightarrow\ `ad \mid \Gamma \longrightarrow `ad \mid \Gamma \mid \ldots \langle\ A : a_i \# N_i\ \rangle \ldots \mid A\ [\#\ ad]}$$

$$\textit{C-Control} : \forall\ \{ \Gamma\ C\ i\ D \}$$
$$\rightarrow C\ !!\ i \equiv A_1 : \ldots : A_n : D$$
$$\overline{\rightarrow \langle\ C\ ,\ v\ \rangle^{c} \mid \ldots\ A_i\ [C \triangleright^{b}\ i]\ \ldots \mid \Gamma \longrightarrow \langle\ D\ ,\ v\ \rangle^{c} \mid \Gamma}$$

$\vdots$

Small-step Semantics: Timed Inference Rules I

$$
\begin{array}{l}
\textbf{record}\ \mathit{Configuration}^{\mathrm{t}}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} : \mathit{Set}\ \textbf{where} \\
\quad \textbf{constructor}\ \_ @ \_ \\
\quad \textbf{field}\ \mathit{cfg} \ \ : \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} \\
\qquad\qquad \mathit{time} : \mathit{Time}
\end{array}
$$

$$
\begin{array}{l}
\textbf{data}\ \_ \longrightarrow_{\mathrm{t}} \_ \ : \ \mathit{Configuration}^{\mathrm{t}}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} \to \mathit{Configuration}^{\mathrm{t}}\ \mathit{ads}'\ \mathit{cs}'\ \mathit{ds}' \\
\qquad\qquad\quad \to \mathit{Set}\ \textbf{where} \\
\quad \mathit{Action} : \forall\ \{\Gamma\ \Gamma'\ t\} \\
\qquad \to \Gamma \longrightarrow \Gamma' \\
\qquad \quad \overline{} \\
\qquad \to \Gamma\ @\ t \longrightarrow_{\mathrm{t}} \Gamma'\ @\ t \\
\\
\quad \mathit{Delay} : \forall\ \{\Gamma\ t\ \delta\} \\
\qquad \quad \overline{\phantom{\Gamma\ @\ t \longrightarrow_{\mathrm{t}} \Gamma\ @\ (t + \delta)\ \ }} \\
\qquad \to \Gamma\ @\ t \longrightarrow_{\mathrm{t}} \Gamma\ @\ (t + \delta)
\end{array}
$$

Small-step Semantics: Timed Inference Rules II

$$
\begin{array}{l}
\mathit{Timeout} : \forall \; \{\Gamma \; \Gamma' \; t \; i \; \mathit{contract}\} \\
\qquad \text{-- all time constraints are satisfied} \\
\quad \to \mathit{All} \; (\_ \leqslant t) \; (\mathit{timeDecorations} \; (\mathit{contract} \; !! \; i)) \\
\qquad \text{-- resulting state if we pick this branch} \\
\quad \to \langle \, [\mathit{contract} \; !! \; i] \, , v \, \rangle^{c} \mid \Gamma \longrightarrow \Gamma' \\
\qquad \overline{} \\
\quad \to (\langle \, \mathit{contract} \, , v \, \rangle^{c} \mid \Gamma) \; @ \; t \longrightarrow_{t} \Gamma' \; @ \; t
\end{array}
$$

Small-step Semantics: Reordering I

```
_≈_ : Configuration ads cs ds → Configuration ads cs ds → Set
c ≈ c′ = cfgToList c ↭ cfgToList c′
  where
    open import Data.List.Permutation using (_↭_)
    cfgToList ∅                 = []
    cfgToList (l | r)           = cfgToList l ++ cfgToList r
    cfgToList {p₁} {p₂} {p₃} c = [p₁ , p₂ , p₃ , c]
```

Small-step Semantics: Reordering II

$$
\begin{aligned}
&\mathit{DEP\text{-}AuthJoin} : \\
&\qquad \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ (A\ \mathit{has}\ v :: A\ \mathit{has}\ v' :: \mathit{ds}) \ni \\
&\qquad\quad \Gamma' \approx \langle\, A\, ,\, v\, \rangle^{\mathrm{d}} \mid \langle\, A\, ,\, v'\, \rangle^{\mathrm{d}} \mid \Gamma \\
&\quad\to \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ (A\ \mathit{has}\ (v + v') :: \mathit{ds}) \ni \\
&\qquad\quad \Gamma'' \approx \langle\, A\, ,\, v\, \rangle^{\mathrm{d}} \mid \langle\, A\, ,\, v'\, \rangle^{\mathrm{d}} \mid A\ [0 \leftrightarrow 1] \mid \Gamma \\
&\qquad \rule{20em}{0.4pt} \\
&\quad\to \Gamma' \longrightarrow \Gamma''
\end{aligned}
$$

Small-step Semantics: Equational Reasoning

$$\mathbf{data}\ \_ \twoheadrightarrow \_ : \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} \to \mathit{Configuration}\ \mathit{ads}'\ \mathit{cs}'\ \mathit{ds}' \to \mathit{Set}$$

$$\_\Box : (M : \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds}) \to M \twoheadrightarrow M$$

$$\_ \to\langle\ \_\ \rangle\_ : \forall\ \{L'\ M\ M'\ N\}\ (L : \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds})$$

$$\to \{L \approx L' \times M \approx M'\}$$

$$\to L' \longrightarrow M'$$

$$\to M \twoheadrightarrow N$$

$$\overline{\to L \twoheadrightarrow N}$$

$$\mathit{begin}\ \_ : \forall\ \{M\ N\} \to M \twoheadrightarrow N \to M \twoheadrightarrow N$$

Small-step Semantics: Example I

$$
\begin{aligned}
&tc : \mathit{Advertisement}\ 1\ [\,]\ [\,]\ (1 :: 0 :: [\,])\\
&tc = \langle\ A\ !\ 1 \land A \ \#\ a \land B\ !\ 0\ \rangle\\
&\qquad\qquad \mathit{reveal}\ [a] \Rightarrow \mathit{withdraw}\ A \vdash \ldots\\
&\qquad\quad \oplus\ \mathit{after}\ t : \mathit{withdraw}\ B
\end{aligned}
$$

Small-step Semantics: Example II

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathit{tc\text{-}semantics} : \langle\, A \,,\, 1 \,\rangle^{\mathrm{d}} \twoheadrightarrow \langle\, A \,,\, 1 \,\rangle^{\mathrm{d}} \mid A : a \,\#\, 6} \\
\mathit{tc\text{-}semantics} = & \langle\, A \,,\, 1 \,\rangle^{\mathrm{d}} \\
\ \rightarrow\langle\, \mathit{C\text{-}Advertise} \,\rangle & `\mathit{tc} \mid \langle\, A \,,\, 1 \,\rangle^{\mathrm{d}} \\
\ \rightarrow\langle\, \mathit{C\text{-}AuthCommit} \,\rangle & `\mathit{tc} \mid \langle\, A \,,\, 1 \,\rangle^{\mathrm{d}} \mid \langle\, A : a \,\#\, 6 \,\rangle \mid A\, [\# \triangleright\ \mathit{tc}] \\
\ \rightarrow\langle\, \mathit{C\text{-}AuthInit} \,\rangle & `\mathit{tc} \mid \langle\, A \,,\, 1 \,\rangle^{\mathrm{d}} \mid \langle\, A : a \,\#\, 6 \,\rangle \mid A\, [\# \triangleright\ \mathit{tc}] \mid A\, [\mathit{tc} \triangleright^{s}\ 0] \\
\ \rightarrow\langle\, \mathit{C\text{-}Init} \,\rangle & \langle\, \mathit{tc} \,,\, 1 \,\rangle^{\mathrm{c}} \mid \langle\, A : a \,\#\, \mathit{inj}_1\ 6 \,\rangle \\
\ \rightarrow\langle\, \mathit{C\text{-}AuthRev} \,\rangle & \langle\, \mathit{tc} \,,\, 1 \,\rangle^{\mathrm{c}} \mid A : a \,\#\, 6 \\
\ \rightarrow\langle\, \mathit{C\text{-}Control} \,\rangle & \langle\, [\mathit{reveal}\ \ldots] \,,\, 1 \,\rangle^{\mathrm{c}} \mid A : a \,\#\, 6 \\
\ \rightarrow\langle\, \mathit{C\text{-}PutRev} \,\rangle & \langle\, [\mathit{withdraw}\ A] \,,\, 1 \,\rangle^{\mathrm{c}} \mid A : a \,\#\, 6 \\
\ \rightarrow\langle\, \mathit{C\text{-}Withdraw} \,\rangle & \langle\, A \,,\, 1 \,\rangle^{\mathrm{d}} \mid A : a \,\#\, 6
\end{array}
$$

□

Next Steps

Next Steps: UTxO

1. Multi-currency support
2. Integrate plutus-metatheory

Next Steps: BitML

1. Further formalization of the meta-theory
    - Traces, Strategies
2. Compilation correctness
    - Compile to abstract UTxO model instead of concrete Bitcoin transactions?

Next Steps: Others

1. Proof automation via domain-specific tactics
2. Featherweight Solidity
    - Provide proof-of-concept model in Agda
    - Perform some initial comparison

Discussion