Formalizing BitML Calculus in Agda

Towards formal verification for smart contracts

Orestis Melkonian

August 20, 2019

Universiteit Utrecht

INPUT | OUTPUT

Introduction

Motivation

- A lot of blockchain applications recently
- Sophisticated transactional schemes via smart contracts
- Reasoning about their execution is:
  1. *necessary*, significant funds are involved
  2. *difficult*, due to concurrency
- Hence the need for automatic tools that verify no bugs exist
  - This has to be done statically!

Background

Bitcoin

- Based on *unspent transaction outputs* (UTxO)
- Smart contracts in the simple language SCRIPT

Ethereum

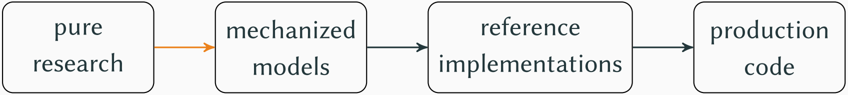
- Based on the notion of accounts
- Smart contracts in (almost) Turing-complete Solidity/EVM

Cardano (IOHK)

- UTxO-based, with several extensions
- Due to the extensions, smart contracts become more expressive

Methodology

- Keep things on an abstract level
  - Setup long-term foundations
- Fully mechanized approach, utilizing Agda's rich type system
- Fits well with IOHK's research-oriented approach



BitML

Basic Types

$$
\begin{aligned}
&\textbf{module } \mathit{BitML}\ (\mathit{Participant} : \mathit{Set}) \\
&\qquad\qquad (\_ \stackrel{?}{=}_{\mathrm{p}} \_ : \mathit{Decidable}\ \{A = \mathit{Participant}\}\ \_ \equiv \_) \\
&\qquad\qquad (\mathit{Honest} : \mathit{List}^{+}\ \mathit{Participant})\ \textbf{where}
\end{aligned}
$$

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

Contract Preconditions

```
data Precondition : Values -- volatile deposits
                  → Values -- 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 vs_v vs_p → Precondition vs_v′ vs_p′
      → Precondition (vs_v ++ vs_v′) (vs_p ++ vs_p′)
```

Contracts I

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

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

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

Contracts II

```
-- split the balance across different branches
split : ∀ {vs} → (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 : Values) : Set where
  constructor _⟨_⟩⊣_
  field G     : Precondition vsᵛ vsᵖ
        C     : Contracts v vsᶜ
        valid : length vsᶜ ≤ length vsᵛ
              × participantsᵍ G ++ participantsᶜ C
                  ⊆
                participant ⟨$⟩ persistentDeposits G
              × v ≡ sum vsᵖ
```

Example Advertisement

$$\textbf{open}\ \mathit{BitML}\ (A \mid B)\ \ldots\ [A]^{+}$$

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

$$
\begin{aligned}
&\mathit{AdvertisedContracts} = \mathit{List}\ (\exists[\, v, \ldots, vs^{\mathrm{p}}\,]\ \mathit{Advertisement}\ v\ \ \ldots\ \ vs^{\mathrm{p}})\\
&\mathit{ActiveContracts} \qquad = \mathit{List}\ (\exists[\, v, vs\,]\ \mathit{Contracts}\ v\ vs)
\end{aligned}
$$

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

Small-step Semantics: Actions II

```
-- join two deposits
_↔_ : ∀ {vs} → (i : Index vs) → (j : Index vs)
    → Action p [] [] vs (p has_ ⟨$⟩ merge i j vs)
-- 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 : Contracts v vs) → (i : Index c)
    → Action p [] [ v , vs , c ] [] []
⋮
```

Small-step Semantics: Actions Example

-- A spends the required ₿ 2, as stated in the pre-condition

$\textit{ex-spend} : \textit{Action}\ A\ [\,5\,,\,[\,200\,]\,,\,[\,200\,]\,,\,[\,3\,,\,2\,]\,,\,\textit{ex-ad}\,]\ [\,]\ [\,2\,]\ [\,]$

$\textit{ex-spend} = \textit{ex-ad} \triangleright^{s}\ 1$

Small-step Semantics: Configurations I

```
data Configuration′ : -- current × required
                      AdvertisedContracts × AdvertisedContracts
                    → ActiveContracts     × ActiveContracts
                    → Deposits            × Deposits
                    → Set where

  -- empty
  ∅ : Configuration′ ([] , []) ([] , []) ([] , [])

  -- contract advertisement
  `_ : (ad : Advertisement v vsᶜ vsᵛ vsᵖ)
    → Configuration′ ([ v , vsᶜ , vsᵛ , vsᵖ , ad ] , []) ([] , []) ([] , [])

  -- active contract
  ⟨_,_⟩ᶜ : (c : Contracts v vs) → Value
          → Configuration′ ([] , []) ([ v , vs , c ] , []) ([] , [])
```

Small-step Semantics: Configurations II

```
-- deposit redeemable by a participant
⟨ _ , _ ⟩ᵈ      : (p : Participant) → (v : Value)
                → Configuration′ ([ ] , [ ]) ([ ] , [ ]) ([ p has v ] , [ ])

-- authorization to perform an action
_ [ _ ]  : (p : Participant) → Action p ads cs vs ds
         → Configuration′ ([ ] , ads) ([ ] , cs) (ds , ((p has _) ⟨$⟩ vs))

-- committed secret
⟨ _ : _ # _ ⟩  : Participant → Secret → ℕ ⊎ ⊥
                → Configuration′ ([ ] , [ ]) ([ ] , [ ]) ([ ] , [ ])

-- revealed secret
_ : _ # _  : Participant → Secret → ℕ
           → Configuration′ ([ ] , [ ]) ([ ] , [ ]) ([ ] , [ ])
```

Small-step Semantics: Configurations III

-- parallel composition

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

Small-step Semantics: Closed Configurations

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

Small-step Semantics: Inference Rules I

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

$\mathit{DEP\text{-}AuthJoin}$:

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

$\mathit{DEP\text{-}Join}$:

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

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

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

$$\overline{\to \Gamma \longrightarrow {}^{\backprime}\mathit{ad} \mid \Gamma}$$

Small-step Semantics: Inference Rules II

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

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

⋮

Small-step Semantics: Timed Inference Rules I

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

$$
\begin{array}{l}
\textbf{data}\ \_ \longrightarrow_{\mathrm{t}} \_\ :\ \mathit{Configuration}^{\mathrm{t}}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} \to \mathit{Configuration}^{\mathrm{t}}\ \mathit{ads}'\ \mathit{cs}'\ \mathit{ds}' \\
\qquad\qquad\qquad \to \mathit{Set}\ \textbf{where}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Action} : \forall\ \{\Gamma\ \Gamma'\ t\} \\
\quad \to \Gamma \longrightarrow \Gamma' \\
\qquad \overline{\phantom{\Gamma\ @\ t \longrightarrow_{\mathrm{t}} \Gamma'\ @\ t}} \\
\quad \to \Gamma\ @\ t \longrightarrow_{\mathrm{t}} \Gamma'\ @\ t
\end{array}
$$

$$
\begin{array}{l}
\mathit{Delay} : \forall\ \{\Gamma\ t\ \delta\} \\
\qquad \overline{\phantom{\Gamma\ @\ t \longrightarrow_{\mathrm{t}} \Gamma\ @\ (t + \delta)}} \\
\quad \to \Gamma\ @\ t \longrightarrow_{\mathrm{t}} \Gamma\ @\ (t + \delta)
\end{array}
$$

Small-step Semantics: Timed Inference Rules II

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

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

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

Small-step Semantics: Equational Reasoning

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

$$\_\,\square : (M : \mathit{Configuration}\ ads\ cs\ ds) \to M \longrightarrow^* M$$

$$\_\longrightarrow\langle\ \_\ \rangle\_ : \forall\ \{L'\ M\ M'\ N\}\ (L : \mathit{Configuration}\ ads\ cs\ ds)$$
$$\to \{L \approx L' \times M \approx M'\}$$
$$\to L' \longrightarrow M'$$
$$\to M \longrightarrow^* N$$
$$\overline{\hspace{12em}}$$
$$\to L \longrightarrow^* N$$

$$\mathit{begin}\ \_ : \forall\ \{M\ N\} \to M \longrightarrow^* N \to M \longrightarrow^* N$$

Timed-commitment Protocol

A promises to reveal a secret, otherwise loses deposit.

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

Small-step Semantics: Example (derivation)

$$
\begin{array}{ll}
\textit{tc-semantics} : \langle A , 1 \rangle^{d} \longrightarrow^{*} \langle A , 1 \rangle^{d} \mid A : a \# 6 & \\
\textit{tc-semantics} = & \langle A , 1 \rangle^{d} \\
\longrightarrow \langle \textit{C-Advertise} \rangle & `\textit{tc} \mid \langle A , 1 \rangle^{d} \\
\longrightarrow \langle \textit{C-AuthCommit} \rangle & `\textit{tc} \mid \langle A , 1 \rangle^{d} \mid \langle A : a \# 6 \rangle \mid A [\# \triangleright \textit{tc}] \\
\longrightarrow \langle \textit{C-AuthInit} \rangle & `\textit{tc} \mid \langle A , 1 \rangle^{d} \mid \langle A : a \# 6 \rangle \mid A [\# \triangleright \textit{tc}] \mid A [\textit{tc} \triangleright^{s} 0 \\
\longrightarrow \langle \textit{C-Init} \rangle & \langle \textit{tc} , 1 \rangle^{c} \mid \langle A : a \# \textit{inj}_1 \; 6 \rangle \\
\longrightarrow \langle \textit{C-AuthRev} \rangle & \langle \textit{tc} , 1 \rangle^{c} \mid A : a \# 6 \\
\longrightarrow \langle \textit{C-Control} \rangle & \langle [\textit{reveal} \; \ldots] , 1 \rangle^{c} \mid A : a \# 6 \\
\longrightarrow \langle \textit{C-PutRev} \rangle & \langle [\textit{withdraw} \; A] , 1 \rangle^{c} \mid A : a \# 6 \\
\longrightarrow \langle \textit{C-Withdraw} \rangle & \langle A , 1 \rangle^{d} \mid A : a \# 6
\end{array}
$$

□

Symbolic Model: Labelled step relation

$$
\begin{aligned}
&\textbf{data}\ \_ \longrightarrow⟦\ \_\ ⟧\ \_\ :\ \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds}\\
&\qquad\qquad\qquad\qquad \to \mathit{Label}\\
&\qquad\qquad\qquad\qquad \to \mathit{Configuration}\ \mathit{ads}'\ \mathit{cs}'\ \mathit{ds}'\\
&\qquad\qquad\qquad\qquad \to \mathit{Set}\ \textbf{where}
\end{aligned}
$$

$\vdots$

$$
\begin{aligned}
&\mathit{DEP\text{-}AuthJoin} :\\
&\quad \langle\, A\,,\, v\,\rangle^{\mathrm{d}} \mid \langle\, A\,,\, v'\,\rangle^{\mathrm{d}} \mid \Gamma\\
&\longrightarrow⟦\ \mathit{auth\text{-}join}\ [\,A\,,\, 0 \leftrightarrow 1\,]\ ⟧\\
&\quad \langle\, A\,,\, v\,\rangle^{\mathrm{d}} \mid \langle\, A\,,\, v'\,\rangle^{\mathrm{d}} \mid A\ [\,0 \leftrightarrow 1\,] \mid \Gamma
\end{aligned}
$$

$\vdots$

$$
\begin{array}{l}
\textbf{data}\ \mathit{Trace} : \mathit{Set}\ \textbf{where} \\
\quad \_\bullet \qquad\qquad\quad : \exists\mathit{TimedConfiguration} \rightarrow \mathit{Trace} \\
\quad \_ :: [\![\ \_\ ]\!]\ \_ : \exists\mathit{TimedConfiguration} \rightarrow \mathit{Label} \rightarrow \mathit{Trace} \rightarrow \mathit{Trace}
\end{array}
$$

$$
\begin{array}{l}
\_ \rightarrowtail [\![\ \_\ ]\!]\ \_ : \mathit{Trace} \rightarrow \mathit{Label} \rightarrow \exists\mathit{TimedConfiguration} \rightarrow \mathit{Set} \\
R \rightarrowtail [\![\ \alpha\ ]\!]\ (\_ , \_ , \_ , \mathit{tc}') \\
\quad = \mathit{proj}_2\ (\mathit{proj}_2\ (\mathit{proj}_2\ (\mathit{lastCfg}\ R))) \longrightarrow [\![\ \alpha\ ]\!]\ \mathit{tc}'
\end{array}
$$

Symbolic Model: Strategies (honest participant)

$$
\begin{array}{l}
\textbf{record}\ \mathit{HonestStrategy}\ (A : \mathit{Participant}) : \mathit{Set}\ \textbf{where} \\
\quad \textbf{field} \\
\qquad \mathit{strategy} : \mathit{Trace} \longrightarrow \mathit{List}\ \mathit{Label} \\
\qquad \mathit{valid} \quad : A \in \mathit{Hon} \\
\qquad\qquad\quad \times\ (\forall\ R\ \alpha \longrightarrow \alpha \in \mathit{strategy}\ R\, * \longrightarrow \\
\qquad\qquad\qquad \exists[\,R'\,]\ (R \rightarrowtail [\![\ \alpha\ ]\!]\ R')) \\
\qquad\qquad\quad \times\ (\forall\ R\ \alpha \longrightarrow \alpha \in \mathit{strategy}\ R\, * \longrightarrow \\
\qquad\qquad\qquad \mathit{All}\ (\_ \equiv A)\ (\mathit{authDecoration}\ \alpha)) \\
\qquad\qquad\quad \vdots
\end{array}
$$

$$
\mathit{HonestStrategies} = \forall\ \{A\} \longrightarrow A \in \mathit{Hon} \longrightarrow \mathit{HonestStrategy}\ A
$$

$$
\begin{array}{l}
\textbf{record } \mathit{AdversarialStrategy}\ (\mathit{Adv} : \mathit{Participant}) : \mathit{Set}\ \textbf{where} \\
\quad \textbf{field} \\
\qquad \mathit{strategy} : \mathit{Trace} \longrightarrow \mathit{List}\ (\mathit{Participant} \times \mathit{List}\ \mathit{Label}) \longrightarrow \mathit{Label} \\
\qquad \mathit{valid} \quad : \mathit{Adv} \notin \mathit{Hon} \\
\qquad\qquad \times \forall \{ R : \mathit{Trace} \} \{ \mathit{moves} : \mathit{List}\ (\mathit{Participant} \times \mathit{List}\ \mathit{Label}) \} \longrightarrow \\
\qquad\qquad\quad \textbf{let } \alpha = \mathit{strategy}\ R \ast\ \mathit{moves}\ \textbf{in} \\
\qquad\qquad\quad (\quad \exists [\, A \,]\ \ (\ A \in \mathit{Hon} \\
\qquad\qquad\qquad\qquad \times\ \mathit{authDecoration}\ \alpha \equiv \mathit{just}\ A \\
\qquad\qquad\qquad\qquad \times\ \alpha \in \mathit{concatMap}\ \mathit{proj}_2\ \mathit{moves}\ ) \\
\qquad\qquad\quad\ \uplus\ (\quad \mathit{authDecoration}\ \alpha \equiv \mathit{nothing} \\
\qquad\qquad\qquad \times\ (\forall\ \delta \longrightarrow \alpha \not\equiv \mathit{delay}\ [\, \delta \,]) \\
\qquad\qquad\qquad \times\ \exists [\, R' \,]\ (R \rightarrowtail [\![\ \alpha\ ]\!]\ R')) \\
\qquad\qquad\quad \vdots \\
\qquad\qquad\quad)
\end{array}
$$

Symbolic Model: Adversary makes final choice

$$
\begin{aligned}
&\mathit{runAdversary} : \mathit{Strategies} \to \mathit{Trace} \to \mathit{Label}\\
&\mathit{runAdversary}\ (S\dagger\ ,\ S)\ R = \mathit{strategy}\ S\dagger\ R * \ \mathit{honestMoves}\\
&\quad \mathbf{where}\\
&\qquad \mathit{honestMoves} = \mathit{mapWith}{\in}\ \mathit{Hon}\ (\lambda\ \{A\}\ p \to A\ ,\ \mathit{strategy}\ (S\ p)\ R*)
\end{aligned}
$$

Symbolic Model: Conformance

$$\textbf{data}\ \_\text{-conforms-to-}\_ : \mathit{Trace} \to \mathit{Strategies} \to \mathit{Set}\ \textbf{where}$$

$$\begin{aligned}
&\mathit{base} : \forall\ \{\Gamma : \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds}\}\ \{\mathit{SS} : \mathit{Strategies}\} \\
&\quad \to \mathit{Initial}\ \Gamma \\
&\quad \overline{\hphantom{\to (\mathit{ads}, \mathit{cs}, \mathit{ds}, \Gamma\ @\ 0)^{\bullet}\ \text{-conforms-to-}\ \mathit{SS}}} \\
&\quad \to (\mathit{ads}\ ,\ \mathit{cs}\ ,\ \mathit{ds}\ ,\ \Gamma\ @\ 0)^{\bullet}\ \text{-conforms-to-}\ \mathit{SS}
\end{aligned}$$

$$\begin{aligned}
&\mathit{step} : \forall\ \{R : \mathit{Trace}\}\ \{T' : \exists \mathit{TimedConfiguration}\}\ \{\mathit{SS} : \mathit{Strategies}\} \\
&\quad \to R\ \text{-conforms-to-}\ \mathit{SS} \\
&\quad \to R \rightarrowtail\!\!\!\to [\![\ \mathit{runAdversary}\ \mathit{SS}\ R\ ]\!]\ T' \\
&\quad \overline{\hphantom{\to (T' :: [\![\ \mathit{runAdversary}\ \mathit{SS}\ R\ ]\!]\ R)\ \text{-conforms-to-}\ \mathit{SS}}} \\
&\quad \to (T' :: [\![\ \mathit{runAdversary}\ \mathit{SS}\ R\ ]\!]\ R)\ \text{-conforms-to-}\ \mathit{SS}
\end{aligned}$$

Symbolic Model: Meta-theory

- *strip-preserves-semantics* :

$$
\begin{aligned}
&(\forall\ A\ s \quad\ \ \rightarrow \alpha \not\equiv \mathit{auth\text{-}rev}\ [\ A\ ,\ s\ ]) \rightarrow \\
&(\forall\ A\ ad\ \Delta \rightarrow \alpha \not\equiv \mathit{auth\text{-}commit}\ [\ A\ ,\ ad\ ,\ \Delta\ ]) \\
&\rightarrow (\forall\ T' \quad \rightarrow \frac{R \rightarrowtail [\![\ \alpha\ ]\!]\ T'}{} \\
&\qquad\qquad\qquad \rightarrow R{*} \rightarrowtail [\![\ \alpha\ ]\!]\ T'{*}) \\
&\times\ \ (\forall\ T' \quad \rightarrow \frac{R{*} \rightarrowtail [\![\ \alpha\ ]\!]\ T'}{} \\
&\qquad\qquad\qquad \rightarrow \exists[\ T''\ ]\ (R \rightarrowtail [\![\ \alpha\ ]\!]\ T'') \times (T'{*} \equiv T''{*})
\end{aligned}
$$

- *adversarial-move-is-semantic* :

$$\exists[\ T'\ ]\ (R \rightarrowtail [\![\ \mathit{runAdversary}\ (S\dagger\ ,\ S)\ \ R\ ]\!]\ \ T')$$

BitML Paper Fixes

Discrepancies in inference rules

e.g. forgetting surrounding context Γ

Non-linear derivations

If one of the hypothesis is another step, we lose equational-style linearity.

Solution: Move result state of the hypothesis to the result of the rule.

Missed assumptions

The original formulation of the *strip-preserves-semantics* lemma required only that the action does not reveal secrets (*C-AuthRev*), but it should not commit secrets either (*C-AuthCommit*).

Future Work

Next Steps: BitML

1. A lot of proof obligations associated with most datatypes
   - Implement decision procedures for them, just like we did for UTxO
2. Computational model
   - Formulation very similar to the symbolic model we already have, but a lot of additional details to handle
3. Compilation correctness: *Symbolic Model* $\approx$ *Computational Model*
   - Compile to abstract UTxO model instead of concrete Bitcoin transactions?
   - Already successfully employed by Marlowe
   - Data scripts stateful capabilities fit well for state transition systems!

Conclusion

Conclusion

- Formal methods are a promising direction for blockchain
  - Especially language-oriented, type-driven approaches
- Although formalization is tedious and time-consuming
  - Strong results and deep understanding of models
  - Certified compilation is here to stay! (c.f. *CompCert, seL4*)
- However, tooling is badly needed....
  - We need better, more sophisticated programming technology for dependently-typed languages

Questions?