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

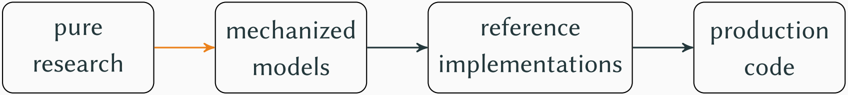
- Based on the notion of accounts
- Smart contracts in (almost) Turing-complete Solidity/EVM

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

$$
\begin{aligned}
&\textbf{module } \mathit{BitML}\ (\mathit{Participant} : \mathit{Set})\\
&\qquad\qquad (\_\overset{?}{=}_{\mathrm{p}}\_ : \mathit{Decidable}\ \{A = \mathit{Participant}\}\ \_\equiv\_)\\
&\qquad\qquad (\mathit{Honest} : \mathit{List}^{+}\ \mathit{Participant})\ \textbf{where}\\
&\mathit{Time} = \mathbb{N}\\
&\mathit{Value} = \mathbb{N}\\
&\mathit{Secret} = \mathit{String}\\
&\mathit{Deposit} = \mathit{Participant} \times \mathit{Value}
\end{aligned}
$$

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 vsᵥ vsₚ → Precondition vsᵥ′ vsₚ′
      → Precondition (vsᵥ ++ vsᵥ′) (vsₚ ++ vsₚ′)
```

Contracts I

```
data Contract : Value  -- the monetary value it carries
              → Values -- the volatile deposits it presumes
              → Set where

  -- collect deposits and secrets
  put _ reveal _ if _ ⇒ _ ⊣ _ :
      (vs : Values) → (s : Secrets) → Predicate s′
    → Contract (v + sum vs) vs′ → s′ ⊆ s
    → Contract v (vs′ ++ vs)

  -- transfer the remaining balance to a participant
  withdraw : ∀ {v vs} → Participant → Contract v vs
```

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
```

$$
\begin{array}{l}
\textbf{record}\ \mathit{Advertisement}\ (v : \mathit{Value})\ (vs^{c}\ vs^{v}\ vs^{p} : \mathit{Values}) : \mathit{Set}\ \textbf{where} \\
\quad \textbf{constructor}\ \_\langle\ \_\ \rangle \dashv \_ \\
\quad \textbf{field}\ G \quad : \mathit{Precondition}\ vs^{v}\ vs^{p} \\
\qquad\quad C \quad : \mathit{Contracts}\ v\ vs^{c} \\
\qquad\quad \mathit{valid} : \mathit{length}\ vs^{c} \leqslant \mathit{length}\ vs^{v} \\
\qquad\qquad\quad \times\ \mathit{participants}^{g}\ G +\!\!\!+\ \mathit{participants}^{c}\ C \\
\qquad\qquad\qquad\quad \subseteq \\
\qquad\qquad\qquad \mathit{participant}\ \langle\$\rangle\ \mathit{persistentDeposits}\ G \\
\qquad\qquad\quad \times\ v \equiv \mathit{sum}\ vs^{p}
\end{array}
$$

Example Advertisement

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

$$\mathit{ex\text{-}ad} : \mathit{Advertisement}\ 5\ [200]\ [200]\ [3\ ,\ 2]$$

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

Small-step Semantics: Actions I

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

```
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
ex-spend : Action A [5 , [200] , [200] , [3 , 2] , ex-ad] [] [2] []
ex-spend = ex-ad ▷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

$$
\begin{array}{l}
\text{-- deposit redeemable by a participant} \\
\langle\, \_\,,\, \_\, \rangle^{\mathrm{d}} \quad : \ (p : \mathit{Participant}) \rightarrow (v : \mathit{Value}) \\
\qquad\qquad \rightarrow \mathit{Configuration}'\ ([\,]\,,[\,])\ ([\,]\,,[\,])\ ([p\ \mathit{has}\ v]\,,[\,]) \\
\text{-- authorization to perform an action} \\
\_\,[\_] \quad : \ (p : \mathit{Participant}) \rightarrow \mathit{Action}\ p\ \mathit{ads}\ \mathit{cs}\ \mathit{vs}\ \mathit{ds} \\
\qquad \rightarrow \mathit{Configuration}'\ ([\,]\,,\mathit{ads})\ ([\,]\,,\mathit{cs})\ (\mathit{ds}\,,((p\ \mathit{has}\ \_)\ \langle\$\rangle\ \mathit{vs})) \\
\text{-- committed secret} \\
\langle\, \_ : \_\#\_\, \rangle \ : \ \mathit{Participant} \rightarrow \mathit{Secret} \rightarrow \mathbb{N} \uplus \bot \\
\qquad\qquad \rightarrow \mathit{Configuration}'\ ([\,]\,,[\,])\ ([\,]\,,[\,])\ ([\,]\,,[\,]) \\
\text{-- revealed secret} \\
\_ : \_\#\_ \ : \ \mathit{Participant} \rightarrow \mathit{Secret} \rightarrow \mathbb{N} \\
\qquad\qquad \rightarrow \mathit{Configuration}'\ ([\,]\,,[\,])\ ([\,]\,,[\,])\ ([\,]\,,[\,])
\end{array}
$$

Small-step Semantics: Configurations III

-- parallel composition

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

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 : $\forall\ \{\Gamma\ ad\}$

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

Small-step Semantics: Inference Rules II

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

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

$\vdots$

Small-step Semantics: Timed Inference Rules I

```
record Configurationᵗ ads cs ds : Set where
  constructor _@_
  field cfg  : Configuration ads cs ds
        time : Time

data _—→ₜ_ : Configurationᵗ ads cs ds → Configurationᵗ ads′ cs′ ds′
           → Set where
  Action : ∀ {Γ Γ′ t}
    → Γ —→ Γ′
      ───────────────────
    → Γ @ t —→ₜ Γ′ @ t

  Delay : ∀ {Γ t δ}
      ─────────────────────────
    → Γ @ t —→ₜ Γ @ (t + δ)
```

Small-step Semantics: Timed Inference Rules II

```
Timeout : ∀ {Γ Γ′ t i contract}
      -- all time constraints are satisfied
  → All (_ ≤ t) (timeDecorations (contract !! i))
      -- resulting state if we pick this branch
  → ⟨ [contract !! i] , v ⟩ᶜ | Γ ⟶ Γ′
    ─────────────────────────────────────────
  → (⟨ contract , v ⟩ᶜ | Γ) @ t ⟶ₜ Γ′ @ t
```

Small-step Semantics: Reordering I

$$
\begin{array}{l}
\_ \approx \_ : \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} \to \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} \to \mathit{Set} \\
c \approx c' = \mathit{cfgToList}\ c \leftrightsquigarrow \mathit{cfgToList}\ c' \\
\quad \textbf{where} \\
\qquad \textbf{open import}\ \mathit{Data.List.Permutation}\ \textbf{using}\ (\_ \leftrightsquigarrow \_) \\
\qquad \mathit{cfgToList}\ \emptyset \hspace{9em} = [\,] \\
\qquad \mathit{cfgToList}\ (l \mid r) \hspace{7em} = \mathit{cfgToList}\ l +\!\!+\ \mathit{cfgToList}\ r \\
\qquad \mathit{cfgToList}\ \{p_1\}\ \{p_2\}\ \{p_3\}\ c = [p_1\ ,\ p_2\ ,\ p_3\ ,\ c]
\end{array}
$$

Small-step Semantics: Reordering II

DEP-AuthJoin :

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

Small-step Semantics: Equational Reasoning

$$
\begin{array}{l}
\textbf{data}\ \_\longrightarrow^*\_\ :\ \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} \to \mathit{Configuration}\ \mathit{ads}'\ \mathit{cs}'\ \mathit{ds}' \\
\qquad\qquad\qquad \to \mathit{Set}\ \textbf{where} \\
\quad \_\,\square : (M : \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds}) \to M \longrightarrow^* M \\
\quad \_\longrightarrow\langle\_\rangle\_ : \forall\, \{L'\ M\ M'\ N\}\ (L : \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds}) \\
\qquad \to \{L \approx L' \times M \approx M'\} \\
\qquad \to L' \longrightarrow M' \\
\qquad \to M \longrightarrow^* N \\
\qquad\quad \overline{\hspace{6cm}} \\
\qquad \to L \longrightarrow^* N \\
\mathit{begin}\ \_ : \forall\, \{M\ N\} \to M \longrightarrow^* N \to M \longrightarrow^* N
\end{array}
$$

Small-step Semantics: Example (contract)

Timed-commitment Protocol

A promises to reveal a secret, otherwise loses deposit.

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

Small-step Semantics: Example (derivation)

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

$\square$

$$
\begin{aligned}
\textbf{data}\ \_ \longrightarrow[\![\ \_\ ]\!]\ \_\ :\ & \mathit{Configuration}\ \mathit{ads}\ \mathit{cs}\ \mathit{ds} \\
\rightarrow\ & \mathit{Label} \\
\rightarrow\ & \mathit{Configuration}\ \mathit{ads}'\ \mathit{cs}'\ \mathit{ds}' \\
\rightarrow\ & \mathit{Set}\ \textbf{where}
\end{aligned}
$$

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

```
data Trace : Set where
  _∙           : ∃TimedConfiguration → Trace
  _ :: ⟦ _ ⟧ _ : ∃TimedConfiguration → Label → Trace → Trace

_ ↣⟦ _ ⟧ _ : Trace → Label → ∃TimedConfiguration → Set
R ↣⟦ α ⟧ (_ , _ , _ , tc′)
  = proj₂ (proj₂ (proj₂ (lastCfg R))) —→⟦ α ⟧ tc′
```

Symbolic Model: Strategies (honest participant)

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

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

Symbolic Model: Strategies (adversary)

```
record AdversarialStrategy (Adv : Participant) : Set where
  field
    strategy : Trace → List (Participant × List Label) → Label

    valid    : Adv ∉ Hon
             × ∀ {R : Trace} {moves : List (Participant × List Label)} →
                 let α = strategy R∗ moves in
                 (   ∃[A]  ( A ∈ Hon
                           × authDecoration α ≡ just A
                           × α ∈ concatMap proj₂ moves )
                  ⊎ (  authDecoration α ≡ nothing
                     × (∀ δ → α ≢ delay [δ])
                     × ∃[R′] (R ↣⟦ α ⟧ R′))
                 ⋮
                 )
```

Symbolic Model: Adversary makes final choice

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

Symbolic Model: Conformance

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

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

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

Symbolic Model: Meta-theory

- *strip-preserves-semantics* :

$$
\begin{array}{ll}
(\forall\ A\ s & \rightarrow \alpha \not\equiv \mathit{auth\text{-}rev}\,[A\,,\,s]) \rightarrow \\
(\forall\ A\ ad\ \Delta & \rightarrow \alpha \not\equiv \mathit{auth\text{-}commit}\,[A\,,\,ad\,,\,\Delta]) \\
\ \rightarrow (\forall\ T' & \rightarrow \dfrac{R \rightarrowtail\llbracket\ \alpha\ \rrbracket\ T'}{} \\
 & \rightarrow R* \rightarrowtail\llbracket\ \alpha\ \rrbracket\ T'*) \\
\ \times\ (\forall\ T' & \rightarrow \dfrac{R* \rightarrowtail\llbracket\ \alpha\ \rrbracket\ T'}{} \\
 & \rightarrow \exists[\,T''\,]\ (R \rightarrowtail\llbracket\ \alpha\ \rrbracket\ T'') \times (T'* \equiv T''*)
\end{array}
$$

- *adversarial-move-is-semantic* :

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

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?