Formalization of the BitML Calculus in Agda

ORESTIS MELKONIAN, Utrecht University, The Netherlands

Email: melkon.or@gmail.com
Research Advisors: Wouter Swierstra (UU) & Manuel Chakravarty (IOHK)
ACM Student Number: 4094241
Category: Graduate (MSc Student)

1 INTRODUCTION

... Bitcoin → smart contracts → bus → formal methods [?] ...
... already some static analysis tools citemooly,madmax,liquidity ...
... advocate language-based, type-driven approach [?] ...
... especially proof assistants based on dependent types [8] ... extrinsic vs intrinsic ...

... BitML: idealistic process calculus for Bitcoin smart contracts [2] ...
... provide the first formalization of the BitML calculus in Agda ...
... set the foundation to later accommodate a full compilation correctness proof ... full abstraction result ...

2 THE BITML CALCULUS

All code is publicly available on Github[1].

2.1 Inherently-typed Contracts

Moving on to actual contracts, we define them by means of a collection of five types of commands; *put* injects participant deposits and revealed secrets in the remaining contract, *withdraw* transfers the current funds to a participant, *split* distributes the current funds across different individual contracts, _ : _ requires the authorization from a participant to proceed and *after* _ : _ allows further execution of the contract only after some time has passed.

```
data Contract : Value  -- the monetary value it carries
              → Values -- the deposits it presumes
              → Set where
  put _ reveal _ if _ ⇒ _ ⊢ _ :
    (vs : List Value) → (s : Secrets) → Predicate s′ → Contract (v + sum vs) vs′ → s′ ⊆ s
     → Contract v (vs′ ++ vs)
  withdraw : ∀{ v } → Participant → Contract v [ ]
  split : (cs : List (∃[ v ] ∃[ vs ] Contract v vs))
     → Contract (sum (proj₁ <$> cs)) (concat (proj₂ <$> cs))
  _ : _ : Participant → Contract v vs → Contract v vs
  after _ : _ : Time → Contract v vs → Contract v vs
```

There is a lot of type-level manipulation across all constructors, since we need to make sure that indices are calculated properly. For instance, the total value in a contract constructed by the

[1] https://github.com/omelkonian/formal-bitml

Author's address: Orestis Melkonian, Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands.

split command is the sum of the values carried by each branch. The *put* command[2] additionally requires an explicit proof that the predicate of the if part only uses secrets revealed by the same command.

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

2.2 Small-step Semantics

... indexed configuration (+ actions) ...

[2] *put* comprises of several components and we will omit those that do not contain any helpful information, e.g. write *put* _ ⇒ _ when there are no revealed secrets and the predicate trivially holds.

```agda
data Configuration′ : -- current × required
                      AdvertisedContracts × AdvertisedContracts
                    → ActiveContracts     × ActiveContracts
                    → List Deposit        × List Deposit
                    → Set where

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

  -- contract advertisement
  `_ : (ad : Advertisement v vs^c vs^g)
     → Configuration′ ([ v , vs^c , vs^g , ad ] , []) ([] , []) ([] , [])

  -- active contract
  ⟨_,_⟩^c : (c : List (Contract v vs)) → Value
          → Configuration′ ([] , []) ([ v , vs , c ] , []) ([] , [])

  -- 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′ ([] , []) ([] , []) ([] , [])

  -- parallel composition
  _|_ : Configuration′ (ads^l , rads^l) (cs^l , rcs^l) (ds^l , rds^l)
      → Configuration′ (ads^r , rads^r) (cs^r , rcs^r) (ds^r , rds^r)
      → Configuration′ (ads^l               ++ ads^r , rads^l ++ (rads^r \ ads^l))
                       (cs^l                ++ cs^r  , rcs^l  ++ (rcs^r  \ cs^l))
                       ((ds^l \ rds^r)      ++ ds^r  , rds^l  ++ (rds^r  \ ds^l))

Configuration : AdvertisedContracts → ActiveContracts → List Deposit → Set
Configuration ads cs ds = Configuration′ (ads , []) (cs , []) (ds , [])
```

... inference rules ...

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

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

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

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

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

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

$$\rightarrow \frac{\exists[\,p \in \mathit{participants}^{g}\ (G\ ad)\,]\ p \in \mathit{Hon}}{\rightarrow \Gamma \longrightarrow {}^{\backprime}ad \mid \Gamma}$$

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

$$\rightarrow \mathit{secrets}\ (G\ ad) \equiv a_0\ \ldots\ a_n$$

$$\rightarrow \frac{(A \in \mathit{Hon} \rightarrow \forall[\,i \in 0\ \ldots\ n\,]\ a_i \not\equiv \bot)}{\rightarrow {}^{\backprime}ad \mid \Gamma \longrightarrow {}^{\backprime}ad \mid \Gamma \mid \ldots \langle A : a_i \sharp N_i \rangle \ldots \mid A\,[\sharp\triangleright\ ad]}$$

$\mathit{C\text{-}Control} : \forall\{\Gamma\ C\ i\ D\}$

$$\rightarrow \frac{C\ !!\ i \equiv A_1 : A_2 : \ldots : A : D}{\rightarrow \langle C , v \rangle^{c} \mid \ldots\ A_i\,[C \triangleright^{b}\ i]\ \ldots \mid \Gamma \longrightarrow \langle D , v \rangle^{c} \mid \Gamma}$$

$\vdots$

... mention timed-configurations at the upper level $\_\longrightarrow_{t}\_$ ...
... implicit re-ordering in $\_\twoheadrightarrow\_$ ...

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

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

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

$$\rightarrow \frac{L \longrightarrow M \rightarrow M \twoheadrightarrow N}{\rightarrow L \twoheadrightarrow N}$$

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

Symbolic model.

- honest strategies
- adversary strategy
- conformance

3 EXAMPLE: TIMED-COMMITMENT PROTOCOL

$$
\begin{aligned}
&tc : Advertisement\ 1\ [\,]\ (1 :: 0 :: [\,])\\
&tc = \langle\, A \,!\, 1 \wedge A \,\sharp a \wedge B \,!\, 0 \,\rangle\ reveal\ [\,a\,] \Rightarrow withdraw\ A \vdash \ldots\ \oplus\ after\ t : withdraw\ B\\
&tc\text{-}semantics : \langle\, A , 1 \,\rangle^{d} \twoheadrightarrow \langle\, A , 1 \,\rangle^{d} \mid A : a \,\sharp 6\\
&tc\text{-}semantics =\\
&\quad begin\\
&\qquad \langle\, A , 1 \,\rangle^{d}\\
&\quad \longrightarrow\langle\, C\text{-}Advertise \,\rangle\\
&\qquad `tc \mid \langle\, A , 1 \,\rangle^{d}\\
&\quad \longrightarrow\langle\, C\text{-}AuthCommit \,\rangle\\
&\qquad `tc \mid \langle\, A , 1 \,\rangle^{d} \mid \langle\, A : a \,\sharp 6 \,\rangle \mid A\,[\,\sharp \triangleright\ \ tc\,]\\
&\quad \longrightarrow\langle\, C\text{-}AuthInit \,\rangle\\
&\qquad `tc \mid \langle\, A , 1 \,\rangle^{d} \mid \langle\, A : a \,\sharp 6 \,\rangle \mid A\,[\,\sharp \triangleright\ \ tc\,] \mid A\,[\,tc \triangleright^{s}\ 0\,]\\
&\quad \longrightarrow\langle\, C\text{-}Init \,\rangle\\
&\qquad \langle\, tc , 1 \,\rangle^{c} \mid \langle\, A : a \,\sharp\, inj_1\ 6 \,\rangle\\
&\quad \longrightarrow\langle\, C\text{-}AuthRev \,\rangle\\
&\qquad \langle\, tc , 1 \,\rangle^{c} \mid A : a \,\sharp 6\\
&\quad \longrightarrow\langle\, C\text{-}Control \,\rangle\\
&\qquad \langle\, [\,reveal\ [\,a\,] \Rightarrow withdraw\ A \vdash \ldots\,] , 1 \,\rangle^{c} \mid A : a \,\sharp 6\\
&\quad \longrightarrow\langle\, C\text{-}PutRev \,\rangle\\
&\qquad \langle\, [\,withdraw\ A\,] , 1 \,\rangle^{c} \mid A : a \,\sharp 6\\
&\quad \longrightarrow\langle\, C\text{-}Withdraw \,\rangle\\
&\qquad \langle\, A , 1 \,\rangle^{d} \mid A : a \,\sharp 6\\
&\quad \square
\end{aligned}
$$

At first, A holds a deposit of ₿ 1, as required by the contract's precondition. Then, the contract is advertised and the participants slowly provide the corresponding prerequisites (i.e. A commits to a secret via $[\,C\text{-}AuthCommit\,]$ and spends the required deposit via $[\,C\text{-}AuthInit\,]$, while B does not do anything). After all pre-conditions have been satisfied, the contract is stipulated (rule $[\,C\text{-}Init\,]$) and the secret is successfully revealed (rule $[\,C\text{-}AuthRev\,]$). Finally, the first branch is picked (rule $[\,C\text{-}Control\,]$) and A retrieves her deposit back (rules $[\,C\text{-}PutRev\,]$ and $[\,C\text{-}Withdraw\,]$).

4 FUTURE

... instead of BitML->Bitcoin [2] ...
... BitML->FormalUTxO [3] [?] ...
... will be easier with the added expressivity from data scripts [?] ... c.f. Marlowe [?] ...

REFERENCES

[] 2019. The Extended UTxO Model. Retrieved 5/2019 from https://github.com/input-output-hk/plutus/blob/master/docs/extended-utxo/README.md

[2] Massimo Bartoletti and Roberto Zunino. 2018. *BitML: a calculus for Bitcoin smart contracts.* Technical Report. Cryptology ePrint Archive, Report 2018/122.

[] Andrew Miller, Zhicheng Cai, and Somesh Jha. 2018. Smart contracts and opportunities for formal methods. In *International Symposium on Leveraging Applications of Formal Methods*. Springer, 280–299.

[3] https://github.com/omelkonian/formal-utxo

[8] Ulf Norell. 2008. Dependently typed programming in Agda. In *International School on Advanced Functional Programming*. Springer, 230–266.

[] Pablo Lamela Seijas and Simon Thompson. 2018. Marlowe: Financial contracts on blockchain. In *International Symposium on Leveraging Applications of Formal Methods*. Springer, 356–375.

[] Tim Sheard, Aaron Stump, and Stephanie Weirich. 2010. Language-based verification will change the world. (2010).

[] Joachim Zahnentferner. 2018. An Abstract Model of UTxO-based Cryptocurrencies with Scripts. *IACR Cryptology ePrint Archive* 2018 (2018), 469.