2nd-year PhD Report

Orestis Melkonian

October 11, 2021

THE UNIVERSITY of EDINBURGH

INPUT | OUTPUT

Year I: *Recap*

Recap

Mechanising the meta-theory of two separate objects of study:

- **BitML**: Bitcoin Modelling Language
- The (extended) **UTxO** model

UTxO [2018-2020]

MSc

$$\begin{tikzcd}
\text{UTxO} \arrow[r, "\textit{+ datum values}", "\textit{+ context}"'] & \text{EUTxO} \arrow[r, "\textit{+ multi-currency}"] \arrow[d, phantom, "\approx"] & \text{EUTxO}_{ma} \arrow[d, phantom, "\approx"] \\
 & \text{CEMs} & \text{Traced CEMs}
\end{tikzcd}$$

PhD-$\mathfrak{J}$

UTxO [2018-2020]

MSc

$$\begin{tikzcd}[column sep=huge]
\mathrm{UTxO} \arrow[r, "+\ \mathit{datum\ values}", "+\ \mathit{context}"'] & \mathrm{EUTxO} \arrow[r, "+\ \mathit{multi\text{-}currency}"] \arrow[d, phantom, "\approx"] & \mathrm{EUTxO}_{ma} \arrow[d, phantom, "\approx"] \\
& \text{CEMs} & \text{Traced CEMs}
\end{tikzcd}$$

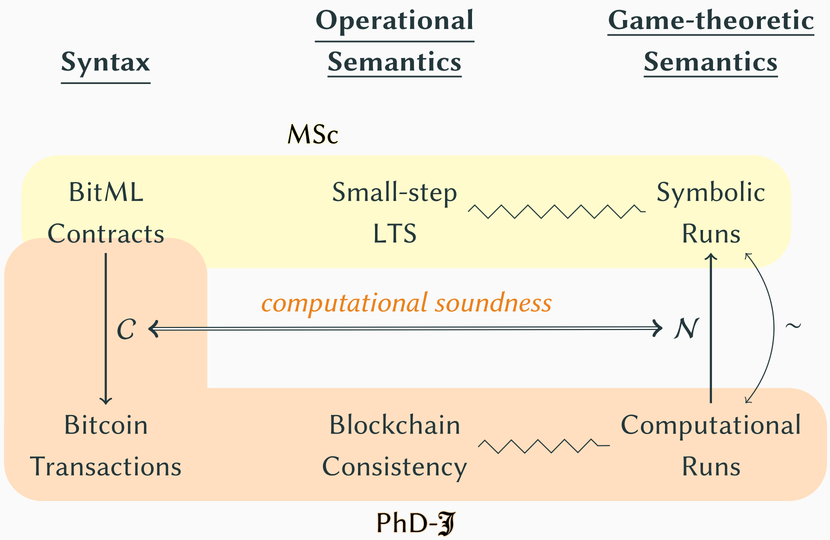
PhD-$\mathfrak{J}$

WTSC @ FC'20

The Extended UTxO Model

M.Chakravarty, J.Chapman, K.MacKenzie, O.Melkonian, M.P.Jones, P.Wadler

BitML [2018-2020]



Year II: *Where I've been...*

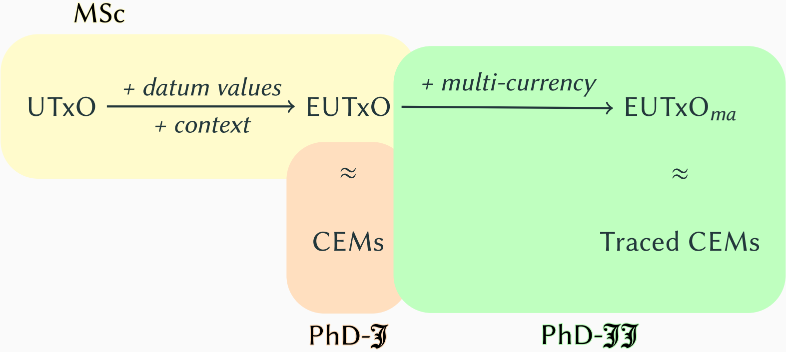
UTxO [2020-2021]

MSc

$$\begin{tikzcd}
\mathrm{UTxO} \arrow[r, "+\ \textit{datum values}", "+\ \textit{context}"'] & \mathrm{EUTxO} \arrow[r, "+\ \textit{multi-currency}"] \arrow[d, phantom, "\approx"] & \mathrm{EUTxO}_{ma} \arrow[d, phantom, "\approx"] \\
 & \mathrm{CEMs} & \text{Traced CEMs}
\end{tikzcd}$$

PhD-$\mathfrak{J}$

UTxO [2020-2021]



UTxO [2020-2021]

MSc

$$\begin{tikzcd}
\text{UTxO} \arrow[r, "+\,\textit{datum values}", "+\,\textit{context}"'] & \text{EUTxO} \arrow[r, "+\,\textit{multi-currency}"] \arrow[d, phantom, "\approx"] & \text{EUTxO}_{ma} \arrow[d, phantom, "\approx"] \\
& \text{CEMs} & \text{Traced CEMs}
\end{tikzcd}$$

PhD-$\mathfrak{I}$ PhD-$\mathfrak{II}$

RSC @ ISoLA'20: *UTxO$_{ma}$: UTxO with Multi-Asset Support*

RSC @ ISoLA'20: *Native Custom Tokens in the Extended UTxO Model*

Separation Logic for UTxO

- In collaboration with W.Swierstra (UU) and J.Chapman (IOHK)

- In collaboration with W.Swierstra (UU) and J.Chapman (IOHK)

Blockchain		Concurrency Theory
ledgers	$\leftrightarrow$	computer memory
memory locations	$\leftrightarrow$	accounts
data values	$\leftrightarrow$	account balances
smart contracts	$\leftrightarrow$	programs accessing memory

- In collaboration with W.Swierstra (UU) and J.Chapman (IOHK)

Blockchain		**Concurrency Theory**
ledgers	$\leftrightarrow$	computer memory
memory locations	$\leftrightarrow$	accounts
data values	$\leftrightarrow$	account balances
smart contracts	$\leftrightarrow$	programs accessing memory

💡 Transfer results from (Concurrent) Separation Logic!

Hoare-style semantics and correspondences

Operational

$$\frac{P(s) \quad (l, s) \rightsquigarrow^* s'}{Q(s')}$$

$\simeq$

Denotational

$$P \vdash Q \circ [\![l]\!]$$

$\simeq$ $\simeq$

Axiomatic

$$\{P\} l \{Q\}$$

Hoare-style semantics and correspondences

Operational

$$\frac{P(s) \quad (l, s) \rightsquigarrow^* s'}{Q(s')}$$

$\simeq$

Denotational

$$P \vdash Q \circ [\![l]\!]$$

$\simeq$ $\simeq$

Axiomatic

$$\{P\} l \{Q\}$$

SL: [FRAME] rule

$$\frac{l \# R \quad \{P\} l \{Q\}}{\{P * R\} l \{Q * R\}}$$

Hoare-style semantics and correspondences

Operational

$$\frac{P(s) \quad (l, s) \rightsquigarrow^* s'}{Q(s')}$$

$\simeq$

Denotational

$$P \vdash Q \circ [\![l]\!]$$

$\simeq$ $\simeq$

Axiomatic

$$\{P\} l \{Q\}$$

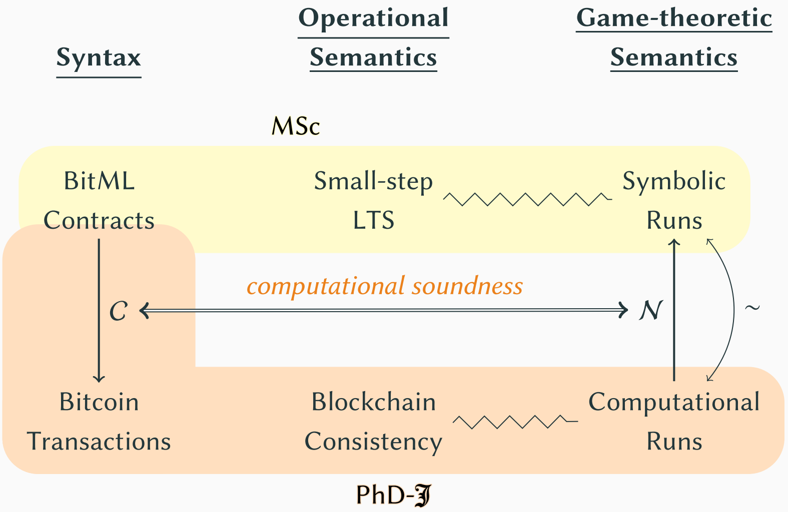
SL: [FRAME] rule

$$\frac{l \# R \quad \{P\} l \{Q\}}{\{P * R\} l \{Q * R\}}$$

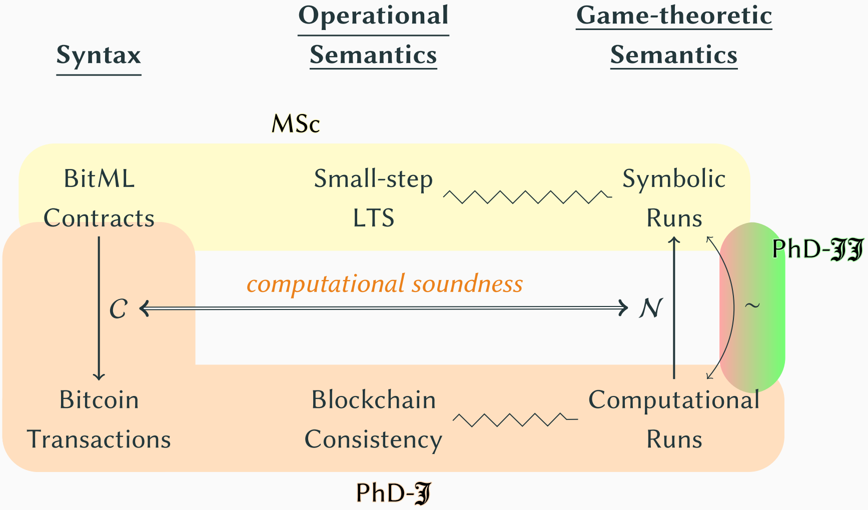
CSL: [PARALLEL] rule

$$\frac{\begin{array}{ccc} l_1 \parallel l_2 = l & l_1 \# P_2 & l_2 \# P_1 \\ \{P_1\} l_1 \{Q_1\} & & \{P_2\} l_2 \{Q_2\} \end{array}}{\{P_1 * P_2\} l \{Q_1 * Q_2\}}$$

BitML [2020-2021]



BitML [2020-2021]



BitML: Coherence

Definition 20 (Coherence). We inductively define the relation $coher(R^s, R^c, r, txout, sechash, \kappa)$, where (i) R^s is a symbolic run, (ii) R^c is a computational run, (iii) r is a randomness source, (iv) $txout$ is an injective function from names x (occurring in R^s) to transaction ouputs (T, o) (where T occurs in R^c), respecting values; (v) $sechash$ is a mapping from secret names a (occurring in R^s) to bitstrings; (vi) κ maps triples $(\{G\}C, D, \mathsf{A})$, where D is a subterm of C, to public keys.

Base case: $coher(R^s, R^c, r, txout, sechash, \kappa)$ holds if all the following conditions hold: (i) $R^s = \Gamma_0 \mid 0$, with Γ_0 initial; (ii) $R^c = \mathsf{T}_0 \cdots$ initial; (iii) all the public keys in R^c are generated from r, according to Definition 13; (iv) $txout$ maps exactly the x of $\langle \mathsf{A}, v \rangle_x$ in Γ_0 to an output in T_0 of value $v\mathbb{B}$, and spendable with $\hat{K}_{\mathsf{A}}(r_{\mathsf{A}})$; (v) dom $sechash = \emptyset$; (vi) dom $\kappa = \emptyset$.

Inductive case: $coher(\dot{R}^s \xrightarrow{\alpha} \Gamma \mid t, \dot{R}^c\lambda^c, r, txout, sechash, \kappa)$ holds if $coher(\dot{R}^s, \dot{R}^c, r, txout', sechash', \kappa')$ and one of the following cases applies.

(1) $\alpha = advertise(\{G\}C)$, $\lambda^c = \mathsf{A} \rightarrow * : C$, where C is obtained by encoding $\{G\}C$ as a bitstring, representing each x in it as the transaction output $txout'(x)$. Further, $txout' = txout$, $sechash' = sechash$, and $\kappa' = \kappa$.

(2) $\alpha = \mathsf{A} : \{G\}C, \Delta$, where: (i) for some B, $\dot{R}^c$ contains $\mathsf{B} \rightarrow * : C$, where C is obtained from $\{G\}C$ and $txout'$ as in Item 1. Note that $\dot{R}^c$ might contain several such messages; below, we let C represent the first occurrence. (ii) for some B, $\lambda^c = \mathsf{B} \rightarrow * : (C, \vec{h}, \vec{k})$ (signed by A), where $\vec{h}$ is a sequence comprising a bitstring h_i with $|h_i| = \eta$ for each secret a_i in Δ, and $\vec{k}$ is a sequence of keys, as the one produced by the stipulation protocol. We require that λ^c is the first occurrence, in the run $\dot{R}^c$, of such a message after C. (iii) Let N_i be the length of a_i fixed in Δ. If $N_i \neq \bot$, we require that $\dot{R}^c$ contains, for some B, a query to the oracle $\mathsf{B} \rightarrow \mathsf{O} : m_i$, and a subsequent reply $\mathsf{O} \rightarrow \mathsf{B} : h_i$ such that $|m_i| = \eta + N_i$. Otherwise, if $N_i = \bot$, we require that h_i does not occur as a reply from O to any query of length $\geq \eta$. (iv) No hash is reused: the h_i are pairwise distinct, and also distinct from $sechash'(b)$ for any $b \in$ dom $(sechash')$. (v) $txout = txout'$. (vi) $sechash$ extends $sechash'$ so that for each secret a_i we have $sechash(a_i) = h_i$. (vii) If $\mathsf{A} \in$ Hon, we define κ by extending κ' according to $\vec{k}$, so to record the public keys of all participants occurring in G for each subterm D of C. If κ' already defines such keys, or $\mathsf{A} \notin$ Hon, we let $\kappa = \kappa'$.

(3) $\alpha = \mathsf{A} : \{G\}C, x$, where: (i) $\lambda^c = \mathsf{B} \rightarrow * : m$ for some B, where m is the signature of the transaction T_{init} of $\mathbb{B}_{\mathrm{adv}}(\{G\}C)$ relatively to the input x with $\hat{K}_{\mathsf{A}}(r_{\mathsf{A}})$. The parameters of the compiler are set as follows: part, PartG and val are inferred from G, we let txout $= txout'$, sechash $= sechash'$, and $\mathsf{K}(\mathsf{B}) = \hat{K}^p_{\mathsf{B}}(r_{\mathsf{B}})$, $\mathsf{K}(D, \mathsf{B}) = \kappa'(\{G\}C, D, \mathsf{B})$ for each B, in $\mathbb{B}_{\mathrm{adv}}(\{G\}C)$. The needed compiler parameters are obtained as in Item 3. (iii) $sechash = sechash'$, $\kappa = \kappa'$, and $txout$ extends $txout'$, mapping z to T_{init}.

(5) $\alpha = \mathsf{A} : x, D$, where: (i) $\dot{R}^s$ contains $\langle C', v \rangle_x$ with $C' = D + \sum_i D_i$, for some $D = \mathsf{A} : D'$. (ii) In $\dot{R}^s$, we find that $\langle C', v \rangle_x$ has $\{G\}C$ as its ancestor advertisement. (iii) $\lambda^c = \mathsf{B} \rightarrow * : m$, where m is a signature with key $\kappa'(\{G\}C, D, \mathsf{A})$ of the first transaction T in $\mathbb{B}_{\mathrm{D}}(D, D, \mathsf{T}', o, v, \mathrm{PartG}, 0)$, where $(\mathsf{T}', o) = txout'(x)$. The compiler parameters are obtained as in Item 3. (iv) $txout = txout'$, $sechash = sechash'$, and $\kappa = \kappa'$. (v) $\dot{R}^c$ contains $\mathsf{B} \rightarrow * : \mathsf{T}$ for some B, and m is the first signature of T in $\dot{R}^c\lambda^c$ after the first broadcast of T.

(6) $\alpha = put(\vec{x}, \vec{a}, y)$, where: (i) $\vec{x} = x_1 \cdots x_k$. (ii) In $\Gamma_{\dot{R}^s}$, the action α consumes $\langle D + C, v \rangle_y$ and the deposits $\langle \mathsf{A}_i, v_i \rangle_{x_i}$ to produce $\langle C', v' \rangle_{y'}$, where $D = \cdots : \mathtt{put} \cdots \mathtt{reveal} \cdots . C'$. Let t be maximum deadline in an `after` in front of D. (iii) In $\dot{R}^s$, we find that $\langle D + C, v \rangle_y$ has $\{G\}C''$ as its ancestor advertisement, for some G and C''. (iv) $\lambda^c = \mathsf{T}$ where T is the first transaction of $\mathbb{B}_{\mathrm{C}}(C', D, \mathsf{T}', o, v', \vec{x}, \mathrm{PartG}, t)$, where $(\mathsf{T}', o) = txout'(y)$. The compiler parameters are obtained as in Item 3. (v) $txout$ extends $txout'$ so that y' is mapped to $(\mathsf{T}, 0)$, $sechash = sechash'$, and $\kappa = \kappa'$.

(7) $\alpha = \mathsf{A} : a$, where: (i) $\lambda^c = \mathsf{B} \rightarrow * : m$ from some B with $|m| \geq \eta$. (ii) $\dot{R}^c = \cdots (\mathsf{B} \rightarrow \mathsf{O} : m)(\mathsf{O} \rightarrow \mathsf{B} : sechash'(a)) \cdots$, for some B. (iii) $txout = txout'$, $sechash = sechash'$ and $\kappa = \kappa'$. (iv) In $\dot{R}^s$ we find an $\mathsf{A} : \{G\}C, \Delta$ action, with a in G, with a corresponding broadcast in $\dot{R}^c$ of $m' = (C, \vec{h}, \vec{k})$. (v) λ^c is the first broadcast of m in $\dot{R}^c$ after the first broadcast of m'.

(8) $\alpha = split(y)$, where: (i) In $\dot{R}^s$, the action α consumes $\langle D + C, v \rangle_y$ to obtain $\langle C_0, v_0 \rangle_{x_0} \mid \cdots \mid \langle C_k, v_k \rangle_{x_k}$ where $D = \cdots : \mathtt{split}\ \vec{v} \rightarrow \vec{C}$ and $\vec{C} = C_0 \ldots C_k$. Let t be the maximum deadline in an `after` in front of D. (ii) In $\dot{R}^s$, we find that $\langle D + C, v \rangle_y$ has $\{G\}C'$ as its ancestor advertisement. (iii) $\lambda^c = \mathsf{T}$ where T is the first transaction of $\mathbb{B}_{\mathrm{par}}(\vec{C}, D, \mathsf{T}', o, \mathrm{PartG}, t)$ where $(\mathsf{T}', o) = txout'(y)$. The compiler parameters are obtained as for Item 3. (iv) $txout$ extends $txout'$ mapping each x_i to (T, i), $sechash = sechash'$, and $\kappa = \kappa'$.

(9) $\alpha = withdraw(\mathsf{A}, v, y)$, where: (i) In $\dot{R}^s$, the action α consumes $\langle D + C, v \rangle_y$ to obtain $\langle \mathsf{A}, v \rangle_x$, where $D = \cdots : \mathtt{withdraw}\ \mathsf{A}$. (ii) In $\dot{R}^s$, we find that $\langle D + C, v \rangle_y$ has $\{G\}C'$ as its ancestor advertisement. (iii) $\lambda^c = \mathsf{T}$ where T is the first transaction of $\mathbb{B}_{\mathrm{D}}(D, D, \mathsf{T}', o, v, \mathrm{PartG}, 0)$ where $(\mathsf{T}', o) = txout'(y)$. The compiler parameters are obtained as for Item 3. (iv) $txout$ extends $txout'$ mapping x to $(\mathsf{T}, 0)$, $sechash = sechash'$, and $\kappa = \kappa'$.

(10) $\alpha = \mathsf{A} : x, x'$, where: (i) In $\dot{R}^s$ we find $\langle \mathsf{A}, v \rangle_x$ and $\langle \mathsf{A}, v' \rangle_{x'}$. (ii) In $\dot{R}^c$ we find $\mathsf{B} \rightarrow * : \mathsf{T}$ for some B, T, where T has as its two inputs $txout'(x)$ and $txout'(x')$, and a single output of

(11) $\alpha = join(x, y)$, where: (i) In $\dot{R}^s$ the action α spends
and $\langle \mathsf{A}, v' \rangle_{x'}$ to obtain $\langle \mathsf{A}, v + v' \rangle_y$. (ii) $\lambda^c = \mathsf{T}$ is
action having as inputs $txout'(x)$ and $txout'(x')$, a
ing one output of value $v + v'$ redeemable with
(iii) $txout$ extends $txout'$ mapping y to $(\mathsf{T}, 0)$, sechash
and $\kappa = \kappa'$.

(12) $\alpha = \mathsf{A} : x, v, v'$. Similar to Item 10.

(13) $\alpha = divide(x, v, v')$. Similar to Item 11.

(14) $\alpha = \mathsf{A} : x, \mathsf{B}$. Similar to Item 10.

(15) $\alpha = donate(x, \mathsf{B})$. Similar to Item 11.

(16) $\alpha = \mathsf{A} : \vec{y}, j$, where: (i) $\vec{y} = y_1 \cdots y_k$. (ii) In $\dot{R}^s$
$\langle \mathsf{B}_i, v_i \rangle_{y_i}$ for $i \in 1..k$, with $\mathsf{B}_j = \mathsf{A}$. (iii) In $\dot{R}^c$
$\mathsf{B} \rightarrow * : \mathsf{T}$ for some B, T, where T has as its
$txout'(y_i)$ for $i \in 1..k$, and possibly others not in ra
(iv) $\lambda^c = \mathsf{B} \rightarrow * : m$ from some B, m where m is
ture of T with $\hat{K}_{\mathsf{A}}(r_{\mathsf{A}})$, corresponding to the j-t
(v) λ^c is the first broadcast of m in $\dot{R}^c$ after the firs
cast of T. (vi) λ^c does not correspond to any of th
cases, i.e. there is no other symbolic action α for
$\dot{R}^s\alpha$ would be coherent with $\dot{R}^c\lambda^c$. (vii) $txout$ =
$sechash = sechash'$, and $\kappa = \kappa'$.

(17) $\alpha = destroy(\vec{x})$, where: (i) $\vec{x} = x_1 \cdots x_k$. (ii) In $\dot{R}^s$,
sumes $\langle \mathsf{A}_i, v_i \rangle_{x_i}$ to obtain 0. (iii) $\lambda^c = \mathsf{T}$ from some
ing as inputs $txout'(x_1), \ldots, txout'(x_k)$, and possi
ers not in ran $txout'$. (iv) λ^c does not correspond to
the other cases, i.e. there is no other symbolic actio
which $\dot{R}^s\alpha$ would be coherent with $\dot{R}^c\lambda^c$. (v) $txout$ =
$sechash = sechash'$, and $\kappa = \kappa'$.

(18) $\alpha = \delta = \lambda^c$, and $txout = txout'$, $sechash$ = sechas
$\kappa = \kappa'$.

Inductive case 2: the predicate $coher(R^s, R^c\lambda^c, r, txout,$ sec
holds if $coher(R^s, R^c, r, txout, sechash, \kappa)$, and one of the fo
cases applies:

(1) $\lambda^c = \mathsf{T}$ where no input of T belongs to ran $txout$.

(2) $\lambda^c = \mathsf{A} \rightarrow \mathsf{O} : m$ or $\lambda^c = \mathsf{O} \rightarrow \mathsf{A} : m$, for some A, m

(3) $\lambda^c = \mathsf{A} \rightarrow * : m$, where λ^c does not correspond
symbolic move, according to the first inductive ca

We write $R^s \sim_r R^c$ iff $coher(R^s, R^c, r, txout, sechash, \kappa)$ fo
$txout$, $sechash$, and κ.

The following lemma is the active contracts analogous of L
Both results are proved by induction on the definition of coh

Lemma 6. *Let $coher(R^s, R^c, r, txout, sechash, \kappa)$. For each contract $\langle C, v \rangle_x$ occurring in Γ_{R^s}, there exists a corresponding transaction output (T, o) in $\mathbf{B}_{R^c}$ with value v. Further, T is ge by the invoking the compiler as $\mathbb{B}_{\mathrm{C}}(C, D_p, \mathsf{T}', o', v, I, \mathcal{P}, t)$ f values of $D_p, \mathsf{T}', o', I, \mathcal{P}, t$, or as $\mathbb{B}_{\mathrm{par}}(\vec{C}, D_p, \mathsf{T}'o', \vec{v}, \mathcal{P}, t)$ f values of $\vec{C}, D_p, \mathsf{T}', o', \vec{v}, \mathcal{P}, t$ such that $C = \vec{C}_{o+1}$ and v = using parameters txout, sechash, κ.*

```
data Tree {l u : Nat} : Set where
  Leaf  : {pf : l ≤ u} → Tree {l} {u}
  Node : (x : Nat)
    → Tree {l} {x} → Tree {x} {u}
    → Tree {l} {u}
{-# COMPILE AGDA2HS Tree #-}

insert : {l u : Nat} (x : Nat)
  → Tree {l} {u}
  → {l ≤ x} → {x ≤ u}
  → Tree {l} {u}
insert x Leaf {l≤x} {x≤u} =
  Node x (Leaf {pf = l≤x}) (Leaf {pf = x≤u})
insert x (Node y l r) {l≤x} {x≤u} =
  case compare x y of λ where
    (LT {pf = x≤y}) → Node y (insert x l {l≤x} {x≤y}) r
    (EQ {pf = x≡y}) → Node y l r
    (GT {pf = y≤x}) → Node y l (insert x r {y≤x} {x≤u})
{-# COMPILE AGDA2HS insert #-}
```

```haskell
data Tree = Leaf
          | Node Natural Tree Tree

insert :: Natural -> Tree -> Tree
insert x Leaf = Node x Leaf Leaf
insert x (Node y l r)
  = case compare x y of
      LT -> Node y (insert x l) r
      EQ -> Node x l r
      GT -> Node y l (insert x r)
```

AGDA2HS: TYPECLASSES

```
record Show (a : Set) : Set where
  field show      : a → String
        showsPrec : Nat → a → ShowS
        showList  : List a → ShowS
record Show₁ (a : Set) : Set where
  field showsPrec : Nat → a → ShowS
  show : a → String
  show x = showsPrec 0 x ""
  showList : List a → ShowS
  showList = defaultShowList (showsPrec 0)
record Show₂ (a : Set) : Set where
  field show : a → String
  showsPrec : Nat → a → ShowS
  showsPrec _ x s = show x ++ s
  showList : List a → ShowS
  showList = defaultShowList (showsPrec 0)
open Show {{...}}
{-# COMPILE AGDA2HS Show class Show₁ Show₂ #-}
instance
  ShowMaybe : {{Show a}} → Show (Maybe a)
  ShowMaybe {a = a} = record {Show₁ s₁}
    where
    s₁ : Show₁ (Maybe a)
    s₁ .Show₁.showsPrec n = λ where
      Nothing → showString "nothing"
      (Just x)  → showParen true
        (showString "just " ∘ showsPrec 10 x)
{-# COMPILE AGDA2HS ShowMaybe #-}
```

```haskell
class Show a where
  show :: a -> String
  showsPrec :: Natural -> a -> ShowS
  showList :: [a] -> ShowS
  {-# MINIMAL showsPrec | show #-}
  show x = showsPrec 0 x ""
  showList = defaultShowList (showsPrec 0)
  showsPrec _ x s = show x ++ s

instance (Show a) => Show (Maybe a) where
  showsPrec n = \case
    Nothing -> showString "nothing"
    (Just x) -> showParen True
      (showString "just " . showsPrec 10 x)
```

AGDA2HS: TYPECLASSES

```
record Show (a : Set) : Set where
  field show       : a → String
        showsPrec : Nat → a → ShowS
        showList   : List a → ShowS
record Show₁ (a : Set) : Set where
  field showsPrec : Nat → a → ShowS
  show : a → String
  show x = showsPrec 0 x ""
```

CPP @ POPL'22

Reasonable Agda is Correct Haskell: Intrinsic Program Verification using AGDA2HS

J.Cockx, O.Melkonian, J.Chapman, U.Norell + TU Delft students

```
  showList = defaultShowList (showsPrec 0)
open Show {{...}}
{-# COMPILE AGDA2HS Show class Show₁ Show₂ #-}
instance
  ShowMaybe : {{Show a}} → Show (Maybe a)
  ShowMaybe {a = a} = record {Show₁ s₁}
    where
    s₁ : Show₁ (Maybe a)
    s₁ .Show₁.showsPrec n = λ where
      Nothing → showString "nothing"
      (Just x)  → showParen true
        (showString "just " ∘ showsPrec 10 x)
{-# COMPILE AGDA2HS ShowMaybe #-}
```

```haskell
  showsPrec _ x s = show x ++ s

instance (Show a) => Show (Maybe a) where
  showsPrec n = \case
    Nothing -> showString "nothing"
    (Just x) -> showParen True
      (showString "just " . showsPrec 10 x)
```

setup-agda: CI infrastructure for Agda

```yaml
name: CI
on: push: {branches: master}
jobs:
  build-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2.3.1
      - uses: omelkonian/setup-agda@v0.1
        with:
          agda-version: 2.6.1.3
          stdlib-version: 1.6
          libraries: |
            omelkonian/formal-prelude#92ef
            omelkonian/formal-bitcoin#0341
            omelkonian/formal-bitml#4382
          main: Main
          token: ${{ secrets.GITHUB_TOKEN }}
```

Modular Automatic Solvers for Agda proofs

- define strategies for automatic proof search
- should be able to define solvers incrementally for specific types
- primarily achieved with Agda's reflection

Modular Automatic Solvers for Agda proofs

- define strategies for automatic proof search
- should be able to define solvers incrementally for specific types
- primarily achieved with Agda's reflection

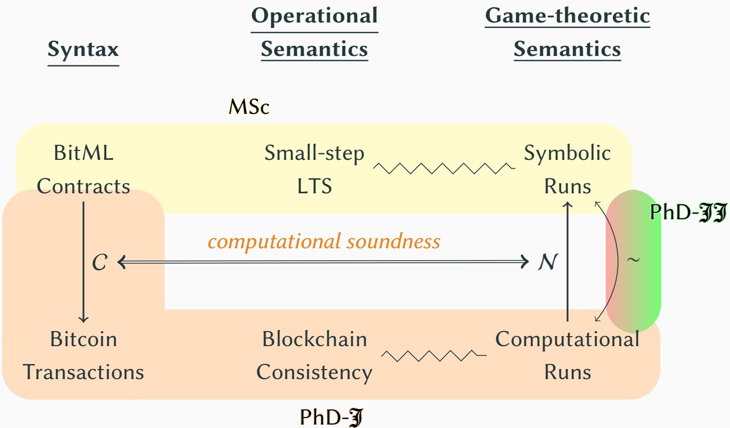
```
open import Prelude.Init using (List)
open import Prelude.Semigroup
open import Prelude.Membership
open import Prelude.Solvers

_ : ∀ {A : Set} {y : A} {xs ys zs : List A}
  → y ∈ ys → y ∈ xs ◇ ys ◇ zs
_ = solve
```

Year III: *Where I'm going...*

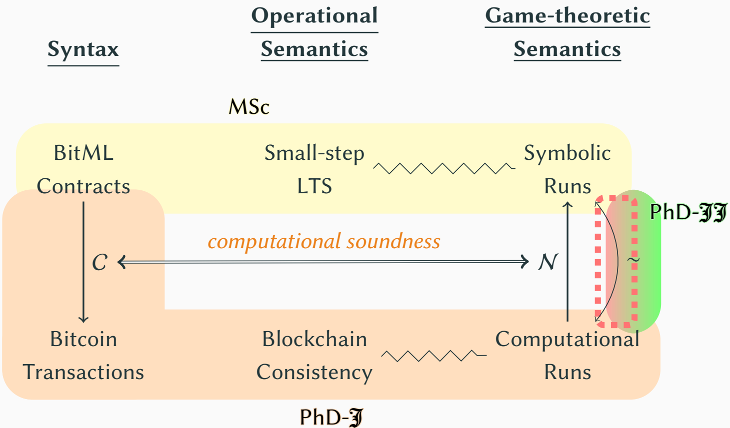
BitML [2021 - mid 2022]

1. Finish up coherence
2. Symbolic ⟶ computational runs
3. Prove computational soundness: compiler preserves coherence
4. Write a paper about it!



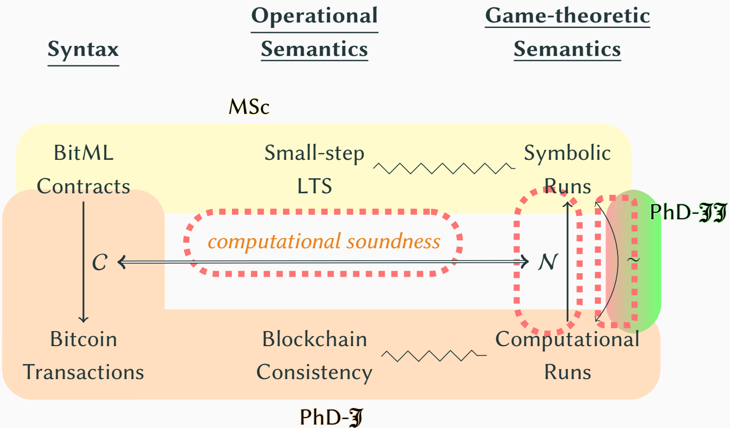
BitML [2021 - mid 2022]

1. Finish up coherence
2. Symbolic $\longrightarrow$ computational runs
3. Prove computational soundness: compiler preserves coherence
4. Write a paper about it!



BitML [2021 - mid 2022]

1. Finish up coherence
2. Symbolic ⟶ computational runs
3. Prove computational soundness: compiler preserves coherence
4. Write a paper about it!



Separation Logic for Blockchain [2021 - mid 2022]

1. Obvious next step: extend results to UTxO ledgers
2. Write a paper about it!

$$\begin{tikzcd}[column sep=large]
\begin{array}{c}\textbf{Operational}\\ \dfrac{P(s) \quad (l, s) \rightsquigarrow^{*} s'}{Q(s')}\end{array} \arrow[rr, dash, dotted, "\simeq"] \arrow[dr, dash, dotted, "\simeq"'] & & \begin{array}{c}\textbf{Denotational}\\ P \vdash Q \circ [\![l]\!]\end{array} \arrow[dl, dash, dotted, "\simeq"] \\
& \begin{array}{c}\textbf{Axiomatic}\\ \{P\} l \{Q\}\end{array} &
\end{tikzcd}$$

Thesis write-up [mid 2022 - late 2022]

- Hopefully by then, enough material to fill a thesis
- Ideally, two more papers on BitML and UTxO at prestigious venues
- Realistically, UTxO exploration alongside thesis writing

Discussion

- More ambitious directions (alas, no time)
  - **agda2hs:** extract executable programs from my mechanisations
  - **BitML:** improve/re-formulate (e.g. BitML $\rightarrow$ EUTxO)
  - **EUTxO:** further extentions / state machine verification

Discussion

- More ambitious directions (alas, no time)
    - **agda2hs:** extract executable programs from my mechanisations
    - **BitML:** improve/re-formulate (e.g. BitML → EUTxO)
    - **EUTxO:** further extentions / state machine verification
- Internship?
    - some interesting positions/projects so far
      🤔 is it worth it though?

Discussion

- More ambitious directions (alas, no time)
  - **agda2hs:** extract executable programs from my mechanisations
  - **BitML:** improve/re-formulate (e.g. BitML $\longrightarrow$ EUTxO)
  - **EUTxO:** further extentions / state machine verification
- Internship?
  - some interesting positions/projects so far
    🤔 is it worth it though?
- Extension?
  - a few more months would lead to more results
    🤔 is it worth it though?

Discussion