BitML^x

Cross-chain Smart Contracts for Bitcoin-style Cryptocurrencies

Federico Badaloni* Chrysoula Oikonomou**
Clara Schneidewind* Pedro Moreno-Sanchez**

*Max Planck Institute for Security and Privacy

**IMDEA Software Institute

FCS 2023
July 9, 2023

Introduction

- Smart contracts: programs running on blockchains, moving cryptocurrencies.

Introduction

- Smart contracts: programs running on blockchains, moving cryptocurrencies.
- Not all blockchains have built-in support for smart contracts.

Introduction

- Smart contracts: programs running on blockchains, moving cryptocurrencies.
- Not all blockchains have built-in support for smart contracts.
- Bitcoin-style cryptocurrencies have limited scripting capabilities.

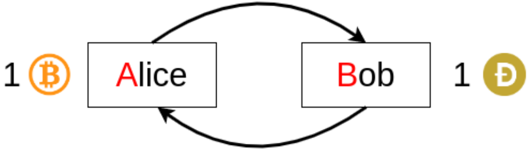
Introduction

- Smart contracts: programs running on blockchains, moving cryptocurrencies.
- Not all blockchains have built-in support for smart contracts.
- Bitcoin-style cryptocurrencies have limited scripting capabilities.
- BitML: high-level language for contracts on Bitcoin.

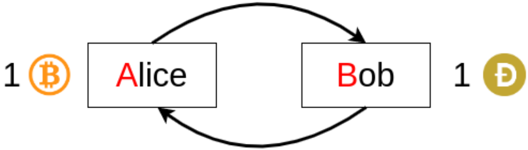
Introduction

- Smart contracts: programs running on blockchains, moving cryptocurrencies.
- Not all blockchains have built-in support for smart contracts.
- Bitcoin-style cryptocurrencies have limited scripting capabilities.
- BitML: high-level language for contracts on Bitcoin.
- BitML^x: cross-blockchain, synchronous, BitML.

Alice and Bob want to swap coins

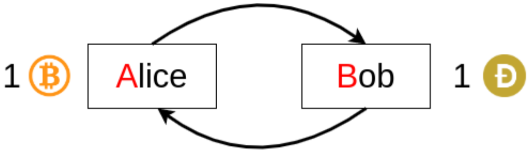


Alice and Bob want to swap coins



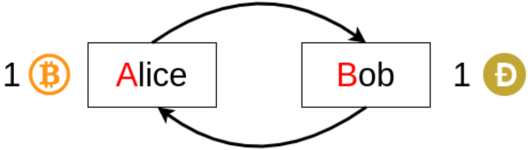
$$\{A :! 1฿ \mid A : \texttt{secret}\ a\}Swap^{฿}$$

Alice and Bob want to swap coins



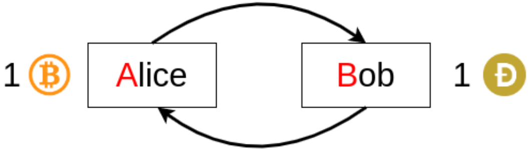
$$\{\overbrace{A :! 1\text{฿} \mid A : \texttt{secret } a}^{\text{preconditions}}\}\ \overbrace{\mathit{Swap}^{\text{฿}}}^{\text{code}}$$

Alice and Bob want to swap coins



$$\{ \overbrace{\underbrace{A :! 1\text{₿}}_{\text{user deposits}} \mid \underbrace{A : \texttt{secret}\ a}_{\text{secret commitments}}}^{\text{preconditions}} \} \overbrace{\mathit{Swap}^{\text{₿}}}^{\text{code}}$$

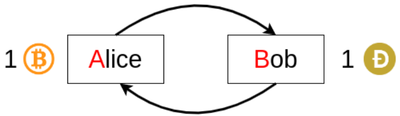
Alice and Bob want to swap coins



$$\overbrace{\{\,\underbrace{A :!\, 1\text{₿}}_{\text{user deposits}} \mid \underbrace{A : \texttt{secret}\ a}_{\text{secret commitments}}\,\}}^{\text{preconditions}}\ \overbrace{\mathit{Swap}^{\text{₿}}}^{\text{code}}$$

$$\{B :!\, 1\text{Đ} \mid B : \texttt{secret}\ b\}\mathit{Swap}^{\text{Đ}}$$

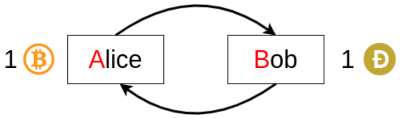
BitML example



$$\{A :! 1\text{฿} \mid A : \texttt{secret } a\}Swap^{\text{฿}}$$

$$\{B :! 1\text{Đ} \mid B : \texttt{secret } b\}Swap^{\text{Đ}}$$

BitML example

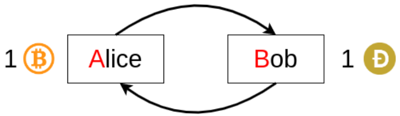


$$\{A :! 1\text{B} \mid A : \texttt{secret } a\}Swap^{\text{B}}$$

$$\{B :! 1\text{Đ} \mid B : \texttt{secret } b\}Swap^{\text{Đ}}$$

$$Swap^{\text{B}} = Exchange^{\text{B}} + Refund^{\text{B}}$$

BitML example



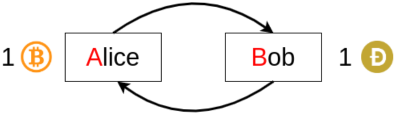
$$\{A :! 1\text{₿} \mid A : \texttt{secret } a\}\mathit{Swap}^{\text{₿}}$$

$$\{B :! 1\text{Ð} \mid B : \texttt{secret } b\}\mathit{Swap}^{\text{Ð}}$$

$$\mathit{Swap}^{\text{₿}} = \mathit{Exchange}^{\text{₿}} + \mathit{Refund}^{\text{₿}}$$

$$\mathit{Exchange}^{\text{₿}} = \texttt{reveal } a \,.\, \texttt{withdraw } B$$

BitML example



$$\{A \;{:}!\; 1\text{₿} \mid A : \; \texttt{secret } a\}\,\mathit{Swap}^{\text{₿}}$$

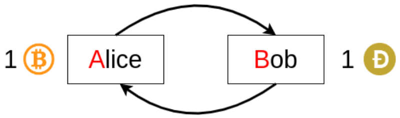
$$\{B \;{:}!\; 1\text{Ɖ} \mid B : \; \texttt{secret } b\}\,\mathit{Swap}^{\text{Ɖ}}$$

$$\mathit{Swap}^{\text{₿}} = \mathit{Exchange}^{\text{₿}} + \mathit{Refund}^{\text{₿}}$$

$$\mathit{Exchange}^{\text{₿}} = \texttt{reveal } a \;.\; \texttt{withdraw } B$$

$$\mathit{Refund}^{\text{₿}} = \texttt{after } t : \; \texttt{withdraw } A$$

BitML example



$\{A \,:!\, 1₿ \mid A : \ \texttt{secret}\ a\}\mathit{Swap}^{₿}$

$\{B \,:!\, 1Đ \mid B : \ \texttt{secret}\ b\}\mathit{Swap}^{Đ}$

$$\mathit{Swap}^{₿} = \mathit{Exchange}^{₿} + \mathit{Refund}^{₿}$$

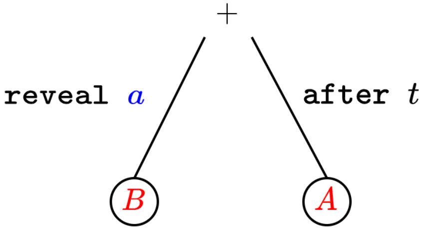
$$\mathit{Exchange}^{₿} = \texttt{reveal}\ a\ .\ \texttt{withdraw}\ B$$

$$\mathit{Refund}^{₿} = \texttt{after}\ t :\ \texttt{withdraw}\ A$$

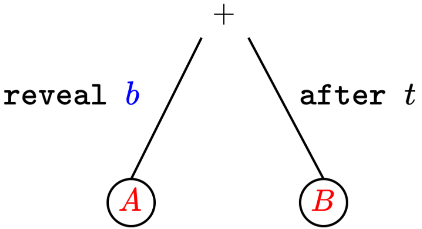
$$\mathit{Swap}^{Đ} = \texttt{reveal}\ b\ .\ \texttt{withdraw}\ A + \texttt{after}\ t :\ \texttt{withdraw}\ B$$

Syntax trees

Alice



Bob

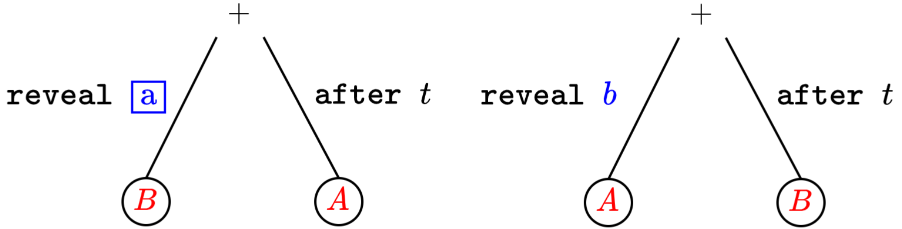


Syntax trees

Alice

- Here is $\boxed{a}$.

Bob



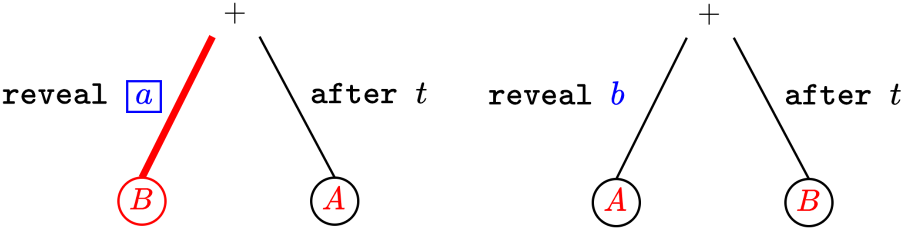
Syntax trees

Alice

- Here is $\boxed{a}$.

Bob

- Thanks!



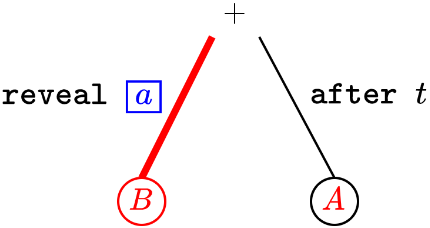
Syntax trees

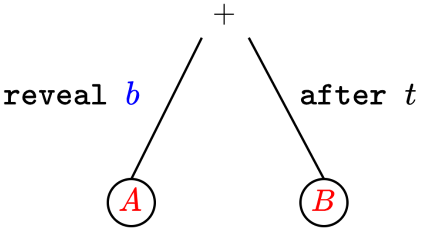
Alice

- Here is $\boxed{a}$.
- Can I have b now?

Bob

- Thanks!





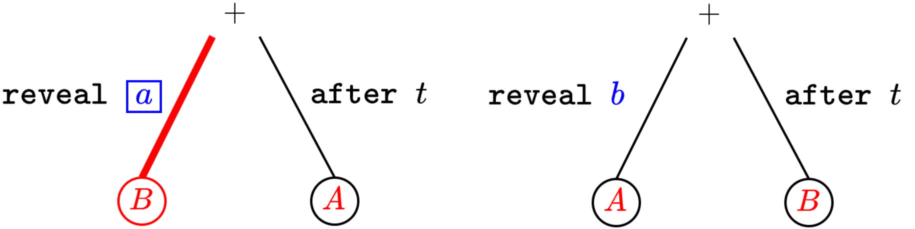
Syntax trees

Alice

- Here is $\boxed{a}$.
- Can I have b now?

Bob

- Thanks!
- ...



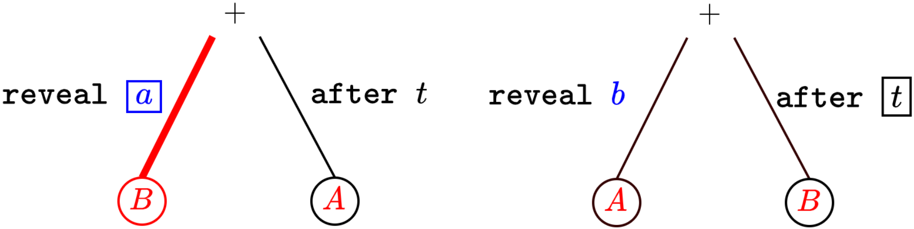
Syntax trees

Alice

- Here is $\boxed{a}$.
- Can I have b now?
- Hello?

Bob

- Thanks!
- ...



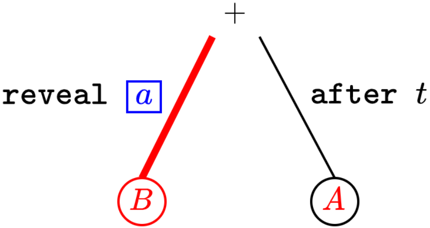
Syntax trees

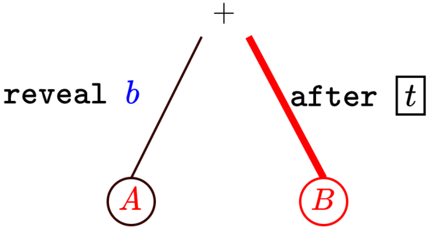
Alice

- Here is $\boxed{a}$.
- Can I have b now?
- Hello?

Bob

- Thanks!
- ...
- ...





Example in BitML^x

Read the bibliography on sound cryptographic protocol designs.

Example in BitML^x

~~Read the bibliography on sound cryptographic protocol designs.~~

Or let BitML^x do it for them!

Example in BitML^x

~~Read the bibliography on sound cryptographic protocol designs.~~

Or let BitML^x do it for them!

$$\{A :! (1\text{฿}, 0\text{Đ}) \mid B :! (0\text{฿}, 1\text{Đ})\}\mathit{Swap}^x$$

Example in BitML^x

~~Read the bibliography on sound cryptographic protocol designs.~~

Or let BitML^x do it for them!

$$\{A :! (1\text{B}, 0\text{D}) \mid B :! (0\text{B}, 1\text{D})\}\mathit{Swap}^x$$

$$\mathit{Swap}^x = \mathit{Exchange}^x +\!\!> \mathit{Refund}^x$$

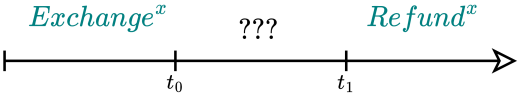
Example in BitML^x

~~Read the bibliography on sound cryptographic protocol designs.~~

Or let BitML^x do it for them!

$$\{A :! (1\text{₿}, 0\text{Đ}) \mid B :! (0\text{₿}, 1\text{Đ})\}\mathit{Swap}^x$$

$$\mathit{Swap}^x = \mathit{Exchange}^x \mathrel{+\!\!\rhd} \mathit{Refund}^x$$



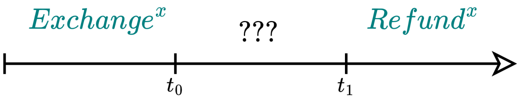
Example in BitML^x

~~Read the bibliography on sound cryptographic protocol designs.~~

Or let BitML^x do it for them!

$$\{A \;:!\, (1\text{฿}, 0\text{Ð}) \mid B \;:!\, (0\text{฿}, 1\text{Ð})\}\, Swap^x$$

$$Swap^x \;=\; Exchange^x \mathrel{+\!\!\rhd} Refund^x$$



$$
\begin{aligned}
Exchange^x \;= \;& \texttt{withdraw}(\\
& \quad (0\text{฿}, 1\text{Ð}) \to A, \\
& \quad (1\text{฿}, 0\text{Ð}) \to B \\
&)
\end{aligned}
$$

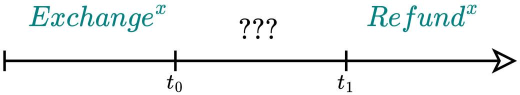
Example in BitML^x

~~Read the bibliography on sound cryptographic protocol designs.~~

Or let BitML^x do it for them!

$$\{A \; :! \, (1\text{฿}, 0\text{Ð}) \mid B \; :! \, (0\text{฿}, 1\text{Ð})\} Swap^x$$

$$Swap^x \; = Exchange^x + \!\!\rhd Refund^x$$



$$
\begin{aligned}
Exchange^x \; = \texttt{withdraw}(\\
(0\text{฿}, 1\text{Ð}) \to A, \\
(1\text{฿}, 0\text{Ð}) \to B \\
)
\end{aligned}
\qquad
\begin{aligned}
Refund^x \; = \texttt{withdraw}(\\
(1\text{฿}, 0\text{Ð}) \to A, \\
(0\text{฿}, 1\text{Ð}) \to B \\
)
\end{aligned}
$$

BitML^x compilation idea

But, does it work?

BitML^x compilation idea

But, does it work?

$$Swap^x \xrightarrow{???} \begin{cases} \vec{T}^x_{\text{₿}} \\ \vec{T}^x_{\text{Đ}} \end{cases}$$

BitML^x compilation idea

But, does it work?

$$Swap^x \xrightarrow{???} \begin{cases} \vec{T}^x_{฿} \\ \vec{T}^x_{Đ} \end{cases}$$

$$Swap^{฿} \xrightarrow{\text{BitML compiler}} \vec{T}_{฿}$$

$$Swap^{Đ} \xrightarrow{\text{BitML compiler}} \vec{T}_{Đ}$$

BitML^x compilation idea

But, does it work?

$$Swap^x \xrightarrow{???} \begin{cases} \vec{T}^x_{฿} \\ \vec{T}^x_{Đ} \end{cases}$$

$$Swap^{฿} \xrightarrow{\text{BitML compiler}} \vec{T}_{฿}$$

$$Swap^{Đ} \xrightarrow{\text{BitML compiler}} \vec{T}_{Đ}$$

$$Swap^x \xrightarrow{\text{BitML}^x\text{ compiler}} \begin{cases} Swap^x_{฿} \\ Swap^x_{Đ} \end{cases}$$

But, does it work?

$$Swap^x \xrightarrow{???} \begin{cases} \vec{T}^x_{₿} \\ \vec{T}^x_{Đ} \end{cases}$$

$$Swap^{₿} \xrightarrow{\text{BitML compiler}} \vec{T}_{₿}$$

$$Swap^{Đ} \xrightarrow{\text{BitML compiler}} \vec{T}_{Đ}$$

$$Swap^x \xrightarrow{\text{BitML}^x \text{ compiler}} \begin{cases} Swap^x_{₿} \xrightarrow{\text{BitML compiler}} \vec{T}^x_{₿} \\ Swap^x_{Đ} \xrightarrow{\text{BitML compiler}} \vec{T}^x_{Đ} \end{cases}$$

Compilation Properties

- Synchronicity: every time a choice is taken on one side, it's replicated on the other.

Compilation Properties

- Synchronicity: every time a choice is taken on one side, it's replicated on the other.
- Synchronicity is hard on the general case:

$$C_1 \not\Rightarrow C_2 \not\Rightarrow C_3 \not\Rightarrow \dots$$

Compilation Properties

- Synchronicity: every time a choice is taken on one side, it's replicated on the other.
- Synchronicity is hard on the general case:

$$C_1 \not\Rightarrow C_2 \not\Rightarrow C_3 \not\Rightarrow \ldots$$

- Fairness: malicious users cannot harm honest users.

Example compilation

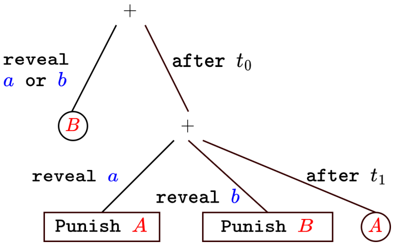
$\{\ A\ :!\ 1฿$
$\mid A:\ \texttt{secret}\ a$
$\mid B:\ \texttt{secret}\ b$
$\}\mathit{Swap}^{x}_{฿}$

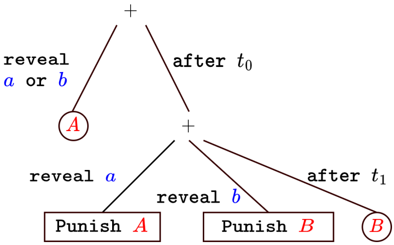
$\{\ B\ :!\ 1Đ$
$\mid A:\ \texttt{secret}\ a$
$\mid B:\ \texttt{secret}\ b$
$\}\mathit{Swap}^{x}_{Đ}$

Example compilation

$\{$ *A* :! 1₿
$\mid$ *A* : `secret` *a*
$\mid$ *B* : `secret` *b*
$\}$*Swap*$^{x}_{₿}$

$\{$ *B* :! 1Đ
$\mid$ *A* : `secret` *a*
$\mid$ *B* : `secret` *b*
$\}$*Swap*$^{x}_{Đ}$



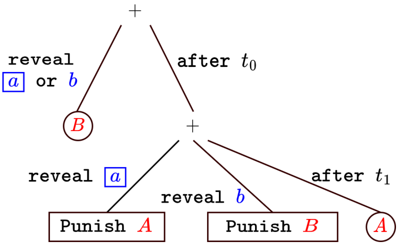


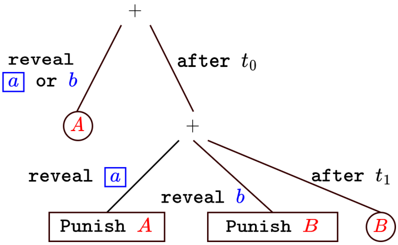
$$t_0 < t_1$$

Example compilation

$\{\ A :!\ 1$₿
$\mid A :\ \texttt{secret}\ \boxed{a}$
$\mid B :\ \texttt{secret}\ b$
$\}\mathit{Swap}^{x}$₿

$\{\ B :!\ 1$Ɖ
$\mid A :\ \texttt{secret}\ \boxed{a}$
$\mid B :\ \texttt{secret}\ b$
$\}\mathit{Swap}^{x}$Ɖ



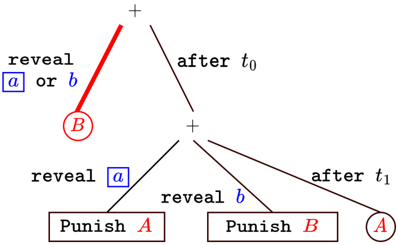


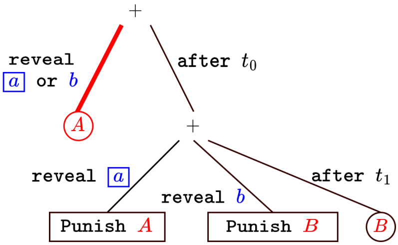
$$t_0 < t_1$$

Example compilation

{ A :! 1₿
| A : `secret` $\boxed{a}$
| B : `secret` b
}$Swap^{x}_{₿}$

{ B :! 1Đ
| A : `secret` $\boxed{a}$
| B : `secret` b
}$Swap^{x}_{Đ}$



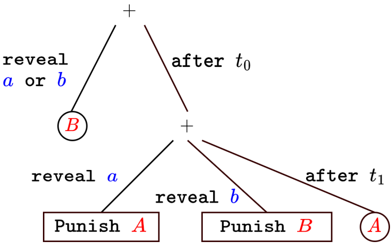


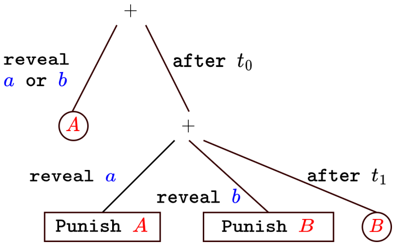
$$t_0 < t_1$$

Example compilation

{ A :! 1₿
| A : `secret` a
| B : `secret` b
}$Swap^{x}_{₿}$

{ B :! 1Ð
| A : `secret` a
| B : `secret` b
}$Swap^{x}_{Ð}$



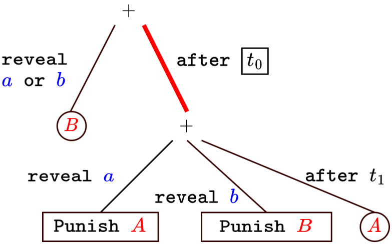


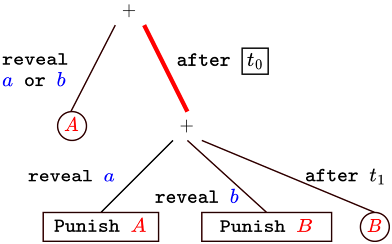
$$t_0 < t_1$$

Example compilation

$$\{\ A \ :!\ 1₿$$
$$\mid A :\ \texttt{secret}\ a$$
$$\mid B :\ \texttt{secret}\ b$$
$$\}\mathit{Swap}^{x}_{₿}$$

$$\{\ B \ :!\ 1Đ$$
$$\mid A :\ \texttt{secret}\ a$$
$$\mid B :\ \texttt{secret}\ b$$
$$\}\mathit{Swap}^{x}_{Đ}$$



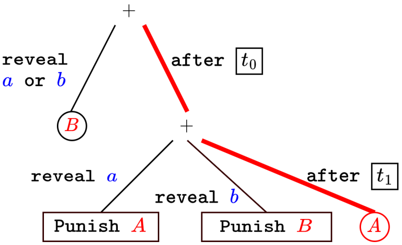


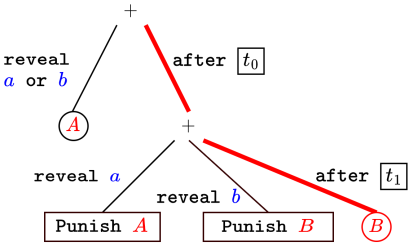
$$t_0 < t_1$$

Example compilation

{ A :! 1₿
| A : `secret` a
| B : `secret` b
}$Swap^{x}_{₿}$

{ B :! 1Đ
| A : `secret` a
| B : `secret` b
}$Swap^{x}_{Đ}$



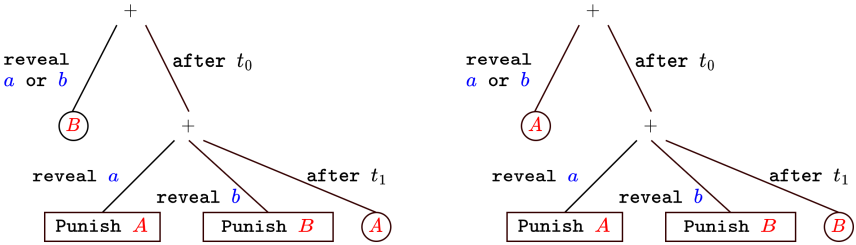


$$t_0 < t_1$$

Example compilation

{ A :! 1₿
| A : `secret` a
| B : `secret` b
}$Swap^{x}_{₿}$

{ B :! 1Đ
| A : `secret` a
| B : `secret` b
}$Swap^{x}_{Đ}$

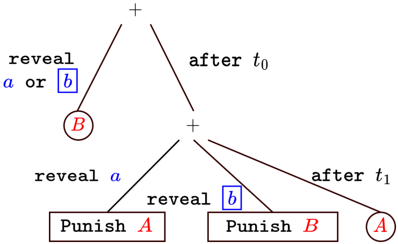


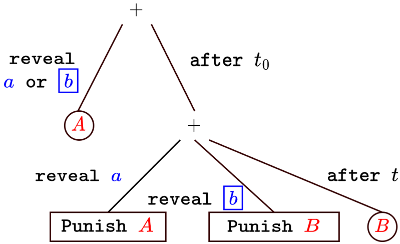
$$t_0 < t_1$$

Example compilation

$$\{\ A :!\ 1₿\ \mid A:\ \texttt{secret}\ a\ \mid B:\ \texttt{secret}\ \boxed{b}\ \}Swap^{x}_{₿}$$

$$\{\ B :!\ 1Đ\ \mid A:\ \texttt{secret}\ a\ \mid B:\ \texttt{secret}\ \boxed{b}\ \}Swap^{x}_{Đ}$$



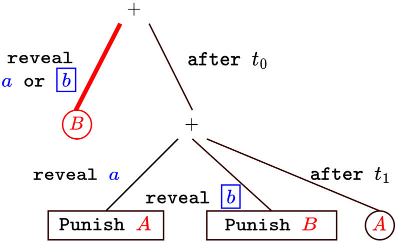


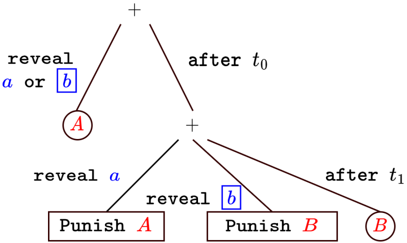
$$t_0 < t_1$$

Example compilation

$\{\ A\ :!\ 1₿$
$\mid A :\ \texttt{secret}\ a$
$\mid B :\ \texttt{secret}\ \boxed{b}$
$\}\mathit{Swap}^{x}_{₿}$

$\{\ B\ :!\ 1Đ$
$\mid A :\ \texttt{secret}\ a$
$\mid B :\ \texttt{secret}\ \boxed{b}$
$\}\mathit{Swap}^{x}_{Đ}$



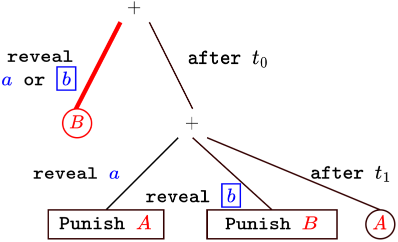


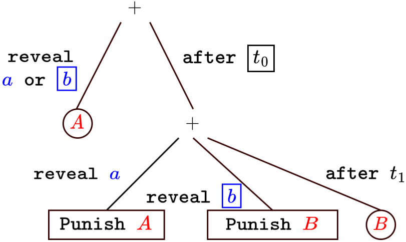
$$t_0 < t_1$$

Example compilation

$$\{\ A :!\ 1₿ \mid A : \texttt{secret}\ a \mid B : \texttt{secret}\ \boxed{b}\ \}\mathit{Swap}^{x}_{₿}$$

$$\{\ B :!\ 1Đ \mid A : \texttt{secret}\ a \mid B : \texttt{secret}\ \boxed{b}\ \}\mathit{Swap}^{x}_{Đ}$$



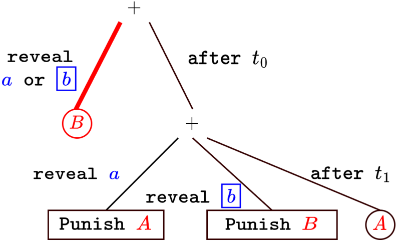


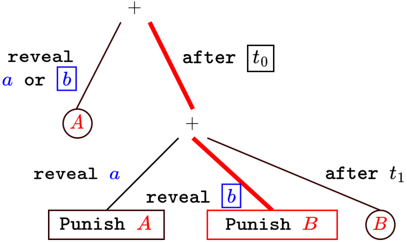
$$t_0 < t_1$$

Example compilation

$\{\ A\ :!\ 1₿$
$\mid A:\ \texttt{secret}\ a$
$\mid B:\ \texttt{secret}\ \boxed{b}$
$\}\mathit{Swap}^{x}_{₿}$

$\{\ B\ :!\ 1Đ$
$\mid A:\ \texttt{secret}\ a$
$\mid B:\ \texttt{secret}\ \boxed{b}$
$\}\mathit{Swap}^{x}_{Đ}$





$$t_0 < t_1$$

Collaterals

- How can we "punish" asynchronous behavior?

Collaterals

- How can we "punish" asynchronous behavior?
- Make them pay! But, how much?

Collaterals

- How can we "punish" asynchronous behavior?
- Make them pay! But, how much?
- Everyone locks in an extra collateral deposit c:

$$c = b \times (n - 2)$$

where b is the contract balance, an n is the number of participants.

Correctness

Theorem (Compiler correctness, informal)

Each strategy of an honest user A on a BitML^x contract C translates into a strategy on the concurrently executing compiled BitML contracts $C_{\text{₿}} \mid C_{\text{Ð}}$ that allows A to extract at least as many assets from $C_{\text{₿}} \mid C_{\text{Ð}}$ as from C with the original strategy.

Thanks!

- BitML^x allows you to model cross-blockchain smart contracts.
- It's compiled to concurrently executing BitML contracts.
- Security by mechanisms of timed commitments and punishments
- Work in Progress:
  - Proving BitML^x correctness.
  - Implementing compiler in Haskell.

Download short paper, slides and (soon) compiler:

Bonus: standard atomic swaps