Native Custom Tokens in The Extended UTXO Model

Manuel M.T. Chakravarty, James Chapman, Kenneth MacKenzie,
Orestis Melkonian, Jann Müller, Michael Peyton Jones, Polina Vinogradova, Philip Wadler

October 28, 2020

Introduction

Motivation

- Most Ethereum smart contracts manage *user-defined assets*
    - either *fungible tokens* based on ERC-20
    - or *non-fungible tokens* based on ERC-721

Motivation

- Most Ethereum smart contracts manage *user-defined assets*
    - either *fungible tokens* based on ERC-20
    - or *non-fungible tokens* based on ERC-721
- Unfortunately non-native, hence inefficient and expensive

This talk

1. Recall EUTXO, an extension of UTXO

This talk

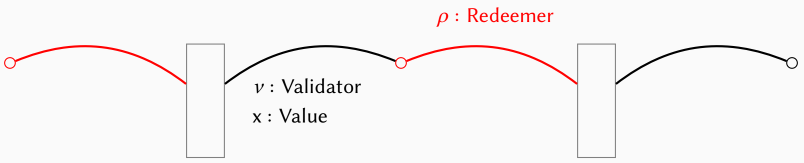
1. Recall EUTXO, an extension of UTXO
2. Introduce $\mathrm{EUTXO}_{\mathrm{ma}}$ to support **native custom tokens**

This talk

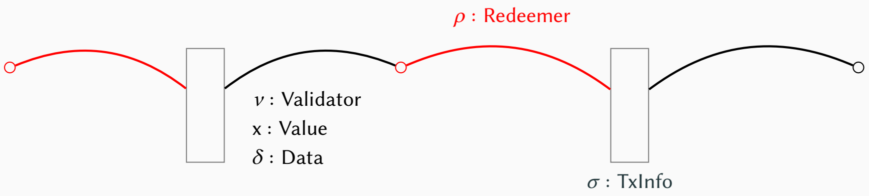
1. Recall EUTXO, an extension of UTXO
2. Introduce EUTXO$_{\mathrm{ma}}$to support **native custom tokens**
3. Utilise multi-currency features to extend the previous meta-theory and make it more robust

EUTXO

UTXO vs EUTXO

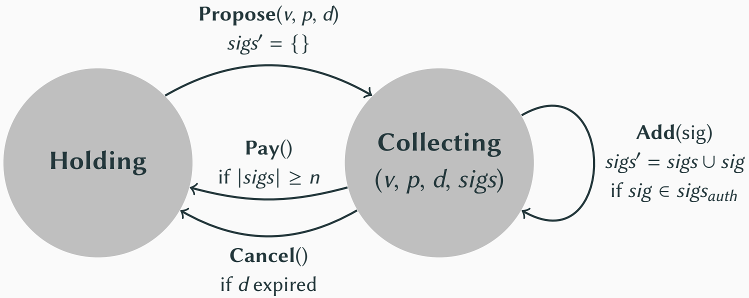


$$\nu(\rho) \overset{?}{=} \text{True}$$

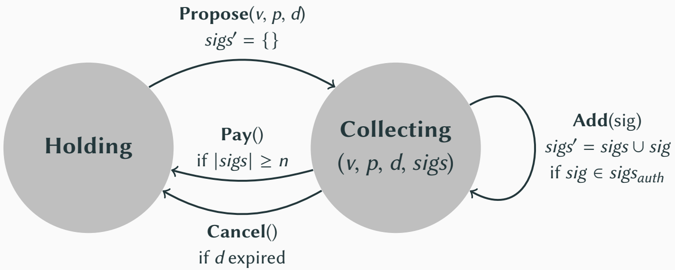


$$\nu(\rho,\ x,\ \delta,\ \sigma) \overset{?}{=} \text{True}$$

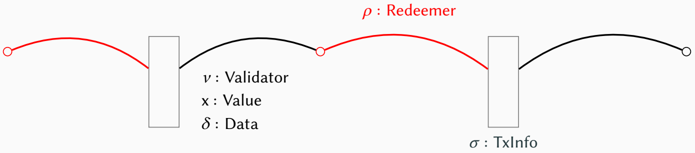
Running Example: Asynchronous Multi-signature Contract



Pay value (v) to payee (p) until deadline (d)



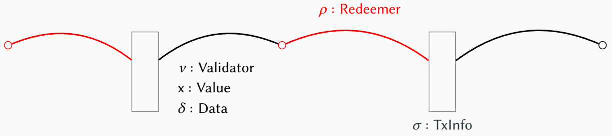
- $\delta \in \{\text{Holding}, \text{Collecting}\}$
- $\rho \in \{\text{Propose}, \text{Add}, \text{Cancel}, \text{Pay}\}$



$$v(\rho,\ x,\ \delta,\ \sigma) \stackrel{?}{=} \text{True}$$

Previous Work [The Extended UTXO Model @ WTSC'20]

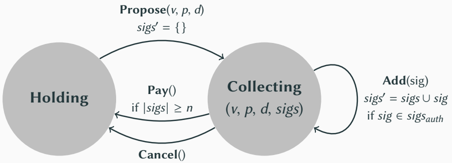
- **Detailed description of the Extended UTXO model (EUTXO)**



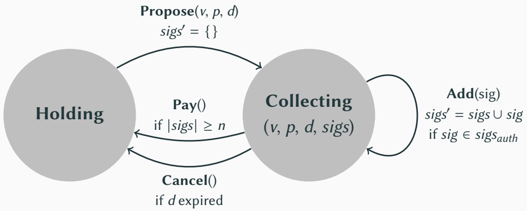
$$\nu(\rho,\ x,\ \delta,\ \sigma) \overset{?}{=} \text{True}$$

- **Formalization in** Agda

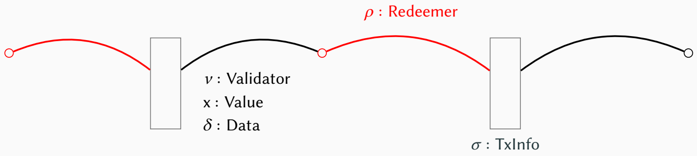
- **Proof of bisimulation with a Constraint Emitting Machines (CEMs)**



Limitation: Initial State



- $\delta \in \{\text{Holding, Collecting}\}$
- $\rho \in \{\text{Propose, Add, Cancel, Pay}\}$



$$\nu(\rho,\ x,\ \delta,\ \sigma) \overset{?}{=} \text{True}$$

EUTXO$_{\text{MA}}$

Token Bundles

$$\{₳ \mapsto \{₳ \mapsto 3\}, \mathit{WoW} \mapsto \{\mathit{sword} \mapsto 1, \mathit{shield} \mapsto 1\}\}$$

Operations on Token Bundles

$$
\begin{aligned}
& \{ ₳ \mapsto \{ ₳ \mapsto 3\}, \mathit{WoW} \mapsto \{\mathit{sword} \mapsto 1, \mathit{shield} \mapsto 1\}\} \\
+ & \{ ₳ \mapsto \{ ₳ \mapsto 1\}, \mathit{WoW} \mapsto \{\mathit{armour} \mapsto 1\}\} \\
= & \{ ₳ \mapsto \{ ₳ \mapsto 4\}, \mathit{WoW} \mapsto \{\mathit{sword} \mapsto 1, \mathit{shield} \mapsto 1, \mathit{armour} \mapsto 1\}\} \ .
\end{aligned}
$$

Forging

- $Tx = \{\ldots \mathit{forge} : \mathrm{TokenBundle}, \mathit{policies} : \sigma \longrightarrow \mathrm{Bool} \ldots\}$

Forging

- $Tx = \{\dots \mathit{forge} : \text{TokenBundle}, \mathit{policies} : \sigma \longrightarrow \text{Bool} \dots\}$
- $\mathit{forge} \in \mathbb{Z}^+$ for minting, $\mathit{forge} \in \mathbb{Z}^-$ for burning

Forging

- $Tx = \{\ldots forge : \text{TokenBundle}, policies : \sigma \to \text{Bool} \ldots\}$
- $forge \in \mathbb{Z}^+$ for minting, $forge \in \mathbb{Z}^-$ for burning
- $\forall\ p\sharp \in forge.domain : p \in policies \wedge p(\sigma) = \text{True}$

Applications

- Tokenised roles
- Fairness in ICO setup
- Algorithmic stablecoins

$\vdots$

Applications: Threaded State Machines

- Every CEM instance is associated with a unique token ♦

Applications: Threaded State Machines

- Every CEM instance is associated with a unique token ♦
- **CEM policy**: check ♦ is minted in an initial state

Applications: Threaded State Machines

- Every CEM instance is associated with a unique token ⧫
- **CEM policy**: check ⧫ is minted in an initial state
- **CEM validator:** check ⧫ is propagated at each transition

Applications: Threaded State Machines

- Every CEM instance is associated with a unique token ⧫
- **CEM policy:** check ⧫ is minted in an initial state
- **CEM validator:** check ⧫ is propagated at each transition

$\Longrightarrow$ can distinguish between different executions

Applications: Threaded State Machines

- Every CEM instance is associated with a unique token ⧫
- **CEM policy:** check ⧫ is minted in an initial state
- **CEM validator:** check ⧫ is propagated at each transition

  $\Longrightarrow$ can distinguish between different executions

  $\Longrightarrow$ solve the initialisation problem

Meta-theory

Token Provenance

- Token = Policy $\times$ Asset

Token Provenance

- Token = Policy $\times$ Asset
- Trace$(l, o, \blacklozenge, n) = t_0, \ldots, t_i, t_{i+1}, \ldots, t_k$, where
    1. $t_0.\mathit{forge}^{\blacklozenge} \geq n$
    2. $t_i \xrightarrow{\blacklozenge} t_{i+1} \geq n$
    3. $o \in t_k.\mathit{outputs}$

Token Provenance

- Token = Policy $\times$ Asset
- $\text{Trace}(l, o, \blacklozenge, n) = t_0, \ldots, t_i, t_{i+1}, \ldots, t_k$, where
  1. $t_0.\mathit{forge}^{\blacklozenge} \geq n$
  2. $t_i \xrightarrow{\blacklozenge} t_{i+1} \geq n$
  3. $o \in t_k.\mathit{outputs}$
- $\text{Provenance}(l, o, \blacklozenge) = \ldots \text{Trace}(l, o, \blacklozenge, n_i) \ldots \text{ s.t. } \sum n_i \geq o.\mathit{value}^{\blacklozenge}$

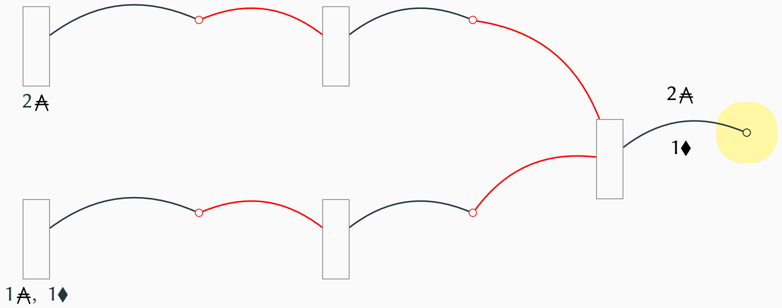
Token Provenance

- $\text{Token} = \text{Policy} \times \text{Asset}$
- $\text{Trace}(l, o, \blacklozenge, n) = t_0, \ldots, t_i, t_{i+1}, \ldots, t_k$, where
  1. $t_0.\mathit{forge}^{\blacklozenge} \geq n$
  2. $t_i \xrightarrow{\blacklozenge} t_{i+1} \geq n$
  3. $o \in t_k.\mathit{outputs}$
- $\text{Provenance}(l, o, \blacklozenge) = \ldots \text{Trace}(l, o, \blacklozenge, n_i) \ldots$ s.t. $\sum n_i \geq o.\mathit{value}^{\blacklozenge}$

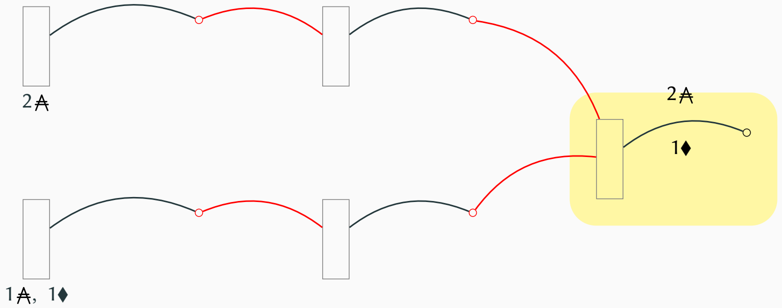
Every output has a provenance

$$\frac{o \in \{t.\mathit{outputs} \mid t \in l\}}{\text{provenance}(l, o, \blacklozenge) : \text{Provenance}(l, o, \blacklozenge)} \; \textsc{Provenance}$$

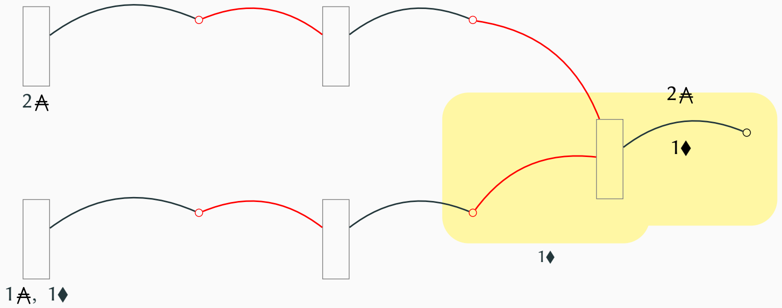
Example Traces



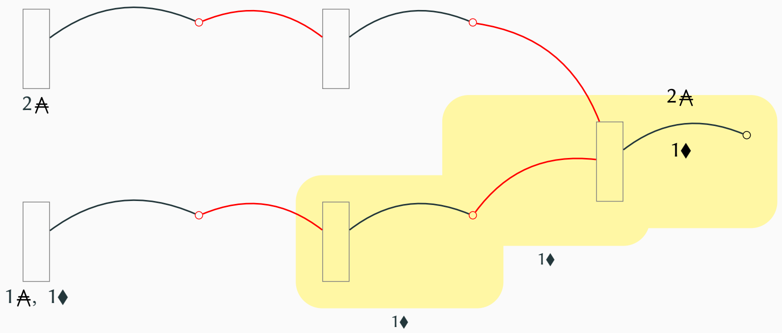
Example Traces



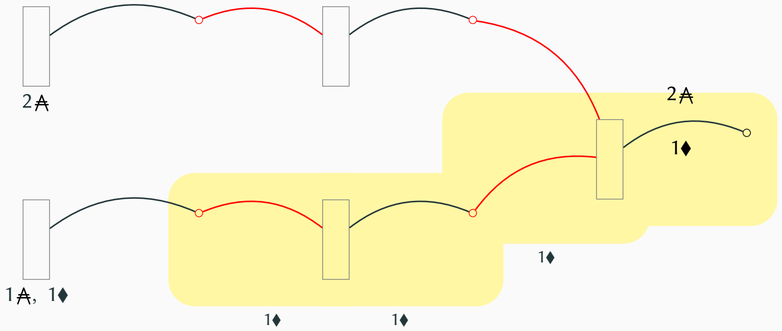
Example Traces



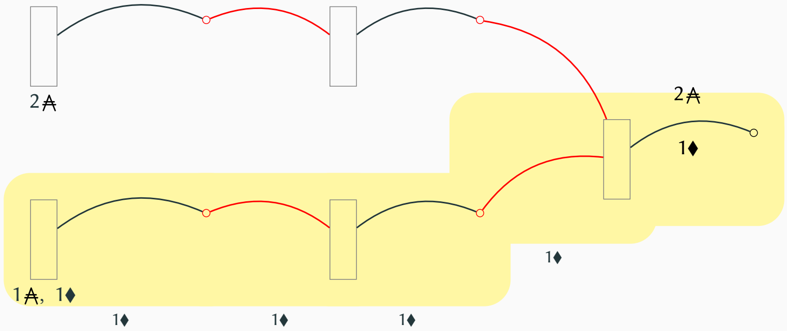
Example Traces



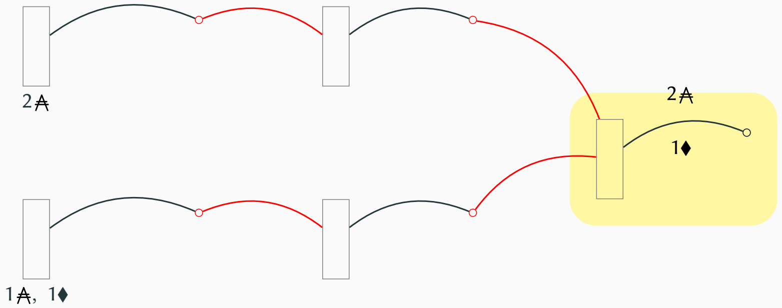
Example Traces



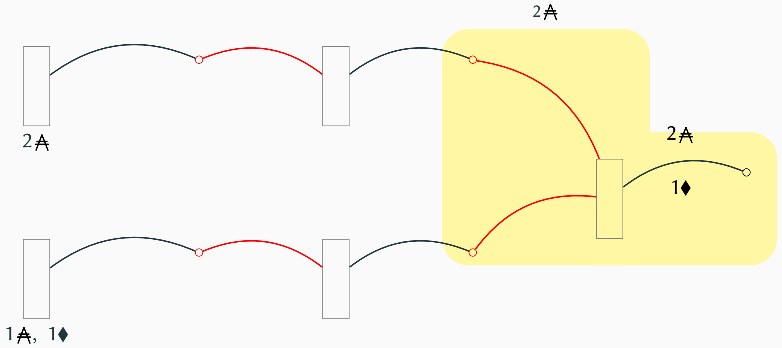
Example Traces



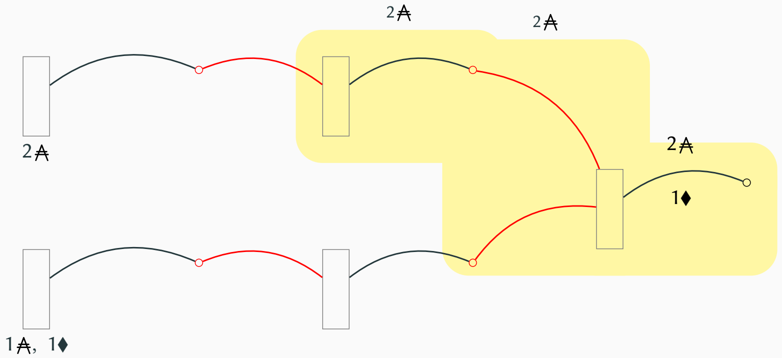
Example Traces



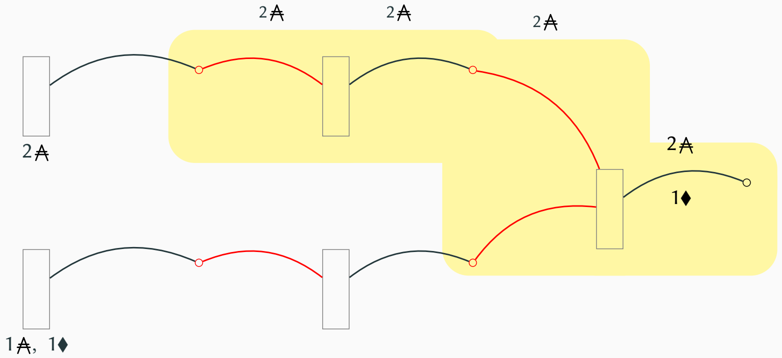
Example Traces



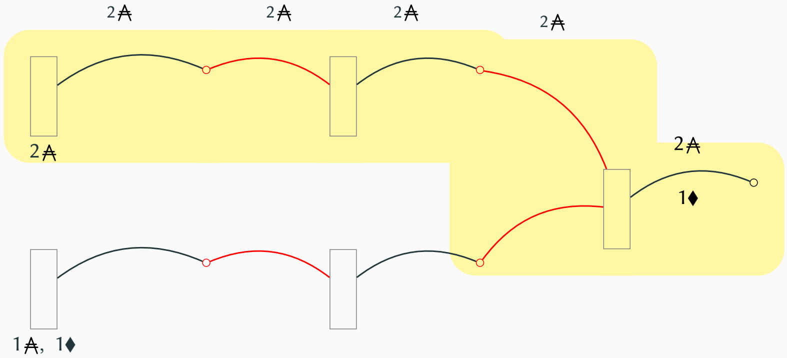
Example Traces



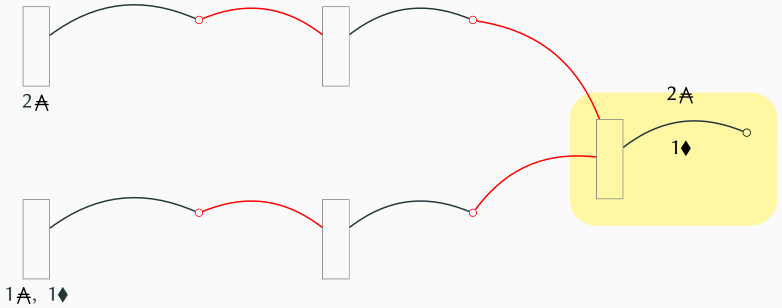
Example Traces



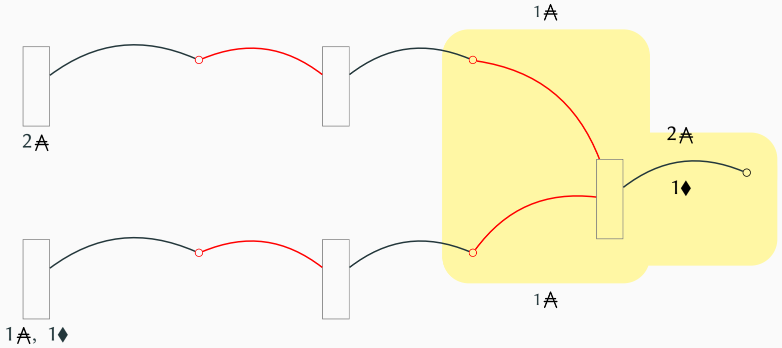
Example Traces



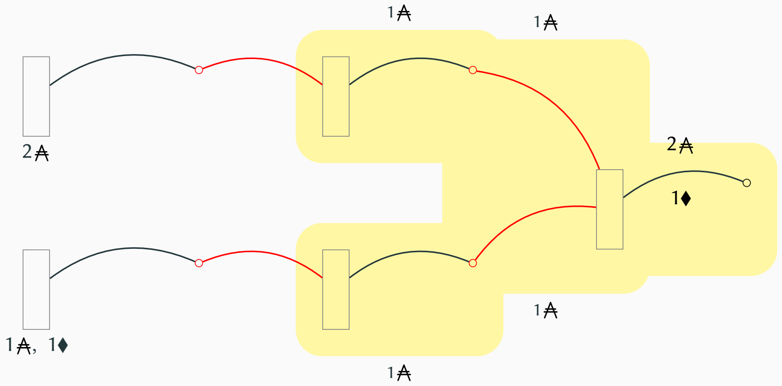
Example Traces



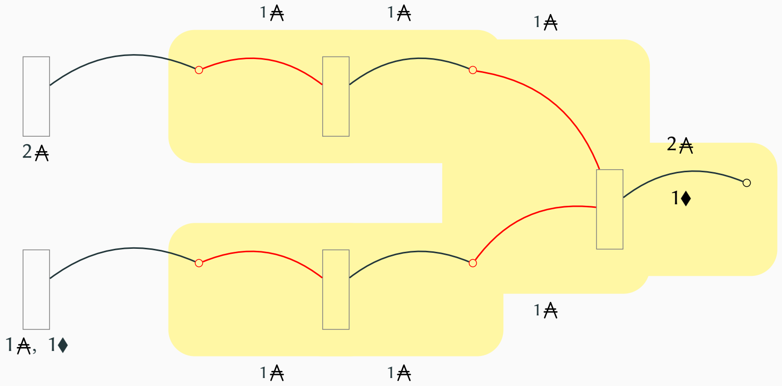
Example Traces



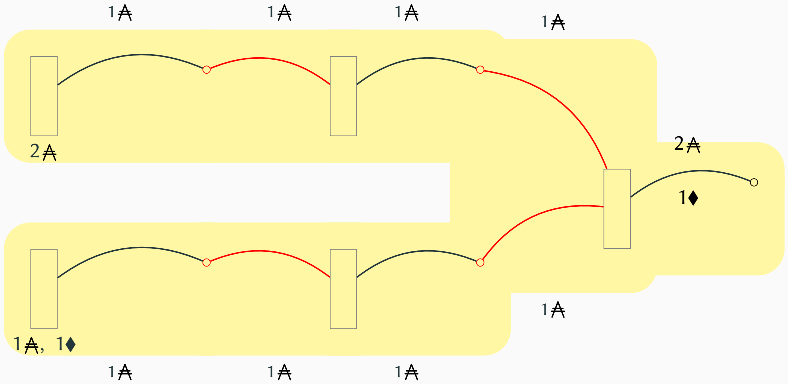
Example Traces



Example Traces



Example Traces



Forging Policies

Provenance is never empty

$$\frac{o \in \{t.outputs \mid t \in l\} \quad o.value^{\blacklozenge} > 0}{|\text{provenance}(l, o, \blacklozenge)| > 0}\ \textsc{Provenance}^{+}$$

Forging Policies

Provenance is never empty

$$\frac{o \in \{t.outputs \mid t \in l\} \quad o.value^{\blacklozenge} > 0}{|\mathrm{provenance}(l, o, \blacklozenge)| > 0}\ \textsc{Provenance}^{+}$$

Global Preservation

$$\sum_{t \in l} t.forge = \sum_{o \in \mathrm{unspentOutputs}(l)} o.value$$

Non-fungible Provenance

Provenance for non-fungible tokens

$$\frac{o \in \{t.outputs \mid t \in l\} \quad o.value^{\blacklozenge} > 0 \quad \sum_{t \in l} t.forge^{\blacklozenge} \leq 1}{|\mathrm{provenance}(l, o, \blacklozenge)| = 1}\ \textsc{NF-Provenance}$$

Threaded CEMs

$$\mathrm{policy}_{\mathcal{C}}(\mathit{txInfo}, c) = \begin{cases} \mathrm{true} & \textit{if } \mathit{txInfo}.\mathit{forge}^{\blacklozenge} = 1 \\ & \textit{and } \mathrm{origin} \in \mathit{txInfo}.\mathit{outputRefs} \\ & \textit{and } \mathrm{initial}(\mathit{txInfo}.\mathit{outputs}^{\blacklozenge}) \\ \mathrm{false} & \textit{otherwise} \end{cases}$$

Threaded CEMs

$$\text{policy}_{\mathcal{C}}(\mathit{txInfo}, c) = \begin{cases} \text{true} & \textit{if } \mathit{txInfo.forge}^{\blacklozenge} = 1 \\ & \textit{and } \text{origin} \in \mathit{txInfo.outputRefs} \\ & \textit{and } \text{initial}(\mathit{txInfo.outputs}^{\blacklozenge}) \\ \text{false} & \textit{otherwise} \end{cases}$$

$$\text{validator}_{\mathcal{C}}(s, i, \mathit{txInfo}) = \begin{cases} \text{true} & \textit{if } s \xrightarrow{i} (s', tx^{\equiv}) \\ & \textit{and } \text{satisfies}(\mathit{txInfo}, tx^{\equiv}) \\ & \textit{and } \text{checkOutputs}(s', \mathit{txInfo}) \\ & \textit{and } \text{propagates}(\mathit{txInfo}, \blacklozenge, s, s') \\ \text{false} & \textit{otherwise} \end{cases}$$

Threaded CEMs: Initiality

All traces originate from initial states

$$\frac{o \in \{t.outputs \mid t \in l\} \quad o.value^{\blacklozenge} > 0}{\exists tr.\ \text{provenance}(l, o, \blacklozenge) = \{tr\} \ \wedge\ \text{policy}_{\mathcal{C}}(tr_0.\text{context}) = \text{true}}\ \textsc{Initiality}$$

Property Preservation

Well-rooted sequences

$$\frac{\text{initial}(s) = \text{true}}{s \rightsquigarrow^* s} \qquad \frac{s \rightsquigarrow^* s' \quad s' \xrightarrow{i} (s'', tx^{\equiv})}{s \rightsquigarrow^* s''}$$

Property Preservation

Well-rooted sequences

$$\frac{\text{initial}(s) = \text{true}}{s \rightsquigarrow^* s} \qquad \frac{s \rightsquigarrow^* s' \quad s' \xrightarrow{i} (s'', tx^{\equiv})}{s \rightsquigarrow^* s''}$$

A property P is invariant when:

$$\frac{\text{initial} \implies P \qquad \forall (s \rightsquigarrow^* s').P(s) \implies P(s')}{\texttt{invariant}\ P}$$

Example: Counter CEM

$$(\mathbb{Z}, \{\mathrm{inc}\}, \mathrm{step}, \mathrm{initial}) \textbf{ where } \mathrm{step}(i, \mathrm{inc}) = \mathrm{just}(i + 1);\ \ \mathrm{initial}(0) = \mathrm{true}$$

Example: Counter CEM

$$(\mathbb{Z}, \{\mathrm{inc}\}, \mathrm{step}, \mathrm{initial}) \textbf{ where } \mathrm{step}(i, \mathrm{inc}) = \mathrm{just}(i + 1);\ \ \mathrm{initial}(0) = \mathrm{true}$$

- $Q = (\_ >= 0)$ is invariant, since $Q(0)$ and $Q(x) \implies Q(x + 1)$

Example: Counter CEM

$$(\mathbb{Z}, \{\mathrm{inc}\}, \mathrm{step}, \mathrm{initial})\ \textbf{where}\ \mathrm{step}(i, \mathrm{inc}) = \mathrm{just}(i+1);\ \ \mathrm{initial}(0) = \mathrm{true}$$

- $Q = (\_ >= 0)$ is invariant, since $Q(0)$ and $Q(x) \implies Q(x+1)$

Extracting CEM traces from EUTXO traces

$$\frac{\mathrm{provenance}(l, o, \blacklozenge) = \{tr\}}{tr.source \rightsquigarrow^{*}\ tr.destination}\ \textsc{Extraction}$$

Beyond Safety

- so far only safety properties

Beyond Safety

- so far only safety properties
- what about temporal ones?

Beyond Safety

- so far only safety properties
- what about temporal ones?

$$\overset{?}{\Longrightarrow}$$ *coinductive* techniques for *infinitary* semantics

Related Work

- Scilla [Sergey et al. @ OOPSLA'19]
    - Ethereum smart contracts as state machines
    - Embed in Coq and prove safety & temporal properties

Related Work

- Scilla [Sergey et al. @ OOPSLA'19]
  - Ethereum smart contracts as state machines
  - Embed in Coq and prove safety & temporal properties
- Bitcoin Modelling Language (BitML) [Bartoletti et al. @ CCS'18]
  - High-level process calculus > Bitcoin transactions
  - Small-step operational semantics   state machines
  - Support for LTL formulas
  - No (complete) mechanization yet

Related Work

- Scilla [Sergey et al. @ OOPSLA'19]
    - Ethereum smart contracts as state machines
    - Embed in Coq and prove safety & temporal properties
- Bitcoin Modelling Language (BitML) [Bartoletti et al. @ CCS'18]
    - High-level process calculus > Bitcoin transactions
    - Small-step operational semantics   state machines
    - Support for LTL formulas
    - No (complete) mechanization yet
- VeriSolid [Mavridou et al. @ FC'20]
    - Solidity contracts as state machines
    - Support for CTL formulas
    - No mechanized meta-theory

This talk

1. Recall EUTXO, an extension of UTXO

This talk

1. Recall EUTXO, an extension of UTXO
2. Introduce $\text{EUTXO}_{\text{ma}}$to support **native custom tokens**

This talk

1. Recall EUTXO, an extension of UTXO
2. Introduce EUTXO$_{\text{ma}}$to support **native custom tokens**
3. Utilise multi-currency features to extend the previous meta-theory and make it more robust

Questions?