# A true concurrent model of smart contracts executions

Massimo Bartoletti[1], Letterio Galletta[2], and Maurizio Murgia[3]

[1] University of Cagliari, Italy
[2] IMT School for Advanced Studies, Lucca, Italy
[3] University of Trento, Italy

**Abstract.** The development of blockchain technologies has enabled the trustless execution of so-called *smart contracts*, i.e. programs that regulate the exchange of assets (e.g., cryptocurrency) between users. In a decentralized blockchain, the state of smart contracts is collaboratively maintained by a peer-to-peer network of mutually untrusted nodes, which collect from users a set of *transactions* (representing the required actions on contracts), and execute them in some order. Once this sequence of transactions is appended to the blockchain, the other nodes validate it, re-executing the transactions in the same order. The serial execution of transactions does not take advantage of the multi-core architecture of modern processors, so contributing to limit the throughput. In this paper we propose a true concurrent model of smart contracts execution. Based on this, we show how static analysis of smart contracts can be exploited to parallelize the execution of transactions.

## 1 Introduction

Smart contracts [20] are computer programs that transfer digital assets between users without a trusted authority. Currently, smart contracts are supported by several blockchains, the first and most widespread one being Ethereum [8]. Users interact with a smart contract by sending *transactions*, which trigger state updates, and may possibly involve transfers of crypto-assets between the called contract and the users. The sequence of transactions on the blockchain determines the state of each contract, and the balance of each user.

The blockchain is maintained by a peer-to-peer network of nodes, which follow a consensus protocol to determine, at each turn, a new block of transactions to be added to the blockchain. This protocol guarantees the correct execution of contracts also in the presence of (a minority of) adversaries in the network, and ensures that all the nodes have the same view of their state. Nodes play the role of *miner* or that of *validator*. Miners gather from the network sets of transactions sent by users, and execute them *serially* to determine the new state. Once a block is appended to the blockchain, validators re-execute all its transactions, to update their local view of the contracts state and of the users' balance. To do this, validators process the transactions exactly in the same order in which they occur in the block, since choosing a different order could potentially result in

inconsistencies between the nodes (note that miners also act as validators, since they validate all the blocks received from the network).

Although executing transactions in a purely sequential fashion is quite effective to ensure the consistency of the blockchain state, in the age of multi-core processors it fails to properly exploit the computational capabilities of nodes. By enabling miners and validators to concurrently execute transactions, it would be possible to improve the efficiency and the throughput of the blockchain.

This paper exploits techniques from concurrency theory to provide a formal backbone for parallel executions of transactions. More specifically, our main contributions can be summarised as follows:

– As a first step, we formalise blockchains, giving their semantics as a function which maps each contract to its state, and each user to her balance. This semantics reflects the standard implementation of nodes, where transactions are evaluated in sequence, without any concurrency.
– We introduce two notions of *swappability* of transactions. The first is purely semantic: two adjacent transactions can be swapped if doing so preserves the semantics of the blockchain. The second notion, called *strong* swappability, is more syntactical: it checks a simple condition (inspired by Bernstein's conditions [6]) on static approximations of the variables read/written by the transactions. Theorem 2 shows that strong swappability is strictly included in the semantic relation. Further, if we transform a blockchain by repeatedly exchanging adjacent strongly swappable transactions, the resulting blockchain is observationally equivalent to the original one (Theorem 4).
– Building upon strong swappability, we devise a true concurrent model of transactions execution. To this purpose, we transform a block of transactions into an *occurrence net*, describing exactly the partial order induced by the swappability relation. We model the concurrent executions of a blockchain in terms of the *step firing sequences* (i.e. finite sequences of *sets* of transitions) of the associated occurrence net. Theorem 5 establishes that the concurrent executions and the serial one are semantically equivalent.
– We describe how miners and validators can use our results to concurrently execute transactions, exploiting the multi-core architecture available on their nodes. Remarkably, our technique is compatible with the current implementation of the Ethereum blockchain, while the other existing approaches to parallelize transactions execution would require a soft-fork.
– We apply our technique to ERC-721 tokens, one of the most common kinds of contracts in Ethereum, showing them to be suitable for parallelization.

The proofs of our results are in Appendix A.


## 2 Transactions and blockchains

In this section we introduce a general model of transactions and blockchains, abstracting from the actual smart contracts language.

A smart contract is a finite set of functions, i.e. terms of the form $\mathtt{f}(\boldsymbol{x})\{S\}$, where $\mathtt{f}$ is a function name, $\boldsymbol{x}$ is the sequence of formal parameters (omitted when empty), and $S$ is the function body. We postulate that the functions in a contract have distinct names. We abstract from the actual syntax of $S$, and we just assume that the semantics of function bodies is defined (see e.g. [5] for a concrete instance of syntax and semantics of function bodies).

Let **Val** be a set of *values*, ranged over by $v, v', \ldots$, let **Const** be a set of *constant names* $x, y, \ldots$, and let **Addr** be a set of *addresses* $\mathcal{X}, \mathcal{Y}, \ldots$, partitioned into *account addresses* $\mathcal{A}, \mathcal{B}, \ldots$ and *contract addresses* $\mathcal{C}, \mathcal{D}, \ldots$. We assume a mapping $\Gamma$ from addresses to contracts.

We assume that each contract has a key-value store, which we render as a partial function **Val** $\rightharpoonup$ **Val** from keys $k \in$ **Val** to values. The state of the blockchain is a function $\sigma : \mathbf{Addr} \to (\mathbf{Val} \rightharpoonup \mathbf{Val})$ from addresses to key-value stores. We postulate that $\mathtt{balance} \in \operatorname{dom} \sigma \mathcal{X}$ for all $\mathcal{X}$. A *qualified key* is a term of the form $\mathcal{X}.k$. We write $\sigma(\mathcal{X}.k)$ for $\sigma \mathcal{X} k$; when $k \notin \operatorname{dom} \sigma \mathcal{X}$, we write $\sigma(\mathcal{X}.k) = \bot$. We use $p, q, \ldots$ to range over qualified keys, $P, Q, \ldots$ to range over sets of them, and $\mathbb{P}$ to denote the set of all qualified keys.

To have a uniform treatment of accounts and contracts, we assume that for all account addresses $\mathcal{A}$, $\operatorname{dom} \sigma \mathcal{A} = \{\mathtt{balance}\}$, and that the contract $\Gamma(\mathcal{A})$ has exactly one function, which just skips. In this way, the statement $\mathcal{A}.\mathtt{transfer}(n)$, which transfers $n$ currency units to $\mathcal{A}$, can be rendered as a call to this function.

*State updates* define how values associated with qualified keys are modified.

**Definition 1 (State update).** *A state update $\pi : \mathbf{Addr} \rightharpoonup (\mathbf{Val} \rightharpoonup \mathbf{Val})$ is a function from qualified keys to values; we denote with $\{v/\mathcal{X}.k\}$ the state update which maps $\mathcal{X}.k$ to $v$. We define $\operatorname{keys}(\pi)$ as the set of qualified keys $\mathcal{X}.k$ such that $\mathcal{X} \in \operatorname{dom} \pi$ and $k \in \operatorname{dom} \pi \mathcal{X}$. We apply updates to states as follows:*

$$(\sigma\pi)\mathcal{X} = \delta_{\mathcal{X}} \quad \text{where} \quad \delta_{\mathcal{X}} k = \begin{cases} \pi\mathcal{X}k & \text{if } \mathcal{X}.k \in \operatorname{keys}(\pi) \\ \sigma\mathcal{X}k & \text{otherwise} \end{cases} \qquad \diamond$$

We denote with $[\![S]\!]^{\mathcal{X}}_{\sigma,\rho}$ the semantics of the statement $S$. This semantics is either a blockchain state $\sigma'$, or it is undefined (denoted by $\bot$). The semantics is parameterised over a state $\sigma$, an address $\mathcal{X}$ (the contract wherein $S$ is evaluated), and an *environment* $\rho : \mathbf{Const} \rightharpoonup \mathbf{Val}$, used to evaluate the formal parameters and the special names $\mathtt{sender}$ and $\mathtt{value}$. These names represent, respectively, the caller of the function, and the amount of currency transferred along with the call. We postulate that $\mathtt{sender}$ and $\mathtt{value}$ are not used as formal parameters.

We define the auxiliary operators $+$ and $-$ on states as follows:

$$\sigma \circ (\mathcal{X} : n) \;=\; \sigma\{(\sigma\mathcal{X}\mathtt{balance}) \circ n / \mathcal{X}.\mathtt{balance}\} \qquad\qquad (\circ \in \{+, -\})$$

i.e., $\sigma + \mathcal{X} : n$ updates $\sigma$ by increasing the $\mathtt{balance}$ of $\mathcal{X}$ of $n$ currency units.

A *transaction* $\mathsf{T}$ is a term of the form:

$$\mathcal{A} \xrightarrow{n} \mathcal{C} : \mathtt{f}(\boldsymbol{v})$$

$$\dfrac{\begin{array}{c} \mathtt{f}(\boldsymbol{x})\{S\} \in \varGamma(\mathcal{C}) \\[2pt] \sigma\,\mathcal{A}\,\mathtt{balance} \geq n \\[2pt] [\![S]\!]^{\mathcal{C}}_{\sigma - \mathcal{A}:n + \mathcal{C}:n,\,\{{}^{\mathcal{A}}\!/\mathtt{sender},\,{}^{n}\!/\mathtt{value},\,\boldsymbol{v}/\boldsymbol{x}\}} = \sigma' \end{array}}{[\![\mathcal{A} \xrightarrow{n} \mathcal{C} : \mathtt{f}(\boldsymbol{v})]\!]_\sigma = \sigma'}\,{}_{[\text{Tx1}]} \qquad \dfrac{\begin{array}{c} \mathtt{f}(\boldsymbol{x})\{S\} \in \varGamma(\mathcal{C}) \\[2pt] \left(\begin{array}{c}\sigma\,\mathcal{A}\,\mathtt{balance} < n \qquad \text{or} \\[2pt] [\![S]\!]^{\mathcal{C}}_{\sigma - \mathcal{A}:n + \mathcal{C}:n,\,\{{}^{\mathcal{A}}\!/\mathtt{sender},\,{}^{n}\!/\mathtt{value},\,\boldsymbol{v}/\boldsymbol{x}\}} = \bot\end{array}\right) \end{array}}{[\![\mathcal{A} \xrightarrow{n} \mathcal{C} : \mathtt{f}(\boldsymbol{v})]\!]_\sigma = \sigma}\,{}_{[\text{Tx2}]}$$

Fig. 1: Semantics of transactions.

Intuitively, $\mathcal{A}$ is the address of the caller, $\mathcal{C}$ is the address of the called contract, $\mathtt{f}$ is the called function, $n$ is the value transferred from $\mathcal{A}$ to $\mathcal{C}$, and $\boldsymbol{v}$ is the sequence of actual parameters. We denote the semantics of $\mathsf{T}$ in $\sigma$ as $[\![\mathsf{T}]\!]_\sigma$, where the function $[\![\cdot]\!]_\sigma$ is defined in Figure 1, which we briefly comment.

The semantics of a transaction $\mathsf{T} = \mathcal{A} \xrightarrow{n} \mathcal{C} : \mathtt{f}(\boldsymbol{v})$, in a given blockchain state $\sigma$, is a new state $\sigma'$. Rule [Tx1] handles the case where the transaction is successful: this happens when $\mathcal{A}$'s balance is at least $n$, and the function call terminates in a non-error state. Note that $n$ units of currency are transferred to $\mathcal{C}$ *before* starting to execute $\mathtt{f}$, and that the names $\mathtt{sender}$ and $\mathtt{value}$ are bound, respectively, to $\mathcal{A}$ and $n$. Rule [Tx2] applies either when $\mathcal{A}$'s balance is not enough, or the execution of $\mathtt{f}$ fails. In these cases, $\mathsf{T}$ does not alter the state.

A *blockchain* $\mathbf{B}$ is a finite sequence of transactions; we denote with $\epsilon$ the empty blockchain. The semantics of a blockchain is obtained by folding the semantics of its transactions, starting from a given state $\sigma$:

$$[\![\epsilon]\!]_\sigma = \sigma \qquad [\![\mathsf{T}\mathbf{B}]\!]_\sigma = [\![\mathbf{B}]\!]_{[\![\mathsf{T}]\!]_\sigma}$$

Note that erroneous transactions can occur within a blockchain, but they have no effect on its semantics (as rule [Tx2] makes them identities w.r.t. the append operation). We assume that in the initial state of the blockchain, denoted by $\sigma^\star$, each address $\mathcal{X}$ has a balance $n^\star_{\mathcal{X}} \geq 0$, while all the other keys are unbound. We write $[\![\mathbf{B}]\!]$ for $[\![\mathbf{B}]\!]_{\sigma^\star}$, where $\sigma^\star \mathcal{X} = \{{}^{n^\star_{\mathcal{X}}}\!/\mathtt{balance}\}$. We say that a state $\sigma$ is *reachable* if $\sigma = [\![\mathbf{B}]\!]$ for some $\mathbf{B}$.

*Example 1.* Consider the following functions of a contract at address $\mathcal{C}$:

$$\mathtt{f}_0()\{x := 1\} \qquad \mathtt{f}_1()\{\mathtt{if}\ x = 0\ \mathtt{then}\ \mathcal{B}.\mathtt{transfer}(1)\} \qquad \mathtt{f}_2()\{\mathcal{B}.\mathtt{transfer}(1)\}$$

Let $\sigma$ be a state such that $\sigma \mathcal{A}\,\mathtt{balance} \geq 2$, and let $\mathbf{B} = \mathsf{T}_0\mathsf{T}_1\mathsf{T}_2$, where:

$$\mathsf{T}_0 = \mathcal{A} \xrightarrow{0} \mathcal{C} : \mathtt{f}_0() \qquad \mathsf{T}_1 = \mathcal{A} \xrightarrow{1} \mathcal{C} : \mathtt{f}_1() \qquad \mathsf{T}_2 = \mathcal{A} \xrightarrow{1} \mathcal{C} : \mathtt{f}_2()$$

By applying rule [Tx1] three times, we have that:

$$[\![\mathsf{T}_0]\!]_\sigma = [\![x := 1]\!]^{\mathcal{C}}_{\sigma,\,\{{}^{\mathcal{A}}\!/\mathtt{sender},\,{}^{0}\!/\mathtt{value}\}} = \sigma\{{}^{1}\!/\mathcal{C}.x\} = \sigma'$$

$$[\![\mathsf{T}_1]\!]_{\sigma'} = [\![\mathtt{if}\ x = 0\ \mathtt{then}\ \mathcal{B}.\mathtt{transfer}(1)]\!]^{\mathcal{C}}_{\sigma' - \mathcal{A}:1 + \mathcal{C}:1,\,\{{}^{\mathcal{A}}\!/\mathtt{sender},\,{}^{1}\!/\mathtt{value}\}}$$

$$\qquad\qquad = \sigma' - \mathcal{A} : 1 + \mathcal{C} : 1 = \sigma''$$

$$[\![\mathsf{T}_2]\!]_{\sigma''} = [\![\mathcal{B}.\mathtt{transfer}(1)]\!]^{\mathcal{C}}_{\sigma'' - \mathcal{A}:1 + \mathcal{C}:1,\,\{{}^{\mathcal{A}}\!/\mathtt{sender},\,{}^{1}\!/\mathtt{value}\}} = \sigma'' - \mathcal{A} : 1 + \mathcal{B} : 1$$

Summing up, $[\![\mathbf{B}]\!]_\sigma = \sigma\{{}^{1}\!/\mathcal{C}.x\} - \mathcal{A} : 2 + \mathcal{B} : 1 + \mathcal{C} : 1$. $\qquad\qquad\diamond$

## 3  Swapping transactions

We define two blockchain states to be *observationally equivalent* when they agree on the values associated to all the qualified keys. Our formalisation is parameterised on a set of qualified keys $P$ over which we require the agreement.

**Definition 2 (Observational equivalence).** *For all $P \subseteq \mathbb{P}$, we define $\sigma \sim_P \sigma'$ iff $\forall p \in P : \sigma p = \sigma' p$. We say that $\sigma$ and $\sigma'$ are* observationally equivalent, *in symbols $\sigma \sim \sigma'$, when $\sigma \sim_P \sigma'$ holds for all $P$.* $\diamond$

**Lemma 1.** *For all $P, Q \subseteq \mathbb{P}$: (i) $\sim_P$ is an equivalence relation; (ii) if $\sigma \sim_P \sigma'$ and $Q \subseteq P$, then $\sigma \sim_Q \sigma'$; (iii) $\sim \, = \, \sim_{\mathbb{P}}$.* $\diamond$

We extend the equivalence relations above to blockchains, by passing through their semantics. For all $P$, we define $\mathbf{B} \sim_P \mathbf{B'}$ iff $[\![\mathbf{B}]\!]_\sigma \sim_P [\![\mathbf{B'}]\!]_\sigma$ holds for all reachable $\sigma$ (note that all the definitions and results in this paper apply to reachable states, since the unreachable ones do not represent actual contract executions). We write $\mathbf{B} \sim \mathbf{B'}$ when $\mathbf{B} \sim_P \mathbf{B'}$ holds for all $P$. The relation $\sim$ is a *congruence* with respect to the append operation, i.e. if $\mathbf{B} \sim \mathbf{B'}$ then we can replace $\mathbf{B}$ with $\mathbf{B'}$ in a larger blockchain, preserving its semantics.

**Lemma 2.** $\mathbf{B} \sim \mathbf{B'} \implies \forall \mathbf{B}_0, \mathbf{B}_1 : \mathbf{B}_0 \mathbf{B} \mathbf{B}_1 \sim \mathbf{B}_0 \mathbf{B'} \mathbf{B}_1$. $\diamond$

Two transactions are *swappable* when exchanging their order preserves observational equivalence.

**Definition 3 (Swappability).** *Two transactions $\mathsf{T} \neq \mathsf{T'}$ are* swappable, *in symbols $\mathsf{T} \rightleftarrows \mathsf{T'}$, when $\mathsf{T}\mathsf{T'} \sim \mathsf{T'}\mathsf{T}$.* $\diamond$

*Example 2.* Recall the transactions in Example 1. We have that $\mathsf{T}_0 \rightleftarrows \mathsf{T}_2$ and $\mathsf{T}_1 \rightleftarrows \mathsf{T}_2$, but $\mathsf{T}_0 \not\rightleftarrows \mathsf{T}_1$ (see Figure 5 in Appendix A). $\diamond$

We shall use the theory of trace languages originated from Mazurkiewicz's works [16] to study observational equivalence under various swapping relations. Below, we fix the alphabet of trace languages as the set $\mathbf{Tx}$ of all transactions.

**Definition 4 (Mazurkiewicz equivalence).** *Let $I$ be a symmetric and irreflexive relation on $\mathbf{Tx}$. The* Mazurkiewicz equivalence $\simeq_I$ *is the least congruence in the free monoid $\mathbf{Tx}^*$ such that: $\forall \mathsf{T}, \mathsf{T'} \in \mathbf{Tx}$: $\mathsf{T}\, I\, \mathsf{T'} \implies \mathsf{T}\mathsf{T'} \simeq_I \mathsf{T'}\mathsf{T}$.*

Theorem 1 below states that the Mazurkiewicz equivalence constructed on the swappability relation $\rightleftarrows$ is an observational equivalence. Therefore, we can transform a blockchain into an observationally equivalent one by a finite number of exchanges of adjacent swappable transactions.

**Theorem 1.** $\simeq_{\rightleftarrows} \, \subseteq \, \sim$. $\diamond$

*Example 3.* We can rearrange the transactions in Example 1 as $\mathsf{T}_0\mathsf{T}_1\mathsf{T}_2 \sim \mathsf{T}_0\mathsf{T}_2\mathsf{T}_1 \sim \mathsf{T}_2\mathsf{T}_0\mathsf{T}_1$. Instead, $\mathsf{T}_1\mathsf{T}_0\mathsf{T}_2 \not\sim \mathsf{T}_2\mathsf{T}_0\mathsf{T}_1$ (e.g., starting from a state $\sigma$ such that $\sigma\mathcal{A}\,\mathtt{balance} = 2$ and $\sigma\mathcal{C}x = 0$, see Figure 6 in Appendix A). $\diamond$

Note that the converse of Theorem 1 does not hold: indeed, $\mathbf{B} \simeq_{\rightleftarrows} \mathbf{B'}$ requires that $\mathbf{B}$ and $\mathbf{B'}$ have the same length, while $\mathbf{B} \sim \mathbf{B'}$ may also hold for blockchains of different length (e.g., $\mathbf{B'} = \mathbf{B}\mathsf{T}$, where $\mathsf{T}$ does not alter the state).

**Safe approximations of read/written keys**  Note that the relation $\rightleftarrows$ is undecidable whenever the contract language is Turing-equivalent. So, to detect swappable transactions we follow a static approach, consisting of two steps. First, we over-approximate the set of keys read and written by transactions, by statically analysing the code of the called functions. We then check a simple condition on these approximations (Definition 7), to detect if two transactions can be swapped. Since static analyses to over-approximate read and written variables are quite standard [17], here we just rely on such approximations, by only assuming their safety. In Definition 5 we state that a set $P$ safely approximates the keys *written* by $\mathsf{T}$, when $\mathsf{T}$ does not alter the state of the keys not in $P$. Defining set of *read* keys is a bit trickier: intuitively, we require that if we execute the transaction starting from two states that agree on the values of the keys in the read set, then these executions should be equivalent, in the sense that they do not introduce new differences between the resulting states (with respect to the difference already existing before).

**Definition 5 (Safe approximation of read/written keys).** *Given a set of qualified keys $P$ and a transaction $\mathsf{T}$, we define:*

$$P \models^w \mathsf{T} \qquad iff \qquad \forall Q : Q \cap P = \emptyset \implies \mathsf{T} \sim_Q \epsilon$$
$$P \models^r \mathsf{T} \qquad iff \qquad \forall \mathbf{B}, \mathbf{B}', Q : \mathbf{B} \sim_P \mathbf{B}' \wedge \mathbf{B} \sim_Q \mathbf{B}' \implies \mathbf{B}\mathsf{T} \sim_Q \mathbf{B}'\mathsf{T} \qquad \diamond$$

*Example 4.* Let $\mathsf{T} = \mathcal{A} \xrightarrow{1} \mathcal{C} : \mathtt{f}()$, where $\mathtt{f}()\{\mathcal{B}.\mathtt{transfer}(1)\}$ is a function of $\mathcal{C}$. The execution of $\mathsf{T}$ affects the $\mathtt{balance}$ of $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$; however, $\mathcal{C}.\mathtt{balance}$ is first incremented and then decremented, and so its value remains unchanged. Then, $\{\mathcal{A}.\mathtt{balance}, \mathcal{B}.\mathtt{balance}\} \models^w \mathsf{T}$, and it is the smallest safe approximation of the keys written by $\mathsf{T}$. To prove that $P = \{\mathcal{A}.\mathtt{balance}\} \models^r \mathsf{T}$, assume two blockchains $\mathbf{B}$ and $\mathbf{B}'$ and a set of keys $Q$ such that $\mathbf{B} \sim_P \mathbf{B}'$ and $\mathbf{B} \sim_Q \mathbf{B}'$. If $[\![\mathbf{B}]\!]\mathcal{A}\mathtt{balance} < 1$, then by [Tx2] we have $[\![\mathbf{B}\mathsf{T}]\!] = [\![\mathbf{B}]\!]$. Since $\mathbf{B} \sim_P \mathbf{B}'$, then also $[\![\mathbf{B}']\!]\mathcal{A}\mathtt{balance} < 1$, and so by [Tx2] we have $[\![\mathbf{B}'\mathsf{T}]\!] = [\![\mathbf{B}']\!]$. Then, $\mathbf{B}\mathsf{T} \sim_Q \mathbf{B}'\mathsf{T}$. Otherwise, if $[\![\mathbf{B}]\!]\mathcal{A}\mathtt{balance} = n \geq 1$, then by [Tx1] the execution of $\mathsf{T}$ transfers one unit of currency from $\mathcal{A}$ to $\mathcal{B}$, so the execution of $\mathsf{T}$ affects exactly $\mathcal{A}.\mathtt{balance}$ and $\mathcal{B}.\mathtt{balance}$. So, it is enough to show that $\mathbf{B} \sim_{\{q\}} \mathbf{B}'$ implies $\mathbf{B}\mathsf{T} \sim_{\{q\}} \mathbf{B}'\mathsf{T}$ for $q \in \{\mathcal{A}.\mathtt{balance}, \mathcal{B}.\mathtt{balance}\}$. For $q = \mathcal{A}.\mathtt{balance}$, we have that $[\![\mathbf{B}'\mathsf{T}]\!]\mathcal{A}\mathtt{balance} = n - 1 = [\![\mathbf{B}\mathsf{T}]\!]\mathcal{A}\mathtt{balance}$. For $q = \mathcal{B}.\mathtt{balance}$, we have that $[\![\mathbf{B}'\mathsf{T}]\!]\mathcal{B}\mathtt{balance} = [\![\mathbf{B}']\!]\mathcal{B}\mathtt{balance} + 1 = [\![\mathbf{B}]\!]\mathcal{B}\mathtt{balance} + 1 = [\![\mathbf{B}\mathsf{T}]\!]\mathcal{B}\mathtt{balance}$. Therefore, we conclude that $P \models^r \mathsf{T}$. $\diamond$

Widening a safe approximation (either of read or written keys) preserves its safety; further, the intersection of two safe write approximations is still safe (see Lemma 6 in Appendix A). From this, it follows that there exists a *least* safe approximation of the keys written by a transaction.

**Strong swappability**  We use safe approximations of the read/written keys to detect when two transactions are swappable. To achieve that, we check whether two transactions $\mathsf{T}$ and $\mathsf{T}'$ operate on disjoint portions of the blockchain state.

More specifically, we recast in our setting Bernstein's conditions [6] for the parallel execution of processes: it suffices to check that the set of keys written by $\mathsf{T}$ is disjoint from those written or read by $\mathsf{T}'$, and vice versa. When this happens we say that the two transactions are *strongly swappable*.

**Definition 6 (Strong swappability).** *We say that two transactions $\mathsf{T} \neq \mathsf{T}'$ are* strongly swappable, *in symbols $\mathsf{T}\#\mathsf{T}'$, when there exist $W, W', R, R' \subseteq \mathbb{P}$ such that $W \models^w \mathsf{T}$, $W' \models^w \mathsf{T}'$, $R \models^r \mathsf{T}$, $R' \models^r \mathsf{T}'$, and:*

$$(R \cup W) \cap W' \;=\; \emptyset \;=\; (R' \cup W') \cap W \qquad\qquad \diamond$$

*Example 5.* Let $\mathtt{f}_1()\{\mathtt{skip}\}$ and $\mathtt{f}_2(x)\{x.\mathtt{transfer}(\mathtt{value})\}$ be functions of the contracts $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, and consider the following transactions:

$$\mathsf{T}_1 = \mathcal{A} \xrightarrow{1} \mathcal{C}_1 : \mathtt{f}_1() \qquad\qquad \mathsf{T}_2 = \mathcal{B} \xrightarrow{1} \mathcal{C}_2 : \mathtt{f}_2(\mathcal{F})$$

where $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{F}$ are account addresses. To prove that $\mathsf{T}_1\#\mathsf{T}_2$, consider the following safe approximations of the written/read keys of $\mathsf{T}_1$ and $\mathsf{T}_2$, respectively:

$$W_1 = \{\mathcal{A}.\mathtt{balance}, \mathcal{C}_1.\mathtt{balance}\} \models^w \mathsf{T}_1 \qquad R_1 = \{\mathcal{A}.\mathtt{balance}\} \models^r \mathsf{T}_1$$
$$W_2 = \{\mathcal{B}.\mathtt{balance}, \mathcal{F}.\mathtt{balance}\} \models^w \mathsf{T}_2 \qquad R_2 = \{\mathcal{B}.\mathtt{balance}\} \models^r \mathsf{T}_2$$

Since $(W_1 \cup R_1) \cap W_2 = \emptyset = (W_2 \cup R_2) \cap W_1$, the two transactions are strongly swappable. Now, let:

$$\mathsf{T}_3 = \mathcal{B} \xrightarrow{1} \mathcal{C}_2 : \mathtt{f}_2(\mathcal{A})$$

and consider the following safe approximations $W_3$ and $R_3$:

$$W_3 = \{\mathcal{B}.\mathtt{balance}, \mathcal{A}.\mathtt{balance}\} \models^w \mathsf{T}_3 \qquad R_3 = \{\mathcal{B}.\mathtt{balance}\} \models^r \mathsf{T}_3$$

Since $W_1 \cap W_3 \neq \emptyset \neq W_2 \cap W_3$, then $\neg(\mathsf{T}_1\#\mathsf{T}_3)$ and $\neg(\mathsf{T}_2\#\mathsf{T}_3)$. $\qquad \diamond$

The following theorem ensures the soundness of our approximation, i.e. that if two transactions are strongly swappable, then they are also swappable. The converse implication does not hold, as witnessed by Example 6.

**Theorem 2.** $\mathsf{T}\#\mathsf{T}' \implies \mathsf{T} \rightleftarrows \mathsf{T}'$. $\qquad\qquad \diamond$

*Example 6 (Swappable transactions, not strongly).* Consider the following functions and transactions of a contract at address $\mathcal{C}$:

$$\mathtt{f}_1()\{\mathtt{if}\,\mathtt{sender} = \mathcal{A}\,\&\&\,k_1 = 0\,\mathtt{then}\,k_1{:=}1\,\mathtt{else}\,\mathtt{throw}\} \quad \mathsf{T}_1 = \mathcal{A} \xrightarrow{1} \mathcal{C} : \mathtt{f}_1()$$
$$\mathtt{f}_2()\{\mathtt{if}\,\mathtt{sender} = \mathcal{B}\,\&\&\,k_2 = 0\,\mathtt{then}\,k_2{:=}1\,\mathtt{else}\,\mathtt{throw}\} \quad \mathsf{T}_2 = \mathcal{B} \xrightarrow{1} \mathcal{C} : \mathtt{f}_2()$$

We prove that $\mathsf{T}_1 \rightleftarrows \mathsf{T}_2$. First, consider a state $\sigma$ such that $\sigma\mathcal{A}\,\mathtt{balance} > 1$, $\sigma\mathcal{B}\,\mathtt{balance} > 1$, $\sigma\mathcal{C}\,\mathtt{balance} = n$, $\sigma\mathcal{C}k_1 = 0$ and $\sigma\mathcal{C}k_2 = 0$. We have that:

$$[\![\mathsf{T}_1\mathsf{T}_2]\!]_\sigma = \sigma\{1/\mathcal{C}.k_1, 1/\mathcal{C}.k_2, n{+}2/\mathcal{C}.\mathtt{balance}\} = [\![\mathsf{T}_2\mathsf{T}_1]\!]_\sigma$$

In the second case, let $\sigma$ be such that $\sigma\mathcal{A}\mathtt{balance} < 1$, or $\sigma\mathcal{B}\mathtt{balance} < 1$, or $\sigma\mathcal{C}k_1 \neq 0$, or $\sigma\mathcal{C}k_2 \neq 0$. It is not possible that the guards in $\mathtt{f}_1$ and $\mathtt{f}_2$ are both true, so $\mathsf{T}_1$ or $\mathsf{T}_2$ raise an exception, leaving the state unaffected. Then, also in this case we have that $[\![\mathsf{T}_1\mathsf{T}_2]\!]_\sigma = [\![\mathsf{T}_2\mathsf{T}_1]\!]_\sigma$, and so $\mathsf{T}_1$ and $\mathsf{T}_2$ are swappable. However, they are *not* strongly swappable if there exist reachable states $\sigma, \sigma'$ such that $\sigma\mathcal{C}k_1 = 0 = \sigma'\mathcal{C}k_2$. To see why, let $W_1 = \{\mathcal{A}.\mathtt{balance}, \mathcal{C}.\mathtt{balance}, \mathcal{C}.k_1\}$. From the code of $\mathtt{f}_0$ we see that $W_1$ is the least safe over-approximation of the written keys of $\mathsf{T}_1$ ($W_1 \models^w \mathsf{T}_1$). This means that every safe approximation of $\mathsf{T}_1$ must include the keys of $W_1$. Similarly, $W_2 = \{\mathcal{B}.\mathtt{balance}, \mathcal{C}.\mathtt{balance}, \mathcal{C}.k_2\}$ is the least safe over-approximation of the written keys of $\mathsf{T}_2$ ($W_2 \models^w \mathsf{T}_2$). Since the least safe approximations of the keys written by $\mathsf{T}_1$ and $\mathsf{T}_2$ are not disjoint, $\mathsf{T}_1\#\mathsf{T}_2$ does not hold. $\qquad\diamond$

Theorem 3 states that the Mazurkiewicz equivalence $\simeq_\#$ is stricter than $\simeq_\rightleftarrows$. Together with Theorem 1, if $\mathbf{B}$ is transformed into $\mathbf{B'}$ by exchanging adjacent strongly swappable transactions, then $\mathbf{B}$ and $\mathbf{B'}$ are observationally equivalent.

**Theorem 3.** $\simeq_\# \subseteq \simeq_\rightleftarrows$. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\diamond$

Note that if the contract language is Turing-equivalent, then finding approximations which satisfy the disjointness condition in Definition 6 is not computable, and so the relation $\#$ is undecidable.

**Parameterised strong swappability** Strongly swappability abstracts from the actual static analysis used to obtain the safe approximations: it is sufficient that such an analysis exists. Definition 7 below parameterises strong swappability over a static analysis, which we represent as a function from transactions to sets of qualified keys, just requiring it to be a safe approximation. Formally, we say that $W$ is a *static analysis of written keys* when $W(\mathsf{T}) \models^w \mathsf{T}$, for all $\mathsf{T}$; similarly, $R$ is a *static analysis of read keys* when $R(\mathsf{T}) \models^r \mathsf{T}$, for all $\mathsf{T}$.

**Definition 7 (Parameterised strong swappability).** *Let $W$ and $R$ be static analyses of written/read keys. We say that $\mathsf{T}$, $\mathsf{T}'$ are* strongly swappable w.r.t. $W$ and $R$*, in symbols $\mathsf{T}\#_R^W\mathsf{T}'$, if:*

$$\big(R(\mathsf{T}) \cup W(\mathsf{T})\big) \cap W(\mathsf{T}') = \emptyset = \big(R(\mathsf{T}') \cup W(\mathsf{T}')\big) \cap W(\mathsf{T}) \qquad \diamond$$

Note that an effective procedure for computing $W$ and $R$ gives an effective procedure to determine whether two transactions are (strongly) swappable.

**Lemma 3.** *For all static analyses $W$ and $R$: (i) $\#_R^W \subseteq \#$; (ii) if $W$ and $R$ are computable, then $\#_R^W$ is decidable.* $\qquad\qquad\qquad\qquad\qquad\qquad\diamond$

From the inclusion in item (i) of Lemma 3 and from Theorem 3 we obtain:

**Theorem 4.** $\simeq_{\#_R^W} \subseteq \simeq_\# \subseteq \simeq_\rightleftarrows$. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\diamond$

# 4 True concurrency for blockchains

Given a swappability relation $\mathcal{R}$, we transform a sequence of transactions **B** into an *occurrence net* $\mathsf{N}_{\mathcal{R}}(\mathbf{B})$, which describes the partial order induced by $\mathcal{R}$. Any concurrent execution of the transactions in **B** which respects this partial order is equivalent to the serial execution of **B** (Theorem 5).

**From blockchains to occurrence nets** We start by recapping the notion of Petri net [18]. A *Petri net* is a tuple $\mathsf{N} = (\mathsf{P}, \mathsf{Tr}, \mathsf{F}, \mathsf{m}_0)$, where $\mathsf{P}$ is a set of *places*, $\mathsf{Tr}$ is a set of *transitions* (with $\mathsf{P} \cap \mathsf{Tr} = \emptyset$), and $\mathsf{F} : (\mathsf{P} \times \mathsf{Tr}) \cup (\mathsf{Tr} \times \mathsf{P}) \to \mathbb{N}$ is a *weight function*. The state of a net is a *marking*, i.e. a multiset $\mathsf{m} : \mathsf{P} \to \mathbb{N}$ defining how many *tokens* are contained in each place; we denote with $\mathsf{m}_0$ the initial marking. The behaviour of a Petri net is specified as a transition relation between markings: intuitively, a transition $\mathsf{t}$ is enabled at $\mathsf{m}$ when each place $\mathsf{p}$ has at least $\mathsf{F}(\mathsf{p}, \mathsf{t})$ tokens in $\mathsf{m}$. When an enabled transition $\mathsf{t}$ is fired, it consumes $\mathsf{F}(\mathsf{p}, \mathsf{t})$ tokens from each $\mathsf{p}$, and produces $\mathsf{F}(\mathsf{t}, \mathsf{p}')$ tokens in each $\mathsf{p}'$. Formally, given $x \in \mathsf{P} \cup \mathsf{Tr}$, we define the *preset* ${}^{\bullet}x$ and the *postset* $x^{\bullet}$ as multisets: ${}^{\bullet}x(y) = \mathsf{F}(y, x)$, and $x^{\bullet}(y) = \mathsf{F}(x, y)$. A transition $\mathsf{t}$ is *enabled* at $\mathsf{m}$ when ${}^{\bullet}\mathsf{t} \subseteq \mathsf{m}$. The transition relation between markings is defined as $\mathsf{m} \xrightarrow{\mathsf{t}} \mathsf{m} - {}^{\bullet}\mathsf{t} + \mathsf{t}^{\bullet}$, where $\mathsf{t}$ is enabled. We say that $\mathsf{t}_1 \cdots \mathsf{t}_n$ is a *firing sequence from $\mathsf{m}$ to $\mathsf{m}'$* when $\mathsf{m} \xrightarrow{\mathsf{t}_1} \cdots \xrightarrow{\mathsf{t}_n} \mathsf{m}'$, and in this case we say that $\mathsf{m}'$ is *reachable from* $\mathsf{m}$. We say that $\mathsf{m}'$ is *reachable* when it is reachable from $\mathsf{m}_0$.

An *occurrence net* [7] is a Petri net such that: (i) $|\mathsf{p}^{\bullet}| \leq 1$ for all $\mathsf{p}$; (ii) $|{}^{\bullet}\mathsf{p}| = 1$ if $\mathsf{p} \notin \mathsf{m}_0$, and $|{}^{\bullet}\mathsf{p}| = 0$ if $\mathsf{p} \in \mathsf{m}_0$; (iii) $\mathsf{F}$ is a relation, i.e. $\mathsf{F}(x, y) \leq 1$ for all $x, y$; (iv) $\mathsf{F}^*$ is a acyclic, i.e. $\forall x, y \in \mathsf{P} \cup \mathsf{Tr} : (x, y) \in \mathsf{F}^* \wedge (y, x) \in \mathsf{F}^* \implies x = y$ (where $\mathsf{F}^*$ is the reflexive and transitive closure of $\mathsf{F}$).

In Figure 2 we transform a blockchain $\mathbf{B} = \mathsf{T}_1 \cdots \mathsf{T}_n$ into a Petri net $\mathsf{N}_{\mathcal{R}}(\mathbf{B})$, where $\mathcal{R}$ is an arbitrary relation between transactions. Although any relation $\mathcal{R}$ ensures that $\mathsf{N}_{\mathcal{R}}(\mathbf{B})$ is an occurrence net (Lemma 4 below), our main results hold when $\mathcal{R}$ is a strong swappability relation. The transformation works as follows: the $i$-th transaction in **B** is rendered as a transition $(\mathsf{T}_i, i)$ in $\mathsf{N}_{\mathcal{R}}(\mathbf{B})$, and transactions related by $\mathcal{R}$ are transformed into concurrent transitions. Technically, this concurrency is specified as a relation $<$ between transitions, such that $(\mathsf{T}_i, i) < (\mathsf{T}_j, j)$ whenever $i < j$, but $\mathsf{T}_i$ and $\mathsf{T}_j$ are not related by $\mathcal{R}$. The places, the weight function, and the initial marking of $\mathsf{N}_{\mathcal{R}}(\mathbf{B})$ are chosen to ensure that the firing ot transitions respects the relation $<$.

*Example 7.* Consider the following transactions and functions of a contract $\mathcal{C}$:

$$\mathsf{T}_{\mathtt{f}} = \mathcal{A} \xrightarrow{0} \mathcal{C} : \mathtt{f}() \qquad \mathtt{f}() \,\{\mathtt{if}\ x = 0\ \mathtt{then}\ y \mathtt{:=} 1\ \mathtt{else\ throw}\}$$

$$\mathsf{T}_{\mathtt{g}} = \mathcal{A} \xrightarrow{0} \mathcal{C} : \mathtt{g}() \qquad \mathtt{g}() \,\{\mathtt{if}\ y = 0\ \mathtt{then}\ x \mathtt{:=} 1\ \mathtt{else\ throw}\}$$

$$\mathsf{T}_{\mathtt{h}} = \mathcal{A} \xrightarrow{0} \mathcal{C} : \mathtt{h}() \qquad \mathtt{h}() \,\{z \mathtt{:=} 1\}$$

Let $P_{\mathtt{f}}^w = P_{\mathtt{g}}^r = \{\mathcal{C}.y\}, P_{\mathtt{f}}^r = P_{\mathtt{g}}^w = \{\mathcal{C}.x\}, P_{\mathtt{h}}^w = \{\mathcal{C}.z\}, P_{\mathtt{h}}^r = \emptyset$. It is easy to check that these sets are safe approximations of their transactions (e.g., $P_{\mathtt{f}}^w$ safely

$$\mathsf{Tr} = \{(\mathsf{T}_i, i) \mid 1 \le i \le n\}$$

$$\mathsf{P} = \{(*, \mathsf{t}) \mid \mathsf{t} \in \mathsf{Tr}\} \cup \{(\mathsf{t}, *) \mid \mathsf{t} \in \mathsf{Tr}\} \cup \{(\mathsf{t}, \mathsf{t}') \mid \mathsf{t} < \mathsf{t}'\}$$

$$\text{where } (\mathsf{T}, i) < (\mathsf{T}', j) \triangleq (i < j) \wedge \neg(\mathsf{T} \,\mathcal{R}\, \mathsf{T}')$$

$$\mathsf{F}(x, y) = \begin{cases} 1 & \text{if } y = \mathsf{t} \text{ and } \big(x = (*, \mathsf{t}) \text{ or } x = (\mathsf{t}', \mathsf{t})\big) \\ 1 & \text{if } x = \mathsf{t} \text{ and } \big(y = (\mathsf{t}, *) \text{ or } y = (\mathsf{t}, \mathsf{t}')\big) \\ 0 & \text{otherwise} \end{cases} \qquad \mathsf{m}_0(\mathsf{p}) = \begin{cases} 1 & \text{if } \mathsf{p} = (*, \mathsf{t}) \\ 0 & \text{otherwise} \end{cases}$$

Fig. 2: Construction of a Petri net from a blockchain $\mathbf{B} = \mathsf{T}_1 \cdots \mathsf{T}_n$.
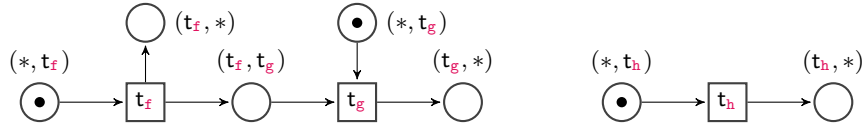


Fig. 3: Occurrence net for Example 7.

approximates the keys written by $\mathsf{T}_\mathsf{f}$). By Definition 6 we have that $\mathsf{T}_\mathsf{f} \# \mathsf{T}_\mathsf{h}$, $\mathsf{T}_\mathsf{g} \# \mathsf{T}_\mathsf{h}$, but $\neg(\mathsf{T}_\mathsf{f} \# \mathsf{T}_\mathsf{g})$. We display $\mathsf{N}_\#(\mathsf{T}_\mathsf{f} \mathsf{T}_\mathsf{h} \mathsf{T}_\mathsf{g})$ in Figure 3, where $\mathsf{t}_\mathsf{f} = (\mathsf{T}_\mathsf{f}, 1)$, $\mathsf{t}_\mathsf{h} = (\mathsf{T}_\mathsf{h}, 2)$, and $\mathsf{t}_\mathsf{g} = (\mathsf{T}_\mathsf{g}, 3)$. Note that $\mathsf{t}_\mathsf{g}$ can only be fired after $\mathsf{t}_\mathsf{f}$, while $\mathsf{t}_\mathsf{h}$ can be fired independently from $\mathsf{t}_\mathsf{f}$ and $\mathsf{t}_\mathsf{g}$. This is coherent with the fact that $\mathsf{T}_\mathsf{h}$ is swappable with both $\mathsf{T}_\mathsf{f}$ and $\mathsf{T}_\mathsf{g}$, while $\mathsf{T}_\mathsf{f}$ and $\mathsf{T}_\mathsf{g}$ are not swappable. $\qquad \diamond$

**Lemma 4.** $\mathsf{N}_\mathcal{R}(\mathbf{B})$ *is an occurrence net, for all $\mathcal{R}$ and $\mathbf{B}$.*

**Step firing sequences** Theorem 5 below establishes a correspondence between concurrent and serial execution of transactions. Since the semantics of serial executions is given in terms of blockchain states $\sigma$, to formalise this correspondence we use the same semantics domain also for concurrent executions. This is obtained in two steps. First, we define concurrent executions of $\mathbf{B}$ as the *step firing sequences* (i.e. finite sequences of *sets* of transitions) of $\mathsf{N}_\#(\mathbf{B})$. Then, we give a semantics to step firing sequences, in terms of blockchain states.

We denote finite sets of transitions, called *steps*, as $\mathsf{U}, \mathsf{U}', \dots$. Their preset and postset are defined as ${}^\bullet\mathsf{U} = \sum_{\mathsf{p} \in \mathsf{U}} {}^\bullet\mathsf{p}$ and $\mathsf{U}^\bullet = \sum_{\mathsf{p} \in \mathsf{U}} \mathsf{p}^\bullet$, respectively. We say that $\mathsf{U}$ is *enabled at* $\mathsf{m}$ when ${}^\bullet\mathsf{U} \le \mathsf{m}$, and in this case firing $\mathsf{U}$ results in the move $\mathsf{m} \xrightarrow{\mathsf{U}} \mathsf{m} - {}^\bullet\mathsf{U} + \mathsf{U}^\bullet$. Let $\mathbf{U} = \mathsf{U}_1 \cdots \mathsf{U}_n$ be a finite sequence of steps. We say that $\mathbf{U}$ is a *step firing sequence from* $\mathsf{m}$ *to* $\mathsf{m}'$ if $\mathsf{m} \xrightarrow{\mathsf{U}_1} \cdots \xrightarrow{\mathsf{U}_n} \mathsf{m}'$, and in this case we write $\mathsf{m} \xrightarrow{\mathbf{U}} \mathsf{m}'$.

**Concurrent execution of transactions** We now define how to execute transactions in parallel. The idea is to execute transactions in *isolation*, and then merge their changes, whenever they are mutually disjoint. The state updated resulting from the execution of a transaction are formalised as in Definition 1.

An *update collector* is a function $\Pi$ that, given a state $\sigma$ and a transaction $\mathsf{T}$, gives an update $\pi = \Pi(\sigma, \mathsf{T})$ which maps (at least) the updated qualified keys to their new values. In practice, update collectors can be obtained by instrumenting the run-time environment of smart contracts, so to record the state changes resulting from the execution of transactions. We formalise update collectors abstracting from the implementation details of such an instrumentation:

**Definition 8 (Update collector).** *We say that a function $\Pi$ is an* update collector *when* $[\![\mathsf{T}]\!]_\sigma = \sigma(\Pi(\sigma, \mathsf{T}))$, *for all $\sigma$ and $\mathsf{T}$.* ◇

There exists a natural ordering of collectors, which extends the ordering between state updates (i.e., set inclusion, when interpreting them as sets of substitutions): namely, $\Pi \sqsubseteq \Pi'$ holds when $\forall \sigma, \mathsf{T} : \Pi(\sigma, \mathsf{T}) \subseteq \Pi'(\sigma, \mathsf{T})$. The following lemma characterizes the least update collector w.r.t. this ordering.

**Lemma 5 (Least update collector).** *Let $\Pi^\star(\sigma, \mathsf{T}) = [\![\mathsf{T}]\!]_\sigma - \sigma$, where we define $\sigma' - \sigma$ as $\bigcup_{\sigma' p \neq \sigma p} \{\sigma' p / p\}$. Then, $\Pi^\star$ is the least update collector.* ◇

The merge of two state updates is the union of the corresponding substitutions; to avoid collisions, we make the merge operator undefined when the domains of the two updates overlap.

**Definition 9 (Merge of state updates).** *Let $\pi_0$, $\pi_1$ be state updates. When* $\mathrm{keys}(\pi_0) \cap \mathrm{keys}(\pi_1) = \emptyset$, *we define $\pi_0 \oplus \pi_1$ as follows:*

$$(\pi_0 \oplus \pi_1)p = \begin{cases} \pi_0 p & \textit{if } p \in \mathrm{keys}(\pi_0) \\ \pi_1 p & \textit{if } p \in \mathrm{keys}(\pi_1) \\ \bot & \textit{otherwise} \end{cases} \qquad ◇$$

The merge operator enjoys the commutative monoidal laws, and can therefore be extended to (finite) sets of state updates.

We now associate step firing sequences with state updates. The semantics of a step $\mathsf{U} = \{(\mathsf{T}_1, 1), \ldots, (\mathsf{T}_n, n)\}$ in $\sigma$ is obtained by applying to $\sigma$ the merge of the updates $\Pi(\sigma, \mathsf{T}_i)$, for all $i \in 1..n$ — whenever the merge is defined. The semantics of a step firing sequence is then obtained by folding that of its steps.

**Definition 10 (Semantics of step firing sequences).** *We define the semantics of step firing sequences, given $\Pi$ and $\sigma$, as:*

$$[\![\epsilon]\!]_\sigma^\Pi = \sigma \qquad [\![\mathsf{U}\mathsf{U}]\!]_\sigma^\Pi = [\![\mathsf{U}]\!]_{\sigma'}^\Pi \quad \textit{where } \sigma' = [\![\mathsf{U}]\!]_\sigma^\Pi = \sigma \bigoplus_{(\mathsf{T}, i) \in \mathsf{U}} \Pi(\sigma, \mathsf{T}) \quad ◇$$

*Example 8.* Let $\mathsf{t_f}, \mathsf{t_g}$, and $\mathsf{t_h}$ be as in Example 7, and let $\sigma \mathbb{C} x = \sigma \mathbb{C} y = 0$. Since $\Pi^\star(\sigma, \mathsf{T_f}) = \{1/\mathbb{C}.y\}$, $\Pi^\star(\sigma, \mathsf{T_g}) = \{1/\mathbb{C}.x\}$, and $\Pi^\star(\sigma, \mathsf{T_h}) = \{1/\mathbb{C}.z\}$, we have:

$$[\![\{\mathsf{t_f}, \mathsf{t_h}\}]\!]_\sigma^{\Pi^\star} = \sigma(\{1/\mathbb{C}.y\} \oplus \{1/\mathbb{C}.z\}) = \sigma\{1/\mathbb{C}.y, 1/\mathbb{C}.z\}$$
$$[\![\{\mathsf{t_g}, \mathsf{t_h}\}]\!]_\sigma^{\Pi^\star} = \sigma(\{1/\mathbb{C}.x\} \oplus \{1/\mathbb{C}.z\}) = \sigma\{1/\mathbb{C}.x, 1/\mathbb{C}.z\}$$
$$[\![\{\mathsf{t_f}, \mathsf{t_g}\}]\!]_\sigma^{\Pi^\star} = (\sigma\{1/\mathbb{C}.y\} \oplus \{1/\mathbb{C}.x\}) = \sigma\{1/\mathbb{C}.y, 1/\mathbb{C}.x\}$$

Note that, for all $\sigma$:

$$\begin{aligned}
[\![\mathsf{T_f T_h}]\!]_\sigma &= [\![\mathsf{T_h T_f}]\!]_\sigma = \sigma\{^1/_{e.y}, ^1/_{e.z}\} = [\![\{\mathsf{t_f, t_h}\}]\!]_\sigma^{\Pi^\star} \\
[\![\mathsf{T_g T_h}]\!]_\sigma &= [\![\mathsf{T_h T_g}]\!]_\sigma = \sigma\{^1/_{e.x}, ^1/_{e.z}\} = [\![\{\mathsf{t_g, t_h}\}]\!]_\sigma^{\Pi^\star}
\end{aligned}$$

So, the serial execution of $\mathsf{T_f}$ and $\mathsf{T_h}$ (in both orders) is equal to their concurrent execution (similarly for $\mathsf{T_g}$ and $\mathsf{T_h}$). Instead, for all $\sigma$ such that $\sigma\mathcal{C}x = \sigma\mathcal{C}y = 0$:

$$[\![\mathsf{T_f T_g}]\!]_\sigma = \sigma\{^1/_{e.y}\} \qquad [\![\mathsf{T_g T_f}]\!]_\sigma = \sigma\{^1/_{e.x}\} \qquad [\![\{\mathsf{t_f, t_g}\}]\!]_\sigma^{\Pi^\star} = \sigma\{^1/_{e.y}, ^1/_{e.x}\}$$

So, concurrent executions of $\mathsf{T_f}$ and $\mathsf{T_g}$ may differ from serial ones. This is coherent with the fact that, in Figure 3, $\mathsf{t_f}$ and $\mathsf{t_g}$ are *not* concurrent. $\diamond$

**Concurrent execution of blockchains** Theorem 5 relates serial executions of transactions to concurrent ones (which are rendered as step firing sequences). Item (a) establishes a confluence property: if two step firing sequences lead to the same marking, then they also lead to the same blockchain state. Item (b) ensures that the blockchain, interpreted as a sequence of transitions, is a step firing sequence, and it is *maximal* (i.e., there is a bijection between the transactions in the blockchain and the transitions of the corresponding net). Finally, item (c) ensures that executing maximal step firing sequences is equivalent to executing serially the blockchain.

**Theorem 5.** *Let $\mathbf{B} = \mathsf{T}_1 \cdots \mathsf{T}_n$. Then, in $\mathsf{N}_\#(\mathbf{B})$:*

(a) *if $\mathsf{m_0} \xrightarrow{\mathbf{U}} \mathsf{m}$ and $\mathsf{m_0} \xrightarrow{\mathbf{U'}} \mathsf{m}$, then $[\![\mathbf{U}]\!]_\sigma^{\Pi^\star} = [\![\mathbf{U'}]\!]_\sigma^{\Pi^\star}$, for all reachable $\sigma$;*
(b) *$\{(\mathsf{T}_1, 1)\} \cdots \{(\mathsf{T}_n, n)\}$ is a maximal step firing sequence;*
(c) *for all maximal step firing sequences $\mathbf{U}$, for all reachable $\sigma$, $[\![\mathbf{U}]\!]_\sigma^{\Pi^\star} = [\![\mathbf{B}]\!]_\sigma$.*

Remarkably, the implications of Theorem 5 also apply to $\mathsf{N}_{\#^W_R}(\mathbf{B})$.

*Example 9.* Recall $\mathbf{B} = \mathsf{T_f T_h T_g}$ and $\mathsf{N}_\#(\mathbf{B})$ from Example 7, let $\mathbf{U} = \{\mathsf{t_f, t_h}\}\{\mathsf{t_g}\}$, and let $\sigma$ be such that $\sigma\mathcal{C}x = \sigma\mathcal{C}y = 0$. As predicted by item (c) of Theorem 5:

$$[\![\mathbf{B}]\!]_\sigma = \sigma\{^1/_{e.y}\}\{^1/_{e.z}\} = [\![\mathbf{U}]\!]_\sigma^{\Pi^\star}$$

Let $\mathbf{U'} = \{\mathsf{t_f}\}\{\mathsf{t_g, t_h}\}$. We have that $\mathbf{U}$ and $\mathbf{U'}$ lead to the same marking, where the places $(\mathsf{t_f}, *)$, $(\mathsf{t_g}, *)$ and $(\mathsf{t_h}, *)$ contain one token each, while the other places have no tokens. By item (a) of Theorem 5 we conclude that $[\![\mathbf{U}]\!]_\sigma^{\Pi^\star} = [\![\mathbf{U'}]\!]_\sigma^{\Pi^\star}$. Now, let $\mathbf{U''} = \{\mathsf{t_h}\}\{\mathsf{t_f, t_g}\}$. Note that, although $\mathbf{U''}$ is maximal, it is not a step firing sequence, since the second step is not enabled (actually, $\mathsf{t_f}$ and $\mathsf{t_g}$ are not concurrent, as pointed out in Example 8). Therefore, the items of Theorem 5 do not apply to $\mathbf{U''}$, coherently with the fact that $\mathbf{U''}$ does not represent any sequential execution of $\mathbf{B}$. $\diamond$

## 5 Case study: ERC-721 token

We now apply our theory to an archetypal Ethereum smart contract, which implements a "non-fungible token" following the standard ERC-721 interface [13,14]. This contract defines the functions to transfer tokens between users, and to delegate their trade to other users. Currently, token transfers involve ∼ 50% of the transactions on the Ethereum blockchain [1], with larger peaks due to popular contracts like Cryptokitties [21].

We sketch below the implementation of the Token contract, using Solidity, the main high-level smart contract language in Ethereum (see Appendix B for the full implementation).

The contract state is defined by the following mappings:

```
mapping(uint256 => address) owner;
mapping(uint256 => bool) exists;
mapping(address => uint256) balance;
mapping (address => mapping (address => bool)) operatorApprovals;
```

Each token is uniquely identified by an integer value (of type uint256), while users are identified by an address. The mapping owner maps tokens to their owners' addresses (the zero address is used to denote a dummy owner). The mapping exists tells whether a token has been created or not, while balance gives the number of tokens owned by each user. The mapping operatorApprovals allows a user to delegate the transfer of all her tokens to third parties.

The function transferFrom transfers a token from the owner to another user. The require assertion rules out some undesirable cases, e.g., if the token does not exist, or it is not owned by the from user, or the user attempts to transfer the token to himself. Once all these checks are passed, the transfer succeeds if the sender of the transaction owns the token, or if he has been delegated by the owner. The mappings owner and balance are updated as expected.

```
function transferFrom(address from, address to, uint256 id)
    external {
  require (exists[id] && from==owner[id]
          && from!=to && to!=address(0));
  if (from==msg.sender || operatorApprovals[from][msg.sender]) {
    owner[id] = to;
    balance[from] -= 1;
    balance[to] += 1;
  }
}
```

The function setApprovalForAll delegates the transfers of all the tokens of the sender to the operator when the boolean isApproved is true, otherwise it revokes the delegation.

```
function setApprovalForAll(address operator, bool isApproved)
    external {
  operatorApprovals[msg.sender][operator] = isApproved;
}
```
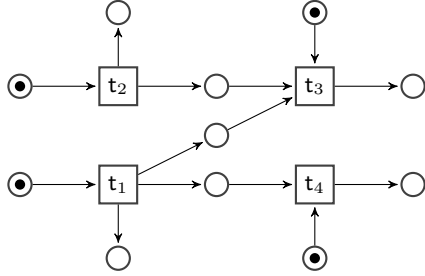
Fig. 4: Occurrence net for the blockchain $\mathbf{B} = \mathsf{T}_1\mathsf{T}_2\mathsf{T}_3\mathsf{T}_4$ of the ERC-721 token.

Assume that user $\mathcal{A}$ owns two tokens, identified by the integers 1 and 2, and consider the following transactions:

$$\mathsf{T}_1 = \mathcal{A} \xrightarrow{0} \mathtt{Token} : \mathtt{transferFrom}(\mathcal{A}, \mathcal{P}, 1)$$

$$\mathsf{T}_2 = \mathcal{A} \xrightarrow{0} \mathtt{Token} : \mathtt{setApprovalForAll}(\mathcal{B}, \mathtt{true})$$

$$\mathsf{T}_3 = \mathcal{B} \xrightarrow{0} \mathtt{Token} : \mathtt{transferFrom}(\mathcal{A}, \mathcal{Q}, 2)$$

$$\mathsf{T}_4 = \mathcal{P} \xrightarrow{0} \mathtt{Token} : \mathtt{transferFrom}(\mathcal{P}, \mathcal{B}, 1)$$

We have that $\mathsf{T}_1 \# \mathsf{T}_2$, $\mathsf{T}_2 \# \mathsf{T}_4$, and $\mathsf{T}_3 \# \mathsf{T}_4$ (this can be proved e.g. by using the static approximations in Appendix B), while the other combinations are not swappable. Let $\mathbf{B} = \mathsf{T}_1\mathsf{T}_2\mathsf{T}_3\mathsf{T}_4$. The resulting occurrence net is displayed in Figure 4. For instance, let $\mathbf{U} = \{\mathsf{T}_1, \mathsf{T}_2\}\{\mathsf{T}_3, \mathsf{T}_4\}$, i.e. $\mathsf{T}_1$ and $\mathsf{T}_2$ are executed concurrently, as well as $\mathsf{T}_3$ and $\mathsf{T}_4$. From item (c) of Theorem 5 we have that this concurrent execution is equivalent to the serial one.

Although this example deals with the marginal case where the sender and the receiver of tokens overlap, in practice the large majority of transactions in a block either involves distinct users, or invokes distinct ERC-721 interfaces, making it possible to increase the degree of concurrency of `transferFrom` transactions.

## 6 Related work and conclusions

We have proposed a static approach to improve the performance of blockchains by concurrently executing transactions. We have started by introducing a model of transactions and blockchains. We have defined two transactions to be *swappable* when inverting their order does not affect the blockchain state. We have then introduced a static approximation of swappability, based on a static analysis of the sets of keys read/written by transactions. We have rendered concurrent executions of a sequence of transactions as *step firing sequences* in the associated occurrence net. Our main technical result, Theorem 5, shows that these concurrent executions are semantically equivalent to the sequential one.

We can exploit our results in practice to improve the performances of miners and validators. Miners should perform the following steps to mine a block:

1. gather from the network a set of transactions, and put them in an arbitrary linear order **B**, which is the mined block;
2. compute the relation $\#_R^W$ on **B**, using a static analysis of read/written keys;
3. construct the occurrence net $\mathsf{N}_{\#_R^W}(\mathbf{B})$;
4. execute transactions concurrently according to the occurrence net, exploiting the available parallelism.

The behaviour of validators is almost identical to that of miners, except that in step (1), rather than choosing the order of transactions, they should adhere to the ordering of the mined block **B**. Note that in the last step, validators can execute any maximal step firing sequence which is coherent with their degree of parallelism: item (c) of Theorem 5 ensures that the resulting state is equal to the state obtained by the miner. The experiments in [11] suggest that parallelization may lead to a significant improvement of the performance of nodes: the benchmarks on a selection of representative contracts show an overall speedup of 1.33x for miners and 1.69x for validators, using only three cores.

Note that malevolent users could attempt a denial-of-service attack by publishing contracts which are hard to statically analyse, and therefore are not suitable for parallelization. This kind of attacks can be mitigated by adopting a mining strategy that gives higher priority to parallelizable transactions.

**Applying our approach to Ethereum**  Applying our theory to Ethereum would require a static analysis of read/written keys at the level of EVM bytecode. As far as we know, the only tool implementing such an analysis is ES-ETH [15]. However, the current version of the tool has several limitations, like e.g. the compile-time approximation of dictionary keys and of values shorter than 32 bytes, which make ES-ETH not directly usable to the purposes of our work. In general, precise static analyses at the level of the Ethereum bytecode are difficult to achieve, since the language has features like dynamic dispatching and pointer aliasing which are notoriously a source of imprecision for static analysis. However, coarser approximations of read/written keys may be enough to speed-up the execution of transactions. For instance, in Ethereum, blocks typically contain many transactions which transfer tokens between participants, and many of them involve distinct senders and receivers. A relatively simple analysis of the code of token contracts (which is usually similar to that in Section 5) may be enough to detect that these transactions are swappable.

Aiming at minimality, our model does not include the *gas mechanism*, which is used in Ethereum to pay miners for executing contracts. The sender of a transaction deposits into it some crypto-currency, to be paid to the miner which appends the transaction to the blockchain. Each instruction executed by the miner consumes part of this deposit; when the deposit reaches zero, the miner stops executing the transaction. At this point, all the effects of the transaction (except the payment to the miner) are rolled back. Our transaction model could be easily extended with a gas mechanism, by associating a cost to statements and recording the gas consumption in the environment. Remarkably, adding gas does not invalidate approximations of read/written keys which are correct while

neglecting gas. However, a gas-aware analysis may be more precise of a gas-oblivious one: for instance, in the statement `if` $k$ `then` $f_{long}();x{:=}1$ `else` $y{:=}1$ (where $f_{long}$ is a function which exceeds the available gas) a gas-aware analysis would be able to detect that x is not written.

**Related work** A few works study how to optimize the execution of smart contracts on Ethereum, using dynamic techniques adopted from software transactional memory [4,11,12]. These works are focussed on empirical aspects (e.g., measuring the speedup obtained on a given benchmark), while we focus on the theoretical counterpart. In [11,12], miners execute a set of transactions speculatively in parallel, using abstract locks and inverse logs to dynamically discover conflicts and to recover from inconsistent states. The obtained execution is guaranteed to be equivalent to a serial execution of the same set of transactions. The work [4] proposes a conceptually similar technique, but based on optimistic software transactional memory. Since speculative execution is non-deterministic, in both approaches miners need to communicate the chosen schedule of transactions to validators, to allow them to correctly validate the block. This schedule must be embedded in the mined block: since Ethereum does not support this kind of block metadata, these approaches would require a "soft-fork" of the blockchain to be implemented in practice. Instead, our approach is compatible with the current Ethereum, since miners only need to append transactions to the blockchain. Compared to [11,4], where conflicts are detected dynamically, our approach relies on a static analysis to detect potential conflicts. Since software transactional memory introduces a run-time overhead, in principle a static technique could allow for faster executions, at the price of a preprocessing phase. Saraph and Herlihy [19] study the effectiveness of speculatively executing smart contracts in Ethereum. They sample past blocks of transactions (from July 2016 to December 2017), replay them by using a speculative execution engine, and measure the speedup obtained by parallel execution. Their results show that simple speculative strategies yield non-trivial speed-ups. Further, they note that many of the data conflicts (i.e. concurrent read/write accesses to the same state location) arise in periods of high traffic, and they are caused by a small number of popular contracts, like e.g. tokens.

In the permissioned setting, Hyperledger Fabric [3] follows the "execute first and then order" paradigm: transactions are executed speculatively, and then their ordering is checked for correctness [2]. In this paradigm, appending a transaction requires a few steps. First, a client proposes a transaction to a set of "endorsing" peers, which simulate the transaction without updating the blockchain. The output of the simulation includes the state updates of the transaction execution, and the sets of read/written keys. These sets are then signed by the endorsing peers, and returned to the client, which submits them to the "ordering" peers. These nodes order transactions in blocks, and send them to the "committing" peers, which validate them. A block $\mathsf{T}_1 \cdots \mathsf{T}_n$ is valid when, if a key $k$ is read by transaction $\mathsf{T}_i$, then $k$ has not been written by a transaction $\mathsf{T}_j$ with $j < i$. Finally, validated blocks are appended to the blockchain. Our model is coherent with Ethereum, which does not support speculative execution.

**Future works** A relevant line of research is the design of domain-specific languages for smart contracts that are directly amenable to techniques that, like ours, increase the degree of concurrency of executions. For this purpose, the language should support static analyses of read/written keys, like the one we use to define the strong swappability relation. Although the literature describes various static analyses of smart contracts, most of them are focussed on finding security vulnerabilities, rather than enhancing concurrency.

Outside the realm of smart contracts, a few papers propose static analyses of read/written variables. The paper [10] describes an analysis based on separation logic, and applies it to resolve conflicts in the setting of *snapshot isolation* for transactional memory in Java. When a conflict is detected, the read/write sets are used to determine how the code can be modified to resolve it. The paper [9] presents a static analysis to infer read and write locations in a C-like language with atomic sections. The analysis is used to translate atomic sections into standard lock operations. The design of new smart contract languages could take advantage of these analyses.

# References

1. Ethereum token dynanics, https://stat.bloxy.info/superset/dashboard/tokens
2. Hyperledger Fabric: Read-write set semantics. https://hyperledger-fabric.readthedocs.io/en/release-1.4/readwrite.html
3. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A.D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., Cocco, S.W., Yellick, J.: Hyperledger Fabric: a distributed operating system for permissioned blockchains. In: EuroSys. pp. 30:1–30:15 (2018). https://doi.org/10.1145/3190508.3190538
4. Anjana, P.S., Kumari, S., Peri, S., Rathor, S., Somani, A.: An efficient framework for optimistic concurrent execution of smart contracts. In: PDP. pp. 83–92 (2019). https://doi.org/10.1109/EMPDP.2019.8671637
5. Bartoletti, M., Galletta, L., Murgia, M.: A minimal core calculus for Solidity contracts. In: Cryptocurrencies and Blockchain Technology. LNCS, vol. 11737, pp. 233–243. Springer (2019). https://doi.org/10.1007/978-3-030-31500-9_15
6. Bernstein, A.J.: Analysis of programs for parallel processing. IEEE Trans. on Electronic Computers **EC-15**(5), 757–763 (1966). https://doi.org/10.1109/PGEC.1966.264565
7. Best, E., Devillers, R.R.: Sequential and concurrent behaviour in petri net theory. Theor. Comput. Sci. **55**(1), 87–136 (1987). https://doi.org/10.1016/0304-3975(87)90090-9

8. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)
9. Cherem, S., Chilimbi, T.M., Gulwani, S.: Inferring locks for atomic sections. In: ACM SIGPLAN PLDI. pp. 304–315 (2008). https://doi.org/10.1145/1375581.1375619
10. Dias, R.J., Lourenço, J.M., Preguiça, N.M.: Efficient and correct transactional memory programs combining snapshot isolation and static analysis. In: USENIX Conf. on Hot topics in Parallelism (HotPar) (2011)
11. Dickerson, T.D., Gazzillo, P., Herlihy, M., Koskinen, E.: Adding concurrency to smart contracts. In: ACM PODC. pp. 303–312 (2017). https://doi.org/10.1145/3087801.3087835
12. Dickerson, T.D., Gazzillo, P., Herlihy, M., Koskinen, E.: Adding concurrency to smart contracts. Bulletin of the EATCS **124** (2018)
13. Entriken, W., Shirley, D., Evans, J., Sachs, N.: EIP 721: ERC-721 non-fungible token standard, https://eips.ethereum.org/EIPS/eip-721
14. Fröwis, M., Fuchs, A., Böhme, R.: Detecting token systems on Ethereum. In: Financial Cryptography and Data Security. LNCS, vol. 11598, pp. 93–112. Springer (2019). https://doi.org/10.1007/978-3-030-32101-7_7
15. Marcia, D.: ES-ETH: Ethereum state change examiner. https://github.com/DiegoMarcia/ES-ETH (2019)
16. Mazurkiewicz, A.W.: Basic notions of trace theory. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. LNCS, vol. 354, pp. 285–363. Springer (1988). https://doi.org/10.1007/BFb0013025
17. Nielson, F., Nielson, H.R., Hankin, C.: Principles of program analysis. Springer (1999). https://doi.org/10.1007/978-3-662-03811-6
18. Reisig, W.: Petri Nets: An Introduction, Monographs in Theoretical Computer Science. An EATCS Series, vol. 4. Springer (1985). https://doi.org/10.1007/978-3-642-69968-9
19. Saraph, V., Herlihy, M.: An empirical study of speculative concurrency in Ethereum smart contracts. CoRR **abs/1901.01376** (2019), http://arxiv.org/abs/1901.01376
20. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997), http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/548
21. Young, J.: CryptoKitties sales hit $12 million, could be Ethereum's killer app after all. https://cointelegraph.com/news/cryptokitties-sales-hit-12-million-could-be-ethereums-killer-app-a (2017)
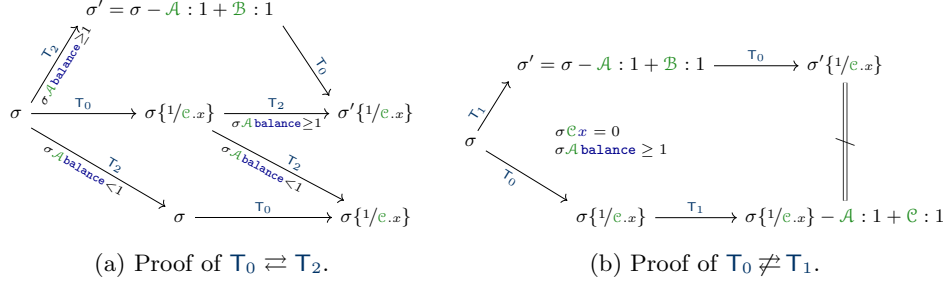
(a) Proof of $\mathsf{T}_0 \rightleftarrows \mathsf{T}_2$.
(b) Proof of $\mathsf{T}_0 \not\rightleftarrows \mathsf{T}_1$.

Fig. 5: Proofs for Example 2. A transition $\mathsf{T}$ from $\sigma$ can be taken only if the guard below the arrow is satisfied in $\sigma$.



$$\sigma' = \sigma - \mathcal{A} : 1 + \mathcal{B} : 1 \quad \sigma'' = \sigma'\{1/e.x\}$$
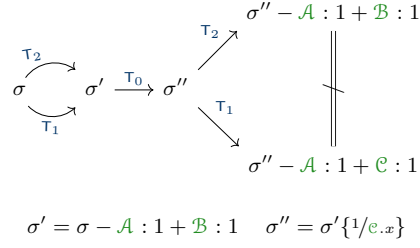
Fig. 6: Proof of $\mathsf{T}_1\mathsf{T}_0\mathsf{T}_2 \not\sim \mathsf{T}_2\mathsf{T}_0\mathsf{T}_1$ for Example 3.

## A  Proofs

**Lemma 1.** *For all $P, Q \subseteq \mathbb{P}$: (i) $\sim_P$ is an equivalence relation; (ii) if $\sigma \sim_P \sigma'$ and $Q \subseteq P$, then $\sigma \sim_Q \sigma'$; (iii) $\sim = \sim_\mathbb{P}$.* ◇

*Proof.* Items (i) and (ii) are trivial. The inclusion $\sim_\mathbb{P} \subseteq \sim$ is trivial, and $\sim \subseteq \sim_\mathbb{P}$, follows from item (ii). □

**Lemma 2.** $\mathbf{B} \sim \mathbf{B}' \implies \forall \mathbf{B}_0, \mathbf{B}_1 : \mathbf{B}_0\mathbf{B}\mathbf{B}_1 \sim \mathbf{B}_0\mathbf{B}'\mathbf{B}_1$. ◇

*Proof.* Direct from the fact that semantics of statements is a function, and it only depends on the blockchain states after the execution of $\mathbf{B}$ and $\mathbf{B}'$, which are equal in any $\sigma$ since $\mathbf{B} \sim \mathbf{B}'$. □

**Theorem 1.** $\simeq_{\rightleftarrows} \subseteq \sim$. ◇

*Proof.* By definition, $\simeq_{\rightleftarrows}$ is the least equivalence relation closed under the rules:

$$\frac{}{\epsilon \simeq_{\rightleftarrows} \epsilon}[\simeq 0] \quad \frac{}{\mathsf{T} \simeq_{\rightleftarrows} \mathsf{T}}[\simeq 1] \quad \frac{\mathsf{T} \rightleftarrows \mathsf{T}'}{\mathsf{T}\mathsf{T}' \simeq_{\rightleftarrows} \mathsf{T}'\mathsf{T}}[\simeq 2] \quad \frac{\mathbf{B}_0 \simeq_{\rightleftarrows} \mathbf{B}'_0 \quad \mathbf{B}_1 \simeq_{\rightleftarrows} \mathbf{B}'_1}{\mathbf{B}_0\mathbf{B}_1 \simeq_{\rightleftarrows} \mathbf{B}'_0\mathbf{B}'_1}[\simeq 3]$$

Let $\mathbf{B} \simeq_{\rightleftarrows} \mathbf{B}'$. We have to show $\mathbf{B} \sim \mathbf{B}'$. We proceed by induction on the rules above. The case for rules $[\simeq 0]$ and $[\simeq 1]$ follows by the fact that $\sim$ is an equivalence

relation (Lemma 1) and hence reflexive. The case for rule $[\simeq 2]$ follows immediately by Definition 3. For rule $[\simeq 3]$, first note that $\mathbf{B} = \mathbf{B}_0\mathbf{B}_1$ and $\mathbf{B}' = \mathbf{B}'_0\mathbf{B}'_1$. By the induction hypothesis it follows that:

$$\mathbf{B}_0 \sim \mathbf{B}'_0 \quad \text{and} \quad \mathbf{B}_1 \sim \mathbf{B}'_1$$

Therefore, by two applications of Lemma 2:

$$\mathbf{B} = \mathbf{B}_0\mathbf{B}_1 \sim \mathbf{B}_0\mathbf{B}'_1 \sim \mathbf{B}'_0\mathbf{B}'_1 = \mathbf{B}' \qquad\qquad \square$$

**Lemma 6.** *Let $\bullet \in \{r, w\}$. Then:*

*(a) if $P \models^\bullet \mathsf{T}$ and $P \subseteq P'$, then $P' \models^\bullet \mathsf{T}$;*
*(b) if $P \models^w \mathsf{T}$ and $Q \models^w \mathsf{T}$, then $P \cap Q \models^w \mathsf{T}$.*

*Proof.* Item (a). For the case $\bullet = w$, let $P \models^w \mathsf{T}$ and $P \subseteq P'$. Let $Q$ be such that $Q \cap P' = \emptyset$. We have to show that $\mathsf{T} \sim_Q \epsilon$. Since $P \subseteq P'$, it must be $Q \cap P = \emptyset$. Then, since $P \models^w \mathsf{T}$, it must be $\mathsf{T} \sim_Q \epsilon$, as required. For the case $\bullet = r$, let $P \models^r \mathsf{T}$ and $P \subseteq P'$. We have to show that, for all $\mathbf{B}_1, \mathbf{B}_2$, if $\mathbf{B}_1 \sim_{P'} \mathbf{B}_2$ and $\mathbf{B}_1 \sim_Q \mathbf{B}_2$, then $\mathbf{B}_1\mathsf{T} \sim_Q \mathbf{B}_2\mathsf{T}$. But this follows immediately by the fact that $P \subseteq P'$ and $P \models^r \mathsf{T}$.

Item (b). Let $R$ be such that $R \cap (P \cap Q) = \emptyset$. Since $P \models^w \mathsf{T}$ and $(R \setminus P) \cap P = \emptyset$, it must be:

$$\mathsf{T} \sim_{R \setminus P} \epsilon$$

Similarly, since $Q \models^w \mathsf{T}$ and $(R \setminus Q) \cap Q = \emptyset$, we have that:

$$\mathsf{T} \sim_{R \setminus Q} \epsilon$$

By assumption $R \cap (P \cap Q) = \emptyset$, then $(R \setminus P) \cup (R \setminus Q) = R$. By Definition 2, we conclude that:

$$\mathsf{T} \sim_R \mathsf{T} \sim_{(R \setminus P) \cup (R \setminus Q)} \epsilon \qquad\qquad \square$$

The following example shows that part (b) of Lemma 6 does not hold for read approximations.

*Example 10.* Let $\mathcal{C}$ be a contract with exactly two procedures:

$$\mathtt{f}(x) \{k := x; k' := x\}$$
$$\mathtt{g}() \{\text{if } k \neq \mathcal{A} \text{ then } \mathcal{B}.\texttt{transfer}(\texttt{balance}) \text{ else skip}\}$$

and let $\mathsf{T} = \mathcal{A} \xrightarrow{0} \mathcal{C} : \mathtt{g}()$. Note that, in any reachable state $\sigma$, it must be $\sigma\mathcal{C}k = \sigma\mathcal{C}k'$. Let $Q$ be such that $\mathbf{B} \sim_Q \mathbf{B}'$, and let $\sigma = [\![\mathbf{B}]\!]$, $\sigma' = [\![\mathbf{B}']\!]$, $n = \sigma\mathcal{C}\texttt{balance}$, and $n' = \sigma'\mathcal{C}\texttt{balance}$. Appending $\mathsf{T}$ to $\mathbf{B}$ and $\mathbf{B}'$ will result in:

$$[\![\mathsf{T}]\!]_\sigma = \begin{cases} \sigma - \mathcal{C} : n + \mathcal{B} : n & \text{if } \sigma\mathcal{C}k \neq \mathcal{A} \\ \sigma & \text{otherwise} \end{cases}$$

$$[\![\mathsf{T}]\!]_{\sigma'} = \begin{cases} \sigma' - \mathcal{C} : n' + \mathcal{B} : n' & \text{if } \sigma'\mathcal{C}k \neq \mathcal{A} \\ \sigma' & \text{otherwise} \end{cases}$$

If $\mathbf{B} \sim_{\{k\}} \mathbf{B'}$, then the conditions $\sigma \mathcal{C} k \neq \mathcal{A}$ and $\sigma' \mathcal{C} k \neq \mathcal{A}$ are equivalent. Therefore, $[\![\mathbf{B}\mathsf{T}]\!] = [\![\mathsf{T}]\!]_\sigma \sim_Q [\![\mathsf{T}]\!]_{\sigma'} = [\![\mathbf{B'}\mathsf{T}]\!]$, and so we have proved that $\{k\} \models^r$ $\mathsf{T}$. Similarly, we obtain that $\{k'\} \models^r \mathsf{T}$, since $k$ and $k'$ are always bound to the same value. Note however that $\{k\} \cap \{k'\} = \emptyset$ is *not* a safe approximation of the keys read by $\mathsf{T}$. For instance, if $\sigma \mathcal{C} k = \mathcal{A} \neq \sigma' \mathcal{C} k'$ and $\sigma \mathcal{C}\texttt{balance} = \sigma' \mathcal{C}\texttt{balance}$, then appending $\mathsf{T}$ to $\mathbf{B}$ or to $\mathbf{B'}$ results in states which differ in the balance of $\mathcal{C}$. $\qquad\square$

**Lemma 7.** *Let $P_\mathsf{T}^w \models^w \mathsf{T}$, $P_{\mathsf{T'}}^w \models^w \mathsf{T'}$, and $P_\mathsf{T}^r \models^r \mathsf{T}$. Then:*

$$P_{\mathsf{T'}}^w \cap P_\mathsf{T}^r = \emptyset = P_\mathsf{T}^w \cap S_{\mathsf{T'}}^w \quad \implies \quad \mathsf{T} \sim_{P_\mathsf{T}^w} \mathsf{T'}\mathsf{T}$$

*Proof.* Since $P_{\mathsf{T'}}^w \models^w \mathsf{T'}$ and $P_{\mathsf{T'}}^w \cap P_\mathsf{T}^w = \emptyset$, by Definition 5 we have $\mathsf{T'} \sim_{P_\mathsf{T}^w} \epsilon$. We prove that $\mathsf{T'} \sim_{P_\mathsf{T}^r} \epsilon$. By contradiction, assume that $\mathsf{T'} \not\sim_p \epsilon$ for some $p \in P_\mathsf{T}^r$. Since $P_{\mathsf{T'}}^w \models^w \mathsf{T'}$, it must be $p \in P_{\mathsf{T'}}^w$ — contradiction, since $P_{\mathsf{T'}}^w \cap P_\mathsf{T}^r = \emptyset$. Since $P_\mathsf{T}^r \models^r \mathsf{T}$, by Definition 5 we conclude that $\mathsf{T'}\mathsf{T} \sim_{P_\mathsf{T}^w} \mathsf{T}$. $\qquad\square$

**Lemma 8.** $\mathsf{T}_1 \#_R^W \mathsf{T}_2 \implies \mathsf{T}_1 \mathsf{T}_2 \sim_{W(\mathsf{T}_1)} \mathsf{T}_2 \mathsf{T}_1$

*Proof.* By Definition 7, it must be:

$$W(\mathsf{T}_1) \cap W(\mathsf{T}_2) = \emptyset$$

So, since $W(\mathsf{T}_2) \models^w \mathsf{T}_2$, by Definition 5 we have:

$$\epsilon \sim_{W(\mathsf{T}_1)} \mathsf{T}_2$$

and so, since $\sim_{W(\mathsf{T}_1)}$ is a congruence:

$$\mathsf{T}_1 \sim_{W(\mathsf{T}_1)} \mathsf{T}_1 \mathsf{T}_2$$

By Definition 7, it must be:

$$W(\mathsf{T}_2) \cap R(\mathsf{T}_1) = \emptyset = W(\mathsf{T}_1) \cap W(\mathsf{T}_2)$$

Then, by Lemma 7:

$$\mathsf{T}_1 \sim_{W(\mathsf{T}_1)} \mathsf{T}_2 \mathsf{T}_1$$

By simmetry and transitivity of $\sim$ (Lemma 1), we conclude:

$$\mathsf{T}_1 \mathsf{T}_2 \sim_{W(\mathsf{T}_1)} \mathsf{T}_2 \mathsf{T}_1 \qquad\qquad\qquad\square$$

**Lemma 9.** $\mathsf{T}_1 \#_R^W \mathsf{T}_2 \implies \mathsf{T}_1 \rightleftarrows \mathsf{T}_2$

*Proof.* By applying Lemma 8 twice:

$$\mathsf{T}_1 \mathsf{T}_2 \sim_{W(\mathsf{T}_1)} \mathsf{T}_2 \mathsf{T}_1 \qquad \mathsf{T}_2 \mathsf{T}_1 \sim_{W(\mathsf{T}_2)} \mathsf{T}_1 \mathsf{T}_2$$

Let $P = \mathbb{P} \setminus (W(\mathsf{T}_1) \cup W(\mathsf{T}_2))$. Since $P \cap W(\mathsf{T}_1) = \emptyset = P \cap W(\mathsf{T}_2)$, by applying Definition 5 twice:

$$\epsilon \sim_P \mathsf{T}_1 \qquad \epsilon \sim_P \mathsf{T}_2$$

Then, since $\sim_P$ is a congruence:

$$\mathsf{T}_1\mathsf{T}_2 \sim_P \mathsf{T}_2\mathsf{T}_1$$

Summing up:

$$\mathsf{T}_1\mathsf{T}_2 \sim_{P\cup(W(\mathsf{T}_1)\cup W(\mathsf{T}_2))} \mathsf{T}_2\mathsf{T}_1$$

from which we obtain the thesis, since $\sim_{\mathbb{P}}=\sim$. $\qquad\square$

**Lemma 10.** $\mathsf{T}_1\#\mathsf{T}_2 \implies \exists W, R : \mathsf{T}_1\#_R^W\mathsf{T}_2$

*Proof.* Straightforward from Definition 6 and Definition 7. $\qquad\square$

**Theorem 2.** $\mathsf{T}\#\mathsf{T}' \implies \mathsf{T} \rightleftarrows \mathsf{T}'$. $\qquad\diamond$

*Proof.* By Lemma 10 and Lemma 9. $\qquad\square$

**Lemma 3.** *For all static analyses $W$ and $R$: (i) $\#_R^W \subseteq \#$; (ii) if $W$ and $R$ are computable, then $\#_R^W$ is decidable.* $\qquad\diamond$

*Proof.* Trivial. $\qquad\square$

**Theorem 3.** $\simeq_\# \subseteq \simeq_\rightleftarrows$. $\qquad\diamond$

*Proof.* Straightforward by Theorem 2. $\qquad\square$

**Lemma 11.** $\mathbf{B} \simeq_\# \mathbf{B}' \implies \mathbf{B} \sim \mathbf{B}'$

*Proof.* Direct by Theorems 1 and 3. $\qquad\square$

**Lemma 12.** $\oplus$ *is commutative and associative, with $\lambda p.\bot$ as neutral element.*

*Proof.* Trivial. $\qquad\square$

**Lemma 13.** *If $\pi_1 \oplus \pi_2 = \pi$, then $\pi = \pi_1\pi_2$.*

*Proof.* Since $\pi_1 \oplus \pi_2$ is defined, it must be $\mathrm{keys}(\pi_1) \cap \mathrm{keys}(\pi_2) = \emptyset$. Let $p$ be a qualified key. We have two cases:

- $p \in \mathrm{keys}(\pi)$. Since $\mathrm{keys}(\pi) = \mathrm{keys}(\pi_1) \cup \mathrm{keys}(\pi_2)$, we have two subcases:
  - $p \in \mathrm{keys}(\pi_1)$. Then, $\pi p = \pi_1 p$. By disjointness, $p \notin \mathrm{keys}(\pi_2)$, and hence $\pi_1\pi_2 p = \pi_1 p$.
  - $p \in \mathrm{keys}(\pi_2)$. Then, $\pi p = \pi_2 p = \pi_1\pi_2 p$.
- $p \notin \mathrm{keys}(\pi)$. Then, $p \notin \mathrm{keys}(\pi_1)$, $p \notin \mathrm{keys}(\pi_2)$, and so $\pi p = \bot = \pi_1\pi_2 p$. $\quad\square$

**Lemma 14.** *If $\mathbf{B}_1 \lhd \mathbb{T}_1$ and $\mathbf{B}_2 \lhd \mathbb{T}_2$, then $\mathbf{B}_1\mathbf{B}_2 \lhd (\mathbb{T}_1 \cup \mathbb{T}_2)$.*

*Proof.* By induction on $|\mathbf{B}_2|$. For the base case, it must be $\mathbf{B}_2 = \epsilon$ and hence $\mathbb{T}_2 = \emptyset$. Then, $\mathbf{B}_1\mathbf{B}_2 = \mathbf{B}_1$ and $\mathbb{T}_1 \cup \mathbb{T}_2 = \mathbb{T}_1$. Therefore, the thesis coincides with the first hypothesis. For the induction case, it must be $\mathbf{B}_2 = \mathbf{B}_2'\mathsf{T}$, with $|\mathbf{B}_2'| = n$. Furthermore, it must be $\mathbb{T}_2 = \{\mathsf{T}\} \cup \mathbb{T}'_2$, for some $\mathbb{T}'_2$ such that $\mathbf{B}_2' \lhd \mathbb{T}'_2$. By the induction hypothesis:

$$\mathbf{B}_1\mathbf{B}_2' \lhd (\mathbb{T}_1 \cup \mathbb{T}'_2)$$

Then:

$$\mathbf{B}_1\mathbf{B}_2'\mathsf{T} = \mathbf{B}_1\mathbf{B}_2 \lhd (\{\mathsf{T}\} \cup \mathbb{T}_1 \cup \mathbb{T}'_2) = \mathbb{T}_1 \cup \mathbb{T}_2 \qquad\square$$

**Lemma 15.** *Let* $\mathbf{B}$ *and* $\mathsf{T}$ *be such that* $\mathbf{B} = \mathbf{B}_1\mathsf{T}'\mathbf{B}_2 \implies \mathsf{T}\#_R^W\mathsf{T}'$. *Then, for all* $\mathbf{B}'$, $[\![\mathbf{B}']\!] \sim_{R(\mathsf{T})} [\![\mathbf{B}]\!]_{[\![\mathbf{B}']\!]}$ *and* $[\![\mathbf{B}']\!] \sim_{W(\mathsf{T})} [\![\mathbf{B}]\!]_{[\![\mathbf{B}']\!]}$.

*Proof.* A simple induction on $|\mathbf{B}|$, using Definition 5 for the induction case. $\square$

**Lemma 16.** *If* $\mathsf{T}\#_R^W\mathsf{T}'$ *for all* $\mathsf{T}' \in \mathbb{T}$ *and* $\mathbf{B}_1 \lhd \mathbb{T}$ *then,* $\mathbf{B} = \mathbf{B}_1\mathsf{T}'\mathbf{B}_2 \implies \mathsf{T}\#_R^W\mathsf{T}'$.

*Proof.* A simple induction on $|\mathbb{T}|$. $\square$

**Lemma 17.** *If* $p \models^r \mathsf{T}$, $\mathbf{B}_1 \sim_p \mathbf{B}_2$ *and* $\mathbf{B}_1 \sim_q \mathbf{B}_2$, *then* $[\![\mathbf{B}_1\mathsf{T}]\!] \sim_q [\![\mathbf{B}_1]\!]\Pi([\![\mathbf{B}_2]\!], \mathsf{T})$.

*Proof.* Let $\pi_1 = \Pi([\![\mathbf{B}_1]\!], \mathsf{T})$ and $\pi_2 = \Pi([\![\mathbf{B}_2]\!], \mathsf{T})$. By Definition 8, $[\![\mathbf{B}_1\mathsf{T}]\!] = [\![\mathbf{B}_1]\!]\pi_1$. Let $p \in Q$. We have two cases:

- $p \in \mathrm{keys}(\pi_2)$.

$$[\![\mathbf{B}_1]\!]\pi_2 p = \pi_2 p = [\![\mathbf{B}_2]\!]\pi_2 p = [\![\mathbf{B}_2\mathsf{T}]\!]p = [\![\mathbf{B}_1\mathsf{T}]\!]p$$

- $p \notin \mathrm{keys}(\pi_2)$.

$$[\![\mathbf{B}_1]\!]\pi_2 p = [\![\mathbf{B}_1]\!]p = [\![\mathbf{B}_2]\!]p = [\![\mathbf{B}_2]\!]\pi_2 p = [\![\mathbf{B}_1\mathsf{T}]\!]p \qquad \square$$

**Definition 11.** *Let* $\Pi$ *be a state updater, and let* $W$ *be such that* $\forall \mathsf{T} : W(\mathsf{T}) \models^w \mathsf{T}$. *We say that* $\Pi$ *and* $W$ *are* compatible *when* $\forall \sigma, \mathsf{T} : \mathrm{keys}(\Pi(\sigma, \mathsf{T})) \subseteq W(\mathsf{T})$.

We extend the semantics of transactions to finite *multisets* of transactions. Hereafter, we denote with $[]$ the empty multiset, with $[\mathsf{T}_1, \ldots, \mathsf{T}_n]$ the multiset containing $\mathsf{T}_1, \ldots, \mathsf{T}_n$, and with $A + B$ the sum between multisets, i.e. $(A + B)(x) = A(x) + B(x)$ for all $x$.

**Definition 12 (Semantics of multisets of transactions).** *We denote the semantics of a multiset of transactions* $\mathbb{T}$, *in a state* $\sigma$ *and an update collector* $\Pi$, *as* $[\![\mathbb{T}]\!]_\sigma^\Pi$, *where the partial function* $[\![\cdot]\!]_\sigma^\Pi$ *is defined as:* $[\![\mathbb{T}]\!]_\sigma^\Pi = \sigma \bigoplus_{\mathsf{T} \in \mathbb{T}} \Pi(\sigma, \mathsf{T})$.

Hereafter, we say that a multiset $\mathbb{T}$ is strongly swappable w.r.t. $\#_R^W$ if $\forall \mathsf{T} \in \mathbb{T}, \forall \mathsf{T}' \in \mathbb{T} \setminus [\mathsf{T}] : \mathsf{T}\#_R^W\mathsf{T}'$.

**Lemma 18.** *If* $\mathbb{T}$ *is strongly swappable w.r.t.* $\#_R^W$, $\mathbf{B} \lhd \mathbb{T}$ *and* $\Pi$ *is compatible with* $W$ *then, for all* $\mathbf{B}_0$: $[\![\mathbb{T}]\!]_{[\![\mathbf{B}_0]\!]}^\Pi = [\![\mathbf{B}]\!]_{[\![\mathbf{B}_0]\!]}$.

*Proof.* By induction on $|\mathbf{B}|$. For the base case, it must be $\mathbf{B} = \epsilon$ and $\mathbb{T} = \emptyset$, and hence $[\![\emptyset]\!]_{[\![\mathbf{B}_0]\!]}^\Pi = [\![\mathbf{B}_0]\!] = [\![\epsilon]\!]_{[\![\mathbf{B}_0]\!]}$. For the induction case, it must be $\mathbf{B} = \mathbf{B}'\mathsf{T}$, with $|\mathbf{B}'| = n$. Clearly, $\mathbb{T} = \{\mathsf{T}\} \cup \mathbb{T}'$ for some $\mathbb{T}'$ such that $\mathbf{B}' \lhd \mathbb{T}'$. Let $\Pi([\![\mathbf{B}_0]\!], \mathsf{T}) = \pi_\mathsf{T}$. By the induction hypothesis:

$$[\![\mathbb{T}']\!]_{[\![\mathbf{B}_0]\!]}^\Pi = [\![\mathbf{B}']\!]_{[\![\mathbf{B}_0]\!]} \qquad (1)$$

Note that:
$$\llbracket \mathbb{T}' \rrbracket^{\Pi}_{\llbracket \mathbf{B}_0 \rrbracket} = \llbracket \mathbf{B}_0 \rrbracket \pi' \tag{2}$$

Where $\pi' \bigoplus_{\mathsf{T}' \in \mathbb{T}'} \Pi(\llbracket \mathbf{B}_0 \rrbracket, \mathsf{T}'))$. Let $\Pi(\llbracket \mathbf{B}_0 \rrbracket, \mathsf{T}) = \pi_\mathsf{T}$. Since $\mathbb{T}$ is strongly swappable w.r.t. $\#^W_R$ and $\Pi$ is compatible with $W$, it must be $\mathrm{keys}(\pi') \cap \mathrm{keys}(\pi_\mathsf{T}) = \emptyset$, and hence $(\pi' \oplus \pi_\mathsf{T})$ is defined. Then, it must be:

$$
\begin{aligned}
\llbracket \mathbb{T} \rrbracket^{\Pi}_{\llbracket \mathbf{B}_0 \rrbracket} &= \llbracket \mathbf{B}_0 \rrbracket (\pi' \oplus \pi_\mathsf{T}) \\
&= \llbracket \mathbf{B}_0 \rrbracket \pi' \pi_\mathsf{T} && \text{By Lemma 13} \\
&= \llbracket \mathbb{T}' \rrbracket^{\Pi}_{\llbracket \mathbf{B}_0 \rrbracket} \pi_\mathsf{T} && \text{By Equation (2)} \\
&= \llbracket \mathbf{B}' \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} \pi_\mathsf{T} && \text{By Equation (1)} \qquad (3)
\end{aligned}
$$

We have that:
$$\llbracket \mathbf{B} \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} = \llbracket \mathbf{B}'\mathsf{T} \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} = \llbracket \mathbf{B}' \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} \pi'_\mathsf{T}$$

Where $\pi'_\mathsf{T} = \Pi(\llbracket \mathbf{B}'_0 \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket}, \mathsf{T})$. As $\mathrm{keys}(\pi_\mathsf{T}) \subseteq W(\mathsf{T})$ and $\mathrm{keys}(\pi_\mathsf{T}) \subseteq W(\mathsf{T})$, it follows immediately that $\llbracket \mathbf{B}' \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} \pi_\mathsf{T} \sim_p \llbracket \mathbf{B} \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket}$ for all $p \notin W(\mathsf{T})$. It remains to show that $\llbracket \mathbf{B}' \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} \pi_\mathsf{T} \sim_{W(\mathsf{T})} \llbracket \mathbf{B} \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket}$. First note that, by Lemmas 15 and 16:

$$\llbracket \mathbf{B}_0 \rrbracket \sim_{R(\mathsf{T})} \llbracket \mathbf{B}' \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} \qquad \llbracket \mathbf{B}_0 \rrbracket \sim_{W(\mathsf{T})} \llbracket \mathbf{B}' \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket}$$

Then, by Lemma 17:
$$\llbracket \mathbf{B}' \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} \pi_\mathsf{T} \sim_{W(\mathsf{T})} \llbracket \mathbf{B} \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket}$$

And hence:
$$\llbracket \mathbf{B}' \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} \pi_\mathsf{T} = \llbracket \mathbf{B} \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket} \tag{4}$$

The thesis $\llbracket \mathbb{T} \rrbracket^{\Pi}_{\llbracket \mathbf{B}_0 \rrbracket} = \llbracket \mathbf{B} \rrbracket_{\llbracket \mathbf{B}_0 \rrbracket}$ follows by Equations (3) and (4). □

**Lemma 19.** *Every $W$ is compatible with $\Pi^\star$.*

*Proof.* Trivial. □

We now formalize when a blockchain $\mathbf{B}$ is a serialization of a multiset of transactions $\mathbb{T}$.

**Definition 13 (Serialization of multisets of transactions).** *We define the relation $\lhd$ between blockchains and multisets of transactions as follows:*

$$\frac{}{\epsilon \lhd []} \qquad\qquad \frac{\mathbf{B} \lhd \mathbb{T}}{\mathbf{B}\mathsf{T} \lhd ([\mathsf{T}] + \mathbb{T})}$$

The following theorem ensures that the parallel execution of strongly swappable transactions is equivalent to any sequential execution of them. Hereafter, we say that a multiset $\mathbb{T}$ is *strongly swappable* if $\forall \mathsf{T} \in \mathbb{T}, \forall \mathsf{T}' \in \mathbb{T} - [\mathsf{T}] : \mathsf{T} \# \mathsf{T}'$.

**Theorem 6.** *If $\mathbb{T}$ is strongly swappable and $\mathbf{B} \lhd \mathbb{T}$, then, for all reachable $\sigma$:* $[\![\mathbb{T}]\!]_\sigma^{\varPi^\star} = [\![\mathbf{B}]\!]_\sigma.$

*Proof.* Direct by Lemmas 18 and 19. □

A *parellelized blockchain* $\mathbb{B}$ is a finite sequence of multisets of transactions; we denote with $\epsilon$ the empty sequence. We extend the semantics of multisets (Definition 12) to parallelized blockchains as follows.

**Definition 14 (Semantics of parallelized blockchains).** *The semantics of parallelized blockchains is defined as follows:*

$$[\![\epsilon]\!]_\sigma^\varPi = \sigma \qquad [\![\mathbb{T}\mathbb{B}]\!]_\sigma^\varPi = [\![\mathbb{B}]\!]_{[\![\mathbb{T}]\!]_\sigma^\varPi}^\varPi$$

*We write $[\![\mathbb{B}]\!]^\varPi$ for $[\![\mathbb{B}]\!]_{\sigma^\star}^\varPi$, where $\sigma^\star$ is the initial state.*

We also extend the serialization relation $\lhd$ (Definition 13) to parallelized blockchains.

**Definition 15 (Serialization of parallelized blockchains).** *We define the relation $\lhd$ between blockchains and parallelized blockchains as follows:*

$$\frac{}{\epsilon \lhd \epsilon} \qquad \frac{\mathbf{B}_1 \lhd \mathbb{T} \quad \mathbf{B}_2 \lhd \mathbb{B}}{\mathbf{B}_1\mathbf{B}_2 \lhd \mathbb{T}\mathbb{B}}$$

The following theorem states that our technique to parallelize the transactions in a blockchain preserves its semantics.

**Theorem 7.** *If each multiset in $\mathbb{B}$ is strongly swappable and $\mathbf{B} \lhd \mathbb{B}$, then, for all reachable $\sigma$: $[\![\mathbb{B}]\!]_\sigma^{\varPi^\star} = [\![\mathbf{B}]\!]_\sigma.$*

*Proof.* By induction on the rule used for deriving $\mathbf{B} \lhd \mathbb{B}$.

– Rule: $\dfrac{}{\epsilon \lhd \epsilon}$.

The thesis follows trivially, since:

$$[\![\epsilon]\!]_\sigma = \sigma = [\![\epsilon]\!]_\sigma^{\varPi^\star}$$

– Rule: $\dfrac{\mathbf{B}_1 \lhd \mathbb{T} \quad \mathbf{B}_2 \lhd \mathbb{B}}{\mathbf{B}_1\mathbf{B}_2 \lhd \mathbb{T}\mathbb{B}}$.

By Theorem 6, for some reachable $\sigma'$:

$$[\![\mathbf{B}_1]\!]_\sigma = \sigma' = [\![\mathbb{T}]\!]_\sigma^{\varPi^\star}$$

By the induction hypothesis:

$$[\![\mathbf{B}_2]\!]_{\sigma'} = [\![\mathbb{B}]\!]_{\sigma'}^{\varPi^\star}$$

The thesis then follows by:

$$[\![\mathbf{B}_2]\!]_{\sigma'} = [\![\mathbf{B}_1\mathbf{B}_2]\!]_\sigma \qquad [\![\mathbb{B}]\!]_{\sigma'}^{\varPi^\star} = [\![\mathbb{T}\mathbb{B}]\!]_\sigma^{\varPi^\star} \qquad □$$

**Lemma 20.** *Let* $N_{\mathcal{R}}(T_1 \cdots T_n) = (P, Tr, F, m_0)$. *Then* $(Tr, <^*)$ *is a partial order.*

*Proof.* Transitivity and reflexivity hold by definition. For antisymmetricity, assume that $(T_i, i) <^* (T_j, j)$ and $(T_j, j) <^* (T_i, i)$. Then, it is easy to verify that $i \leq j$ and $j \leq i$, and so $i = j$. Since $T_i$ and $T_j$ are uniquely determined by $i$ and $j$, we have that $T_i = T_j$. Therefore, $(T_i, i) = (T_j, j)$, as required. $\square$

**Lemma 4.** $N_{\mathcal{R}}(\mathbf{B})$ *is an occurrence net, for all* $\mathcal{R}$ *and* $\mathbf{B}$.

*Proof.* By Figure 2, the first three conditions of the definition of occurrence net are easy to verify. To prove that $F^*$ is acyclic, we proceed by contradiction. Assume that there is a sequence $\boldsymbol{x} = x_0, x_1, \ldots x_m$ such that $(x_i, x_{i+1}) \in F$ for all $0 \leq i < m$, and $x_0 = x_m$ with $m > 0$. Note that the above sequence alternates between transitions and places, and so, since $m > 0$, at least one place and one transition occur in $\boldsymbol{x}$. Further, a place between two transitions $t \neq t'$ can exist only if $t < t'$. Therefore, if $t, t'$ occur in $\boldsymbol{x}$, it must be $t <^* t'$ and $t' <^* t$. So, if $\boldsymbol{x}$ contains at least two transitions, by Lemma 20, we have a contradiction. If only one transition $t = (T, i)$ occurs in $\boldsymbol{x}$, then there is a place of the form $(t, t)$ occuring in $\boldsymbol{x}$. Therefore, $t < t$, which implies $i < i$ — contradiction. $\square$

**Lemma 21.** *Let* $N = (P, Tr, F, m_0)$ *be an occurrence net. For all* $t, t' \in Tr$, *if* $t \neq t'$ *then* ${}^\bullet t \cap {}^\bullet t' = \emptyset$.

*Proof.* By contradiction, assume that $p \in {}^\bullet t \cap {}^\bullet t'$ with $t \neq t'$. Then, $\{t, t'\} \subseteq p^\bullet$, and hence $|p^\bullet| \geq 2$ — contradiction with constraint (i) of the definition of occurrence nets. $\square$

**Lemma 22.** *Let* $m$ *be a reachable marking of an occurrence net* $N$. *Then:*

1. *If* $m \xrightarrow{t} m'$ *and* $m \xrightarrow{t} m''$, *then* $m' = m''$ *(determinism).*
2. *If* $m \xrightarrow{t} m'$, $m \xrightarrow{t'} m''$ *and* $t \neq t'$, *then there exists* $m'''$ *such that* $m' \xrightarrow{t'} m'''$ *and* $m'' \xrightarrow{t} m'''$ *(diamond property).*
3. *If* $m \xrightarrow{t} \rightarrow^* \xrightarrow{t'}$ *then* $t \neq t'$ *(linearity).*
4. *If* $m \xrightarrow{t}^* m$ *then* $|t| = 0$ *(acyclicity).*

*Proof.* For Item 1, by definition of the firing of transitions of Petri Nets it must be $m' = m - {}^\bullet t + t^\bullet = m''$.

For Item 2, since $m \xrightarrow{t} m'$ and $m \xrightarrow{t'} m''$, it must be:

$$\begin{aligned} {}^\bullet t \subseteq m && m' = m - {}^\bullet t + t^\bullet \\ {}^\bullet t' \subseteq m && m'' = m - {}^\bullet t' + t'^\bullet \end{aligned}$$

By Lemma 21 it follows that $t'$ is enabled at $m'$, and $t$ is enabled at $m''$. Then, by definition of firing:

$$m' \xrightarrow{t'} m' - {}^\bullet t' + t'^\bullet \qquad \text{and} \qquad m'' \xrightarrow{t} m'' - {}^\bullet t + t^\bullet$$

Then:

$$
\begin{aligned}
m' - {}^\bullet t' + t'^\bullet &= (m - {}^\bullet t + t^\bullet) - {}^\bullet t' + t'^\bullet \\
&= (m - {}^\bullet t' + t'^\bullet) - {}^\bullet t + t^\bullet \qquad (\text{as } {}^\bullet t' \subseteq m) \\
&= m'' - {}^\bullet t + t^\bullet
\end{aligned}
$$

Hence, the thesis follows by choosing $m''' = m' - {}^\bullet t' + t'^\bullet$.

Item 3 follows directly by induction on the lenght of the reduction $\to^*$, exploiting the fact that $F^*$ is a partial order.

Item 4 follows by the fact that $F^*$ is a partial order. □

**Lemma 23.** *Let* $N = (P, Tr, F, m_0)$ *be an occurrence net, and let* $m$ *be a reachable marking, such that, for some* $t$, $m'$, $m''$:



*Then,* $m'' \to^n m'$.

*Proof.* By induction on $n$. For the base case, it must be $m \to^1 m'$, and hence $m \xrightarrow{t'} m'$ for some $t'$. Since $m \xrightarrow{t} m''$, by the contrapositive of item 2 of Lemma 22 (diamond property) it follows $t = t'$, and so, by item 1 of Lemma 22 (determinism) we have that $m' = m''$. Clearly:

$$
m'' \to^0 m'
$$

For the inductive case, let $n = m + 1$, for some $m$. Then, for some $t', m'''$:

$$
m \xrightarrow{t'} m''' \to^{m+1} m'
$$

If $t = t'$, then by item 1 of Lemma 22 (determinism) it follows that $m''' = m''$, and so we have the thesis $m'' \to^n m'$. Otherwise, if $t \neq t'$, by item 2 of Lemma 22 (diamond property), there must exists $m_1$ such that:

$$
m'' \xrightarrow{t'} m_1 \text{ and } m''' \xrightarrow{t} m_1
$$

We are in the following situation:

Since $m + 1 = n$, by the induction hypothesis:

$$m_1 \rightarrow^m m'$$

Therefore, we have the thesis

$$m'' \xrightarrow{t'} m_1 \rightarrow^m m' \diamond$$

**Lemma 24.** *Let* $(\mathsf{T}, i), (\mathsf{T}', j)$ *be transitions of* $\mathsf{N}_{\#_R^W}(\mathbf{B})$, *and let* $\mathsf{m}$ *be a reachable marking. Then:*

$$(\mathsf{T}, i) < (\mathsf{T}', j) \text{ and } \mathsf{m} \xrightarrow{(\mathsf{T}, i)} \implies \mathsf{m} \xrightarrow{(\mathsf{T}', j)} \not\rightarrow$$

*Proof.* By the construction in Figure 2, since $(\mathsf{T}, i) < (\mathsf{T}', j)$, then $\mathsf{p} = ((\mathsf{T}, i), (\mathsf{T}', j))$ is a place of the occurrence net, and $\mathsf{F}((\mathsf{T}, i), \mathsf{p}) = 1$ and $\mathsf{F}(\mathsf{p}, (\mathsf{T}', j)) = 1$. □

**Definition 16 (Independency).** *Let* $\mathsf{N}$ *be an occurrence net. We say that two transitions* $\mathsf{t}$ *and* $\mathsf{t}'$ *are* independent, *in symbols* $\mathsf{t} \, I \, \mathsf{t}'$, *if* $\mathsf{t} \neq \mathsf{t}'$ *and there exists a reachable marking* $\mathsf{m}$ *such that:*

$$\mathsf{m} \xrightarrow{\mathsf{t}} \text{ and } \mathsf{m} \xrightarrow{\mathsf{t}'}$$

*We define the relation* $\simeq$ *as the least congruence in the monoid* $\mathsf{Tr}^*$ *such that, for all* $\mathsf{t}, \mathsf{t}' \in \mathsf{Tr}$*:* $\mathsf{t} \, I \, \mathsf{t}' \implies \mathsf{tt}' \simeq \mathsf{t}'\mathsf{t}$.

**Lemma 25.** *Let* $\mathsf{N}$ *be an occurrence net, with a reachable marking* $\mathsf{m}$. *If* $\mathsf{m} \xrightarrow{\mathsf{U}}$ *then* $\mathsf{t} \, I \, \mathsf{t}'$, *for all* $\mathsf{t} \neq \mathsf{t}' \in \mathsf{U}$.

*Proof.* Since $\mathsf{m} \xrightarrow{\mathsf{U}}$, then $\mathsf{m} \xrightarrow{\mathsf{t}}$ for all $\mathsf{t} \in \mathsf{U}$. □

**Lemma 26.** *Let* $\mathsf{N}$ *be an occurrence net, and let* $\mathsf{m}$ *be a reachable marking. If* $\mathsf{m} \xrightarrow{\mathsf{t}_1}^* m'$ *and* $\mathsf{m} \xrightarrow{\mathsf{t}_2}^* m'$, *then* $\mathsf{t}_1 \simeq \mathsf{t}_2$.

*Proof.* We proceed by induction on the length of the longest reduction among $\mathsf{m} \xrightarrow{\mathsf{t}_1}^* m'$ and $\mathsf{m} \xrightarrow{\mathsf{t}_2}^* m'$. For the base case, the thesis is trivial as both $\mathsf{t}_1$ and $\mathsf{t}_2$ are empty. For the inductive case, assume that $\mathsf{t}_1$ is longer or equal to $\mathsf{t}_2$ (the other case is symmetric). Let $\mathsf{t}_1 = \mathsf{t}_1 \mathsf{t}_1'$. We first show that $\mathsf{t}_2$ is not empty. By contradiction, if $\mathsf{t}_2$ is empty, then $\mathsf{m} = m'$. But then, by item 4 of Lemma 22 (acyclicity) it follows that $\mathsf{t}_1$ is empty as well: contradiction. Therefore, $\mathsf{t}_2 = \mathsf{t}_2 \mathsf{t}_2'$ for some $\mathsf{t}_2$ and $\mathsf{t}_2'$. Clearly, $\mathsf{t}_1$ is longer than $\mathsf{t}_1'$ and $\mathsf{t}_2'$. Let $\mathsf{m} \xrightarrow{\mathsf{t}_1} m_1$ and $\mathsf{m} \xrightarrow{\mathsf{t}_2} m_2$. We have two subcases.

- If $\mathsf{t}_1 = \mathsf{t}_2$, by determinism (Lemma 22) it follows that $m_1 = m_2$. Let $m'' = m_1$. By the hypothesis of the lemma, we have $m'' \xrightarrow{\mathsf{t}_1'}^* m'$ and $m'' \xrightarrow{\mathsf{t}_2'}^* m'$. Then, by the induction hypothesis we have $\mathsf{t}_1' \simeq \mathsf{t}_2'$, and so the thesis $\mathsf{t}_1 \mathsf{t}_1' \simeq \mathsf{t}_2 \mathsf{t}_2'$ follows since $\simeq$ is a congruence.

– If $t_1 \neq t_2$, then by Definition 16 it must be $t_1$ I $t_2$. By the diamond property (Lemma 22), there exists $m''$ such that $m_1 \xrightarrow{t_2} m''$ and $m_2 \xrightarrow{t_1} m''$. By linearity (item 3 of Lemma 22), $m' \xrightarrow{t_1} \!\!\!\!/\,\,$ and $m' \xrightarrow{t_2} \!\!\!\!/\,\,$. By Lemma 23, applied on $m_2$, there exists $\mathbf{t}$ such that $m'' \xrightarrow{\mathbf{t}}^* m'$ and $|\mathbf{t}| + 1 = |\mathbf{t'_2}|$. So, we are in the following situation:
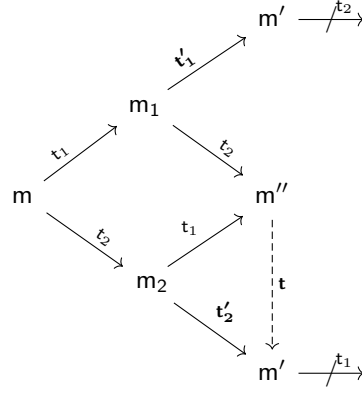


Therefore, we have that:

$$m_1 \xrightarrow{t_2}\xrightarrow{\mathbf{t}}^* m' \quad \text{and} \quad m_1 \xrightarrow{\mathbf{t'_1}}^* m'$$

$$m_2 \xrightarrow{t_1}\xrightarrow{\mathbf{t}}^* m' \quad \text{and} \quad m_2 \xrightarrow{\mathbf{t'_2}}^* m'$$

Note that $|t_2\mathbf{t}| = |t_1\mathbf{t}| = |\mathbf{t}| + 1 = |\mathbf{t'_2}| \leq |\mathbf{t'_1}| < |\mathbf{t_1}|$. Hence, by applying the induction hypothesis twice:

$$t_2\mathbf{t} \simeq \mathbf{t'_1} \quad \text{and} \quad t_1\mathbf{t} \simeq \mathbf{t'_2}$$

Then, since $\simeq$ is a congruence:

$$t_1 t_2 \mathbf{t} \simeq t_1 \mathbf{t'_1} \quad \text{and} \quad t_2 t_1 \mathbf{t} \simeq t_2 \mathbf{t'_2}$$

Since $t_1$ I $t_2$, then $t_1 t_2 \mathbf{t} \simeq t_2 t_1 \mathbf{t}$. By transitivity of $\simeq$:

$$\mathbf{t_1} \;=\; t_1 \mathbf{t'_1} \;\simeq\; t_1 t_2 \mathbf{t} \;\simeq\; t_2 t_1 \mathbf{t} \;\simeq\; t_2 \mathbf{t'_2} \;=\; \mathbf{t_2} \qquad\qquad \square$$

**Definition 17.** *For all sequences of transitions $\mathbf{t}$, we define the set $tr(\mathbf{t})$ of the transitions in $\mathbf{t}$ as:*
$$tr(\mathbf{t}) = \{t \mid \exists \mathbf{t_1}, \mathbf{t_2} : \mathbf{t} = \mathbf{t_1} t \mathbf{t_2}\}$$

*and we extend tr to step firing sequences $\mathbf{U}$ as follows:*

$$tr(\mathbf{U}) = \bigcup \{U \mid \exists \mathbf{U_1}, \mathbf{U_2} : \mathbf{U} = \mathbf{U_1} U \mathbf{U_2}\}$$

**Lemma 27.** *If $\mathbf{t} \simeq \mathbf{t'}$ then $tr(\mathbf{t}) = tr(\mathbf{t'})$.*

*Proof.* Trivial by Definition 16. □

**Lemma 28.** *Let* N *be an occurrence net, and let* m *be a reachable marking. If* m $\xrightarrow{U_1}$ $m_1$, m $\xrightarrow{U_2}$ $m_2$ *and* $tr(U_1) = tr(U_2)$, *then* $m_1 = m_2$.

*Proof.* Since m $\xrightarrow{U_1}$ $m_1$ and m $\xrightarrow{U_2}$ $m_2$, there exist sequentialisation $t_1$ of $U_1$ and $t_2$ of $U_2$ such that m $\xrightarrow{t_1}$ $m_1$ and m $\xrightarrow{t_2}$ $m_2$. Since by hypothesis $tr(U_1) = tr(U_2)$, then $tr(t_1) = tr(t_2)$. We proceed by induction on the lenght of $t_1$. The base case is trivial, as $m = m_1 = m_2$. For the inductive case, suppose $t_1 = t_1 t'_1$, with $|t'_1| = n$. By determinism, there exists a unique marking $m'_1$ such that m $\xrightarrow{t_1}$ $m'_1$ (a single step). Since $tr(U_1) = tr(U_2)$, it must be $t_2 = t_2 t'_2$, with $|t'_2| = n$. Let $m'_2$ be the unique marking such that m $\xrightarrow{t_2}$ $m'_2$ (a single step).

There are two subcases.

- If $t_1 = t_2$, it must be $m'_1 = m'_2$, and hence the thesis follows immediately by the induction hypothesis.
- If $t_1 \neq t_2$, by the diamond property (item 1 of Lemma 22), there exists m′ such that $m'_1 \xrightarrow{t_2} m'$ and $m'_2 \xrightarrow{t_1} m'$. Since $t_2 \in tr(t'_1)$, by linearity (item 3 of Lemma 22) it follows that $m_1 \xrightarrow{t_2} \not\!\!\!\rightarrow$, and hence, by applying Lemma 23 on $m'_1$ we obtain $m' \xrightarrow{t''_1} m_1$ for some $t''_1$. Summing up, we have that:

$$m'_1 \xrightarrow{t'_1} m_1 \qquad \text{and} \qquad m'_1 \xrightarrow{t_2 t''_1} m_1$$

Then, by Lemma 26, $t'_1 \simeq t_2 t''_1$, and hence:

$$t_1 = t_1 t'_1 \simeq t_1 t_2 t''_1$$

By Lemma 27:
$$tr(t_1) = tr(t_1 t_2 t''_1)$$

Similarly, we can conclude that $m' \xrightarrow{t''_2} m_2$ for some $t''_2$ and that:

$$tr(t_2) = tr(t_2 t_1 t''_2)$$

Since $tr(t_1) = tr(t_2)$, we can conclude:

$$tr(t''_1) = tr(t''_2)$$

Since $|t''_1| = n - 1 < n + 1 = |t_1|$, the thesis follows by the induction hypothesis. □

**Lemma 29.** *Let* $(T, i)$ *and* $(T', j)$ *be transitions of* $N_{\#_R^W}(B)$. *Then:*

$$(T, i) \text{ I } (T', j) \quad \implies \quad T \#_R^W T'$$

*Proof.* By Definition 16, $(\mathsf{T}, i)\; \mathrm{I}\; (\mathsf{T}', j)$ implies that $(\mathsf{T}, i) \neq (\mathsf{T}', j)$ and there exists some reachable marking $\mathsf{m}$ such that $\mathsf{m} \xrightarrow{(\mathsf{T},i)}$ and $\mathsf{m} \xrightarrow{(\mathsf{T}',j)}$. By contradiction, assume that $\neg(\mathsf{T} \#_R^W \mathsf{T}')$. Then, since $i < j$ or $j > i$, by Figure 2 we would have that $(\mathsf{T}, i) < (\mathsf{T}', j)$ or $(\mathsf{T}', j) < (\mathsf{T}, i)$. Then, by Lemma 24 we obtain a contradiction. $\qquad\square$

**Definition 18.** *Let* $\mathsf{N}_{\mathcal{R}}(\mathbf{B}) = (\mathsf{P}, \mathsf{Tr}, \mathsf{F}, \mathsf{m}_0)$. *We define the function* $\alpha : \mathsf{Tr} \to \mathbf{Tx}$ *as* $\alpha(\mathsf{T}, i) = \mathsf{T}$. *We then extend* $\alpha$ *to a function from steps to multisets of transactions as follows:*

$$\alpha(\emptyset) = [] \qquad \alpha(\mathsf{U} \cup \{\mathsf{t}\}) = [\alpha(\mathsf{t})] + \alpha(\mathsf{U})$$

*Finally, we extend* $\alpha$ *to finite sequences of steps as follows:*

$$\alpha(\epsilon) = \epsilon \qquad \alpha(\mathsf{U}\mathbf{U}) = \alpha(\mathsf{U})\alpha(\mathbf{U})$$

**Lemma 30.** *Let* $\mathsf{N}_{\mathcal{R}}(\mathbf{B}) = (\mathsf{P}, \mathsf{Tr}, \mathsf{F}, \mathsf{m}_0)$, *and let* $\mathbf{U}$ *be a step firing sequence. Then, for all* $\Pi, \sigma$:

$$[\![\mathbf{U}]\!]_\sigma^\Pi = [\![\alpha(\mathbf{U})]\!]_\sigma^\Pi$$

**Lemma 31.** *If* $\mathbf{t} \simeq \mathbf{t}'$ *in* $\mathsf{N}_{\#_R^W}(\mathbf{B})$, *then* $\alpha(\mathbf{t}) \simeq_{\#_R^W} \alpha(\mathbf{t}')$.

*Proof.* Define:

$$\simeq' = \left\{ (\mathbf{t}, \mathbf{t}') \,\middle|\, \alpha(\mathbf{t}) \simeq_{\#_R^W} \alpha(\mathbf{t}') \right\}$$

It suffice to show that $\simeq\; \subseteq\; \simeq'$. Note that $\simeq'$ is a congruence satisfying:

$$\mathsf{T} \#_R^W \mathsf{T}' \implies (\mathsf{T}, i)(\mathsf{T}', j) \simeq' (\mathsf{T}', j)(\mathsf{T}, i)$$

But then, by Lemma 29, it follows that $\simeq'$ also satisfies:

$$(\mathsf{T}, i)\; \mathrm{I}\; (\mathsf{T}', j) \implies (\mathsf{T}, i)(\mathsf{T}', j) \simeq' (\mathsf{T}', j)(\mathsf{T}, i) \tag{5}$$

Since $\simeq$ is the smallest congruence satisfying eq. (5), we have that $\simeq\; \subseteq\; \simeq'$, as required. $\qquad\square$

**Theorem 5.** *Let* $\mathbf{B} = \mathsf{T}_1 \cdots \mathsf{T}_n$. *Then, in* $\mathsf{N}_\#(\mathbf{B})$:

*(a) if* $\mathsf{m}_0 \xrightarrow{\mathbf{U}} \mathsf{m}$ *and* $\mathsf{m}_0 \xrightarrow{\mathbf{U}'} \mathsf{m}$, *then* $[\![\mathbf{U}]\!]_\sigma^{\Pi^\star} = [\![\mathbf{U}']\!]_\sigma^{\Pi^\star}$, *for all reachable* $\sigma$;
*(b)* $\{(\mathsf{T}_1, 1)\} \cdots \{(\mathsf{T}_n, n)\}$ *is a maximal step firing sequence;*
*(c) for all maximal step firing sequences* $\mathbf{U}$, *for all reachable* $\sigma$, $[\![\mathbf{U}]\!]_\sigma^{\Pi^\star} = [\![\mathbf{B}]\!]_\sigma$.

*Proof.* For item (a), assume that $\mathsf{m}_0 \xrightarrow{\mathbf{U}} \mathsf{m}$ and $\mathsf{m}_0 \xrightarrow{\mathbf{U}'} \mathsf{m}$. A standard result from Petri nets theory ensures that there exists sequentializations $\mathbf{t}$ of $\mathbf{U}$ and $\mathbf{t}'$ of $\mathbf{U}'$ such that:

$$\mathsf{m}_0 \xrightarrow{\mathbf{t}} \mathsf{m} \quad \text{and} \quad \mathsf{m}_0 \xrightarrow{\mathbf{t}'} \mathsf{m}$$

By Lemma 26, it must be $\mathbf{t} \simeq \mathbf{t}'$. Then, by Lemma 31:

$$\alpha(\mathbf{t}) \simeq_\# \alpha(\mathbf{t}')$$

By Lemma 11:
$$\llbracket \alpha(\mathbf{t}) \rrbracket_\sigma = \llbracket \alpha(\mathbf{t'}) \rrbracket_\sigma$$

By Lemmas 25 and 29, it follows that all multisets of transactions in $\alpha(\mathbf{U})$, as well as those in $\alpha(\mathbf{U'})$, are strongly swappable. Therefore, by Theorem 7:
$$\llbracket \alpha(\mathbf{U}) \rrbracket_\sigma^{\Pi^\star} = \llbracket \alpha(\mathbf{U'}) \rrbracket_\sigma^{\Pi^\star}$$

Then, by Lemma 30:
$$\llbracket \mathbf{U} \rrbracket_\sigma^{\Pi^\star} = \llbracket \mathbf{U'} \rrbracket_\sigma^{\Pi^\star}$$

For item (c), let $\mathbf{U'} = [\mathbf{t}_1] \ldots [\mathbf{t}_n]$, where $n = |\mathbf{B}|$. It is trivial to see that $\mathbf{U'}$ is maximal and that $\mathbf{B} \lhd \alpha(\mathbf{U'})$. By Theorem 7:
$$\llbracket \alpha(\mathbf{U'}) \rrbracket_\sigma^{\Pi^\star} = \llbracket \mathbf{B} \rrbracket_\sigma$$

Since $\mathbf{U}$ and $\mathbf{U'}$ are both maximal, by Lemma 28 and by item (a), it follows that:
$$\llbracket \mathbf{U'} \rrbracket_\sigma^{\Pi^\star} = \llbracket \mathbf{U} \rrbracket_\sigma^{\Pi^\star}$$

Since $\llbracket \mathbf{U'} \rrbracket_\sigma^{\Pi^\star} = \llbracket \alpha(\mathbf{U'}) \rrbracket_\sigma^{\Pi^\star}$ (by Lemma 30), and so:
$$\llbracket \mathbf{U} \rrbracket_\sigma^{\Pi^\star} = \llbracket \mathbf{B} \rrbracket_\sigma \qquad\qquad \square$$

## B  Full implementation of the ERC721 Token

```solidity
pragma solidity >= 0.4.2;

contract Token {

    mapping(uint256 => address) owner;
    mapping(uint256 => address) approved;
    mapping(uint256 => bool) exists;
    mapping(address => uint256) balance;
    mapping (address => mapping (address => bool))
        operatorApprovals;

    function ownerOf(uint256 tokenId) external view returns (
        address) {
        require (exists[tokenId]);
        require (owner[tokenId] != address(0));
        return owner[tokenId];
    }

    function balanceOf(address addr) external view returns (
        uint256) {
        require (addr != address(0));
        return balance[addr];
    }

    function approve(address addr, uint256 tokenId) external {
            require (exists[tokenId]);
            require (owner[tokenId] == msg.sender && addr != msg.
                sender);
            approved[tokenId] = addr;
    }

    function setApprovalForAll(address operator, bool isApproved)
        external {
        operatorApprovals[msg.sender][operator] = isApproved;
    }

    function getApproved(uint256 tokenId) external view returns (
        address) {
        require (exists[tokenId]);
        return approved[tokenId];
    }

    function isApprovedForAll(address addr, address operator)
        external view returns (bool) {
        return operatorApprovals[addr][operator];
    }

    function transferFrom(address from, address to, uint256
        tokenId) external {
        require (exists[tokenId]);
        require (from == owner[tokenId] && from != to);
        require (to != address(0));
```

```
        if (from == msg.sender || operatorApprovals[from][msg.
            sender] || approved[tokenId] == msg.sender) {
            owner[tokenId] = to;
            approved[tokenId] = address(0);
            balance[from] -= 1;
            balance[to] += 1;
        }
    }

    function mint(address to, uint256 tokenId) external {
        require (!exists[tokenId]);
        require (to != address(0));

        exists[tokenId] = true;
        owner[tokenId] = to;
        balance[to] += 1;
    }
}
```

We have the following safe approximations of read/written keys of the `Token` contract (assuming the straightforward generalization of the semantic domain of Section 2 to deal with multiple key-value stores:

$$
\begin{aligned}
P_r^1 &= \{\texttt{exists}[1], \texttt{owner}[1], \texttt{balance}[\mathcal{A}], \texttt{balance}[\mathcal{P}]\} \models^r \mathsf{T}_1 \\
P_w^1 &= \{\texttt{owner}[1], \texttt{balance}[\mathcal{A}], \texttt{balance}[\mathcal{P}]\} &\models^w \mathsf{T}_1 \\
P_r^2 &= \emptyset &\models^r \mathsf{T}_2 \\
P_w^2 &= \{\texttt{operatorApprovals}[\mathsf{A}][\mathsf{B}]\} &\models^w \mathsf{T}_2 \\
P_r^3 &= \{\texttt{exists}[2], \texttt{owner}[2], \texttt{balance}[\mathcal{A}], \texttt{balance}[\mathcal{Q}]\} \models^r \mathsf{T}_3 \\
P_w^3 &= \{\texttt{owner}[2], \texttt{balance}[\mathcal{A}], \texttt{balance}[\mathcal{Q}]\} &\models^w \mathsf{T}_3 \\
P_r^4 &= \{\texttt{exists}[1], \texttt{owner}[1], \texttt{balance}[\mathcal{P}], \texttt{balance}[\mathcal{B}]\} \models^r \mathsf{T}_4 \\
P_w^4 &= \{\texttt{owner}[1], \texttt{balance}[\mathcal{P}], \texttt{balance}[\mathcal{B}]\} &\models^w \mathsf{T}_4
\end{aligned}
$$