A theoretical basis for MEV

Massimo Bartoletti[1], Roberto Zunino[2]

[1] Università degli Studi di Cagliari, Cagliari, Italy
[2] Università di Trento, Trento, Italy

Abstract. Maximal Extractable Value (MEV) refers to a wide class of economic attacks to public blockchains, where adversaries with the power to reorder, drop or insert transactions in a block can "extract" value from smart contracts. Empirical research has shown that mainstream DeFi protocols are massively targeted by these attacks, with detrimental effects on their users and on the blockchain network. Despite the increasing real-world impact of these attacks, their theoretical foundations remain insufficiently established. We propose a formal theory of MEV, based on a general, abstract model of blockchains and smart contracts. Our theory is the basis for proofs of security against MEV attacks.

1 Introduction

Most blockchain protocols delegate the construction of blocks to consensus nodes that can freely pick users' transactions from the *mempool*, possibly add their own, and propose blocks containing these transactions in a chosen order. This arbitrariness in block construction can turn consensus nodes into adversaries, which exploit their transaction-ordering powers to maximize their gain at the expense of users. In the crypto jargon these attacks are referred to as "extracting" value, and the adversaries' gain is called *Maximal Extractable Value*, or MEV.

This issue is not purely theoretical: indeed, mainstream DeFi protocols like Automated Market Makers and Lending Pools are common targets of MEV attacks, which overall have led to attacks worth more than 1.2 billion dollars so far [1]. Notably, the profits derived from MEV attacks largely exceed those given by block rewards and transaction fees [22].

MEV attacks are so profitable that currently most Ethereum blocks proposals are due to centralized private relay networks that outsource the identification of MEV opportunities to anyone, and use their large networks of validators to include the MEV-extracting transactions in blocks [40,33]. While this systematic MEV extraction has some benefits (e.g., it has decreased transaction fees for users at the expense of MEV seekers [41]), it is detrimental to blockchain decentralization, transparency, and network congestion [34].

Given the practical relevance of MEV, various research efforts have focused on improving its understanding. Most approaches are preeminently empirical, and focus on heuristics to extract value from certain types of contracts [23,44,29,13,10], on the quantification of their impact in the wild [35,38,42,43,39], or on techniques to mitigate MEV attacks [18,16,26,17,21,19,11]. All these works, however, do not

answer one fundamental question: *what is MEV, exactly?* This contrasts with fundamental principles of modern cryptography, where formal definitions of security properties and of adversaries' powers are essential to the study of cryptographic schemes. In the absence of a rigorous definition, it is impossible to prove that a contract is *MEV-free*, i.e., secure w.r.t. MEV attacks. Formalizing MEV is challenging, as it requires a complex characterization of the adversary: (i) as an entity who can control the construction of blocks, where they can craft and insert their own transactions; (ii) whose actual identity and current wealth are immaterial w.r.t. MEV extraction. This complexity requires a comprehensive formalization of the adversary powers, knowledge, and their MEV attacks. Existing MEV definitions [9,37,31] are partial (e.g., they do not formalize adversarial knowledge), and they wrongly classify some types of contracts (see Section 4).

Contributions We summarise our contributions as follows:

- An abstract model of contracts, equipped with key economic notions like wealth and gain. We keep our model general in order to make it applicable to different blockchains and contract languages.
- An adversary model that includes a formalization of the adversaries' knowledge, i.e., the transactions that they can deduce by combining their private knowledge with that of the mempool. This improves over [9,37,31], where adversaries can craft blocks either by using their own knowledge or playing verbatim the transactions in the mempool, but cannot combine these two sources of information, so losing some potential attacks.
- A formal definition of the MEV extractable by a *given* set of users. We give this definition in two variants, depending on whether extracting MEV requires or not to exploit transactions in the mempool. This allows us to distinguish between the "legit" MEV that is intended in normal contract interactions, and the "bad" MEV that is not. To the best of our knowledge, this is the first work that attempts to separate "legit" MEV from "bad" MEV. We show that our definitions capture new attacks, not covered by [9,37,31].
- A formal definition of *universal* MEV, i.e., the maximal gain that can be achieved by *any* adversary, regardless of their actual identity and current wealth. Our MEV definition has a game-theoretic flavour: honest players try to minimize the damage, while adversaries try to maximize their gain. We support our notion through a theoretical study of its main properties, like monotonicity and finiteness;
- Proofs for MEV-freedom: we assess our MEV theory on real-world contracts such as crowdfunding, bounties, AMMs, and Lending Pools.

Overall, our formalization is a necessary first step towards the construction of analysis tools for the MEV-freedom of contracts. Because of space constraints, we provide our benchmark of use cases, additional results and the proofs of all our statements in separated appendices.

2 Blockchain model

We introduce below a formal model for reasoning about MEV. Aiming at generality and agnosticism of actual blockchains, rather than providing a concrete contract language we abstractly model contracts as state transition systems.

We assume a countably infinite set $\mathbb{A}$ of *actors* ($\mathtt{A}, \mathtt{B}, \ldots$), a set $\mathbb{X}$ of *transactions* ($\mathsf{X}, \mathsf{X}', \ldots$), and a set $\mathbb{T}$ of *token types* ($\mathtt{T}, \mathtt{T}', \ldots$). Actors are active entities, such as honest contract users or adversaries trying to extract MEV. We assume that tokens are *fungible*, i.e. units of the same type are interchangeable; NFTs are a special case of fungible tokens that are minted in a single indivisible unit.

We use calligraphic uppercase letters for sets (e.g., a set of actors $\mathcal{A}$, a set of transactions $\mathcal{X}$), and bold calligraphic uppercase for finite sequences (e.g., a sequence of transactions $\boldsymbol{\mathcal{X}}$). Pointwise sum of functions is denoted by $+$ and $\sum$.

We model the token holdings of a set of actors as a *wallet* $w : \mathbb{T} \to \mathbb{N}$, i.e., a map from token types to non-negative integers. A *wallet state* $W : \mathbb{A} \to (\mathbb{T} \to \mathbb{N})$ maps each actor to a wallet. We write $W(\mathcal{A})$ for $\sum_{\mathtt{A} \in \mathcal{A}} W(\mathtt{A})$, i.e., the pointwise addition of the wallets of the actors in $\mathcal{A}$. A *blockchain state* consists of a wallet state and a contract state. For the general results in this paper, the actual structure of contract states is immaterial, and therefore we do not specify it in Definition 1. When reasoning about specific contracts, we will make this structure explicit, including data and tokens. We model contracts as transition systems between blockchain states, with transitions triggered by transactions.

Definition 1 (Contract) *A contract is a triple made of:*

- $\mathbb{S} = \mathbb{C} \times \mathbb{W}$*, a set of* blockchain states*, where* $\mathbb{C}$ *is a set of* contract states*, and* $\mathbb{W}$ *is a set of wallet states;*
- $\mapsto : (\mathbb{S} \times \mathbb{X}) \to \mathbb{S}$*, a partial transition function that, given a blockchain state and a transaction, gives the next blockchain state;*
- $\mathbb{S}_0 \subseteq \mathbb{S}$*, a set of initial blockchain states.*

We denote by $\omega(S)$ *the wallet state of a blockchain state* S*, and with* $\omega_{\mathtt{A}}(S)$ *the wallet of an actor* $\mathtt{A}$ *in* S*, i.e.,* $\omega_{\mathtt{A}}(S) = \omega(S)(\mathtt{A})$.

A transaction X is *valid* in a state S if S has an outgoing transition $\mapsto$ labelled X. In real-world contract platforms like Ethereum, invalid transactions can be included in blocks, but have no effect on the state of the contract. To model this behaviour, we transform $\mapsto$ into a (deterministic and) total relation $\to : (\mathbb{S} \times \mathbb{X}^*) \to \mathbb{S}$ between blockchain states; $\to$ is labelled with *sequences* of transactions. For an empty sequence ε, we let $S \xrightarrow{\varepsilon} S$, i.e., doing nothing has no effect. For a non-empty sequence $\mathcal{Y}\mathsf{X}$, we let $S \xrightarrow{\mathcal{Y}\mathsf{X}} S'$ when either:

$$S \xrightarrow{\mathcal{Y}} S'' \text{ for some } S'', \text{ and } S'' \overset{\mathsf{X}}{\mapsto} S' \quad \text{or} \quad S \xrightarrow{\mathcal{Y}} S' \text{ and } \mathsf{X} \text{ is not valid in } S'$$

A state S is *reachable* if $S_0 \xrightarrow{\boldsymbol{\mathcal{X}}} S$ for some initial state S_0 and sequence $\boldsymbol{\mathcal{X}}$. We implicitly assume that all the states mentioned in our results are reachable.

Our model does not allow actors to freely exchange tokens, but this is not a limitation: if desired, this can be encoded in the contract transition function. This is coherent with how Ethereum handles e.g., ERC20 tokens, whose transfer capabilities are programmed in the contract.

Our model is quite general, and also includes some behaviors that are not meaningful in practice: e.g., there may be states where the total amount of tokens in actors' wallets is *infinite*. To rule out these cases, we require the following *finite tokens axiom*, ensuring that the overall amount of tokens in wallets is finite:[3]

$$\sum_{\mathtt{T}} W(\mathbb{A})(\mathtt{T}) \in \mathbb{N} \tag{1}$$

As a consequence of the axiom, $\omega_{\mathcal{A}}(S)$ has finite support for all $\mathcal{A}$ and S. Hereafter, we denote by $\mathbb{N}^{(\mathbb{T})}$ the set of finite-support functions from $\mathbb{T}$ to $\mathbb{N}$.

Measuring the effect of an attack to a contract requires to estimate the *wealth* of the adversary before and after the attack. To account for the fact that different token types can have different prices, we assume an additive function \$ that, given a wallet w, determines its *wealth* $\$w$.[4]

Definition 2 (Wealth) *We say that* $\$: \mathbb{N}^{(\mathbb{T})} \to \mathbb{N}$ *is a* wealth function *if* $\$(w_0 + w_1) = \$w_0 + \$w_1$ *holds for all* $w_0, w_1 \in \mathbb{N}^{(\mathbb{T})}$.

Let $\mathbf{1}_{\mathtt{T}}$ be the wallet that contains exactly one token of type $\mathtt{T}$. Then, we can write any wallet w as the (potentially infinite) sum:

$$w \;=\; \sum_{\mathtt{T}} w(\mathtt{T}) \cdot \mathbf{1}_{\mathtt{T}} \tag{2}$$

If w has finite support, then the sum in (2) has only a finite number of non-zero terms. From additivity of \$, it follows that the wealth of a (finite-support) w is the sum of the amount of each token $\mathtt{T}$ in w, times the *price* of $\mathtt{T}$, i.e., $\$\mathbf{1}_{\mathtt{T}}$:

$$\$w \;=\; \sum_{\mathtt{T}} \$\big(w(\mathtt{T}) \cdot \mathbf{1}_{\mathtt{T}}\big) \;=\; \sum_{\mathtt{T}} w(\mathtt{T}) \cdot \$\mathbf{1}_{\mathtt{T}} \tag{3}$$

We measure the success of a MEV attack in terms of *gain*, i.e., the difference of the attackers' wealth before and after the attack.

Definition 3 (Gain) *We define the gain of* $\mathcal{A}$ *upon performing a sequence* $\mathfrak{X}$ *of transactions from state* S *as* $\gamma_{\mathcal{A}}(S, \mathfrak{X}) = \$\omega_{\mathcal{A}}(S') - \$\omega_{\mathcal{A}}(S)$ *if* $S \xrightarrow{\mathfrak{X}} S'$.

The following proposition establishes some basic properties of gain. In particular, the maximal gain can always be extracted by a *finite* set of actors.

Proposition 1 *$\gamma_{\mathcal{A}}(S, \mathfrak{X})$ is always defined and has a finite integer value, given by $\gamma_{\mathcal{A}}(S, \mathfrak{X}) = \sum_{\mathtt{A} \in \mathcal{A}} \gamma_{\mathtt{A}}(S, \mathfrak{X})$. Furthermore, there exists $\mathcal{A}_0 \subseteq_{fin} \mathcal{A}$ (finite subset of $\mathcal{A}$) such that, for all $\mathcal{B}$, if $\mathcal{A}_0 \subseteq \mathcal{B} \subseteq \mathcal{A}$ then $\gamma_{\mathcal{A}_0}(S, \mathfrak{X}) = \gamma_{\mathcal{B}}(S, \mathfrak{X})$.*

[3] The finite tokens axiom applies only to the wallet state, while the tokens stored within the contract are unconstrained. Our theory works fine under this milder hypothesis, since to reason about MEV we do not need to count the tokens within the contract, but only the gain of actors.

[4] Note that $\mathtt{A}$'s wealth only depends on $\mathtt{A}$'s wallet, neglecting other parts of the state. As a result, actors with the same tokens have the same wealth, and wealth is insensitive to price fluctuations. This is because our notion of MEV is designed to capture attacks occurring in a *single* block. We discuss long-range attacks in Section 5.

3 Maximal Extractable Value

Formalizing MEV is challenging, because it requires a twofold characterization of the adversary as a set of actors with the ability to reorder, drop and insert transactions, and whose actual identity and wealth are immaterial to MEV extraction. We then find it convenient to divide our formalization into three steps:

1. We define the set of transactions $\kappa_{\mathcal{A}}(\mathcal{X})$ that actors $\mathcal{A}$ can *deduce* by combining their private knowledge with that of the mempool $\mathcal{X}$ (Def. 4).
2. We define the MEV of a *given* set of actors $\mathcal{A}$ in a state S and mempool $\mathcal{X}$ as the *maximal gain* that $\mathcal{A}$ can achieve from S by firing a sequence of transactions in their deducible knowledge $\kappa_{\mathcal{A}}(\mathcal{X})$ (Def. 5). We also provide a variant, dubbed "bad" MEV, which models the case where the attack is only possible by exploiting transactions in the mempool.
3. We define the *universal* MEV in a state S and mempool $\mathcal{X}$ as the MEV that an *arbitrary* set of actors can achieve regardless of identity, only assuming they have (or can buy) the tokens needed to carry out the attack (Def. 9).

3.1 Adversary model

Given a mempool $\mathcal{X}$, adversaries $\mathcal{A}$ can craft new transactions by mauling the transactions data in $\mathcal{X}$ (e.g., method arguments in Ethereum transactions). To this goal, $\mathcal{A}$ can reuse any piece of data in $\mathcal{X}$, with the constraints that they cannot forge signatures, and cannot deduce any value that is not efficiently computable from the previous ones (e.g., inverting a hash).

In our abstract model, we generalize this inference with an axiomatization of the set $\kappa_{\mathcal{A}}(\mathcal{X})$ of transactions deducible by the adversary $\mathcal{A}$ from a given mempool $\mathcal{X}$. We start by requiring ***extensivity***, ***idempotence*** and ***monotonicity*** on $\mathcal{X}$, so making $\kappa_{\mathcal{A}}(\cdot)$ an upper closure operator for any $\mathcal{A}$. In particular, these axiom imply that $\kappa_{\mathcal{A}}(\mathcal{X})$ include all the transactions in the mempool $\mathcal{X}$, and that larger mempools lead to larger adversarial inferences. The ***continuity*** axiom is a standard structural requirement: in our theory, it is pivotal to prove that MEV can always be extracted from a finite mempool. The ***finite causes*** axiom ensures that any *finite* set of transactions can be deduced by a *finite* set of actors that only use their private knowledge. Abstractly, the private knowledge is the set of transactions deducible from an empty mempool $\mathcal{X}$ (in practice, this corresponds to the set of transactions that $\mathcal{A}$ can craft by using their private keys). The ***private knowledge*** axiom states that a larger private knowledge requires a larger set of actors: hence, if two sets of actors can deduce exactly the same transactions, then they must be equal. Finally, the ***no shared secrets*** axiom formalises a separation between the private knowledge of different actors: namely, it implies that if a transaction can be deduced by two disjoint sets of actors, then it can be deduced by anyone.

Definition 4 (Transaction deducibility) *We say that* $\kappa : 2^{\mathbb{A}} \times 2^{\mathbb{X}} \to 2^{\mathbb{X}}$ *is a transaction deducibility function if it satisfies the following axioms:*

```
contract BadHTLC {
  commit(a pays 1:T,b,c) { // a must send 1:T to the contract
    require balance(T)==1; commitment=c; // prevents multiple commits
  }
  reveal(a sig,y) { // a must sign the transaction and reveal the secret y
    require balance(T)>0 && H(y)==commitment; // y must be a preimage
    transfer(a,balance(T):T); // send all T balance to a
  }
  timeout(a sig,Oracle sig) { // a and Oracle must sign the transaction
    require balance(T)>0; // the contract must have some tokens T
    transfer(a,balance(T):T); // send all T balance to a
  }
}
```

Fig. 1. A Hash Time Locked Contract.

Extensivity $\mathcal{X} \subseteq \kappa_{\mathcal{A}}(\mathcal{X})$
Idempotence $\kappa_{\mathcal{A}}(\kappa_{\mathcal{A}}(\mathcal{X})) = \kappa_{\mathcal{A}}(\mathcal{X})$
Monotonicity *if* $\mathcal{A} \subseteq \mathcal{A}'$, $\mathcal{X} \subseteq \mathcal{X}'$, *then* $\kappa_{\mathcal{A}}(\mathcal{X}) \subseteq \kappa_{\mathcal{A}'}(\mathcal{X}')$
Continuity *for all chains* $\mathcal{X}_0 \subseteq \mathcal{X}_1 \subseteq \mathcal{X}_2 \subseteq \cdots$, $\kappa_{\mathcal{A}}(\bigcup_{i\in\mathbb{N}}\mathcal{X}_i) = \bigcup_{i\in\mathbb{N}}\kappa_{\mathcal{A}}(\mathcal{X}_i)$
Finite causes $\forall \mathcal{X}_0 \subseteq_{fin} \mathbb{X}.\ \exists \mathcal{A}_0 \subseteq_{fin} \mathbb{A}.\ \mathcal{X}_0 \subseteq \kappa_{\mathcal{A}_0}(\emptyset)$
Private knowledge *if* $\kappa_{\mathcal{A}}(\emptyset) \subseteq \kappa_{\mathcal{A}'}(\emptyset)$, *then* $\mathcal{A} \subseteq \mathcal{A}'$
No shared secrets $\kappa_{\mathcal{A}}(\mathcal{X}) \cap \kappa_{\mathcal{B}}(\mathcal{X}) \subseteq \kappa_{\mathcal{A}\cap\mathcal{B}}(\mathcal{X})$

We illustrate $\kappa_{\mathcal{A}}(\mathcal{X})$ through an example. For simplicity, in the contract language used in our examples we drop all the features of Solidity that are inessential to the understanding of MEV; still, the language is expressive enough to express real-world use cases like those found in DeFi (see Appendix B). Our contract language has a formal semantics (see Appendix A). While this semantics is crucial to prove the presence or absence of MEV in our benchmark of use cases, for now it will be sufficient to rely on its intuitive understanding.

Example 1. The `BadHTLC` contract in Figure 1 implements a Hash-Time Locked Contract, where a committer promises that she will either reveal a secret within a certain deadline, or pay a penalty of 1:`T` to anyone after the deadline. The procedure `commit` initialises the contract state, setting the variable `commitment`. The parameter "`a pays` 1:`T`" asks any `a` (who becomes the committer) to deposit 1:`T` into the contract. The procedure `reveal` allows anyone to redeem the deposit by revealing the secret: there, the parameter "`a sig`" requires the transaction to be signed by `a`; the command `transfer(a,balance(T):T)` transfers to `a` all the tokens `T` stored in the contract. Dually, `timeout` allows anyone to redeem the deposit after the deadline, triggered by a time oracle who signs the transaction.

We anticipate that `BadHTLC` suffers from a MEV attack in a state where the secret has been committed but not revealed yet, like e.g.:

$$S \;=\; \texttt{A}[0{:}\texttt{T}] \mid \texttt{BadHTLC}[1{:}\texttt{T}, \texttt{commitment} = H(s)] \mid \cdots \tag{4}$$

In state S, `A` has 0:`T` and `BadHTLC` has 1:`T`. Suppose the mempool $\mathcal{X}$ contains a transaction $\mathsf{X}_{\texttt{A}} = \texttt{reveal}(\texttt{A sig}, s)$ sent by `A` to redeem the deposit. Since the secret s is public in the mempool, any adversary `M` can craft a transaction $\mathsf{X}_{\texttt{M}} =$

$\texttt{reveal}(\texttt{M sig}, s)$ by combining their own knowledge (to provide $\texttt{M}$'s signature) with that of $\mathcal{X}$ (to provide s). Therefore, $\mathsf{X}_{\texttt{M}} \in \kappa_{\{\texttt{M}\}}(\mathcal{X})$, and so $\texttt{M}$ can extract MEV by front-running $\mathsf{X}_{\texttt{A}}$ with $\mathsf{X}_{\texttt{M}}$. Indeed, we have that:

$$S \xrightarrow{\mathsf{X}_{\texttt{M}}} \texttt{A}[0\!:\!\texttt{T}] \mid \texttt{M}[1\!:\!\texttt{T}] \mid \texttt{BadHTLC}[0\!:\!\texttt{T}, \cdots] \mid \cdots$$

Note instead that $\texttt{M}$ *alone* cannot deduce the transaction that would allow her to trigger the timeout. Formally, $\mathsf{Y}_{\texttt{M}} = \texttt{timeout}(\texttt{M sig}, \texttt{Oracle sig}) \notin \kappa_{\{\texttt{M}\}}(\mathcal{X})$. Crafting $\mathsf{Y}_{\texttt{M}}$ is not possible without the cooperation of the oracle, even if $\texttt{M}$ knows a transaction $\texttt{timeout}(\texttt{B sig}, \texttt{Oracle sig})$ from a past interaction, since each signature is tied to a specific transaction. We remark that this attack requires the adversary to combine private and mempool knowledge, which does not seem to be properly accounted for in current MEV formalizations [9,37,31]. ◇

Proposition 2 establishes some key properties of κ, which will be instrumental to prove more complex properties about MEV. Item (1) states that different sets of actors have a different private knowledge. Item (2) implies that transactions that can be deduced by disjoint sets of actors can be deduced by anyone. Item (3) states that two groups of actors joining forces could infer more transactions than they could independently infer. Item (4) states that a group $\mathcal{B}$ that can exploit both a mempool $\mathcal{X}$ and the inference of a larger group $\mathcal{A}$ on a smaller mempool $\mathcal{Y}$ cannot infer more transactions than $\mathcal{A}$ infer from $\mathcal{X}$. Remarkably, Item (5) rules out the case where, to deduce a transaction X, a set of actors needs to combine knowledge from an *infinite* mempool $\mathcal{X}$ (the proof exploits the continuity of κ).

Proposition 2 *For all $\mathcal{A}$, $\mathcal{B}$, $\mathcal{X}$ and $\mathcal{Y}$, we have that:*

(1) *if* $\mathcal{A} \neq \mathcal{B}$, *then* $\kappa_{\mathcal{A}}(\emptyset) \neq \kappa_{\mathcal{B}}(\emptyset)$
(2) $\kappa_{\mathcal{A}}(\mathcal{X}) \cap \kappa_{\mathcal{B}}(\mathcal{X}) = \kappa_{\mathcal{A}\cap\mathcal{B}}(\mathcal{X})$
(3) $\kappa_{\mathcal{A}}(\mathcal{X}) \cup \kappa_{\mathcal{B}}(\mathcal{X}) \subseteq \kappa_{\mathcal{A}}(\kappa_{\mathcal{B}}(\mathcal{X})) \subseteq \kappa_{\mathcal{A}\cup\mathcal{B}}(\mathcal{X})$
(4) *if* $\mathcal{B} \subseteq \mathcal{A}$ *and* $\mathcal{Y} \subseteq \mathcal{X}$, *then* $\kappa_{\mathcal{B}}(\kappa_{\mathcal{A}}(\mathcal{Y}) \cup \mathcal{X}) \subseteq \kappa_{\mathcal{A}}(\mathcal{X})$
(5) $\forall \mathsf{X} \in \kappa_{\mathcal{A}}(\mathcal{X}).\ \exists \mathcal{X}_0 \subseteq_{\mathit{fin}} \mathcal{X}.\ \mathsf{X} \in \kappa_{\mathcal{A}}(\mathcal{X}_0)$

3.2 MEV extractable by a given set of actors

The axiomatization of adversarial knowledge is the core of our MEV definition. Namely, the MEV of a *given* set of actors $\mathcal{A}$ is the maximal gain that $\mathcal{A}$ can achieve by firing a sequence of transactions *deducible* by $\mathcal{A}$ using their private knowledge and that of the mempool.

Definition 5 (MEV) *The MEV extractable* by a set of actors $\mathcal{A}$ *from a mempool $\mathcal{X}$ in a state S is given by:*

$$\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) = \max\left\{\gamma_{\mathcal{A}}(S, \boldsymbol{\mathcal{Y}}) \,\middle|\, \boldsymbol{\mathcal{Y}} \in \kappa_{\mathcal{A}}(\mathcal{X})^*\right\} \tag{5}$$

By allowing $\mathcal{A}$ to fire arbitrary bundles in $\kappa_{\mathcal{A}}(\mathcal{X})^*$ (the set of finite sequences of transactions in $\kappa_{\mathcal{A}}(\mathcal{X})$), we are actually empowering $\mathcal{A}$ with the ability to

reorder, drop and insert transactions. This is coherent with the practice, where miners/validators are commonly in charge for assembling blocks.[5]

Note that the max in (5) may not exist: for instance, consider a contract where each transaction (fireable by anyone) increases by $1 : \mathtt{T}$ the tokens in $\mathtt{A}$'s wallet. The wallet states reachable in this contract satisfy the finite tokens axiom (1), but the MEV of $\mathtt{A}$ is unbounded, because for each fixed n, there exists a reachable state where $\mathtt{A}$'s gain is greater than n. A sufficient condition for the existence of the max in (5) is that, in any reachable state, the wealth of all actors is bounded by a constant:

$$\forall S_0 \in \mathbb{S}_0.\ \exists n.\ \forall S.\ \ S_0 \to \cdots \to S \implies \$\omega_{\mathbb{A}}(S) < n \tag{6}$$

Hereafter, we assume that contracts satisfy (6), namely they are \$-*bounded*. We now establish some key properties of MEV. First, MEV is always defined for \$-bounded contracts.

Proposition 3 $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X})$ *is defined and has a non-negative value.*

The MEV is preserved by removing from $\mathcal{X}$ all the transactions that the actors $\mathcal{A}$ can generate by themselves:

Proposition 4 $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) = \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X} \setminus \kappa_{\mathcal{A}}(\emptyset))$

MEV is monotonic w.r.t. the mempool $\mathcal{X}$. This follows from the monotonicity of transactions deducibility κ, since a wider knowledge gives more opportunities to increase one's gain.

Proposition 5 *If* $\mathcal{X} \subseteq \mathcal{X}'$, *then* $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) \leq \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}')$.

Perhaps surprisingly, MEV is *not* monotonic w.r.t. the set $\mathcal{A}$ of actors who are extracting it. For instance, if $\mathtt{A}$ has a positive gain by firing a transaction that yields a negative opposite gain for $\mathtt{B}$ (and $\mathtt{B}$ has no other ways to have a positive gain), then the MEV of $\{\mathtt{A}\}$ is positive, while that of $\{\mathtt{A}, \mathtt{B}\}$ is zero.

In general, MEV is *not* even monotonic w.r.t. the amount of tokens in wallets, i.e., being richer does not always increase one's ability of extracting MEV. In particular, $\mathtt{A}$ might be able to extract MEV in a state $\mathtt{A}[w] \mid S$, but not in a state $\mathtt{A}[w + w_\Delta] \mid S$. For instance, this may happen when the contract enables the MEV-extracting transaction only in states containing an exact number of tokens in users' wallets. In fact, in most real-world contracts the effect of transactions never depends on tokens which are not controlled by the contract. We formalise this property of contracts by requiring that each transaction enabled in a certain wallet state W produces the same effect in a "richer" wallet state $W + W_\Delta$.

Definition 6 (Wallet-monotonic contract) *A contract is* wallet-monotonic *if for all* W, W', W_Δ, $\mathtt{C}$, $\mathtt{C}'$ *and* $\mathtt{X}$*:*

$$(W, \mathtt{C}) \overset{\mathtt{X}}{\mapsto} (W', \mathtt{C}') \implies (W + W_\Delta, \mathtt{C}) \overset{\mathtt{X}}{\mapsto} (W' + W_\Delta, \mathtt{C}')$$

[5] Some blockchain networks instead do not allow the current leader node to propose a block, but use special protocols that ensure a fair ordering of transactions [28,30,36].

This naturally extends to sequences of valid transactions. For this class of contracts, MEV is monotonic w.r.t. wallets.

Proposition 6 *Let $S = (W, \mathtt{C})$ and let $S_\Delta = (W + W_\Delta, \mathtt{C})$. If the contract is wallet-monotonic, then* $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) \leq \mathrm{MEV}_{\mathcal{A}}(S_\Delta, \mathcal{X})$.

In general, a single actor could not be able to extract MEV, since the contract could require the interaction between multiple actors in order to trigger a payment. Proposition 7 shows that the (maximal!) MEV can always be obtained by a *finite* set of actors.

Proposition 7 *For all $\mathcal{A}$, $\mathcal{X}$ and S, there exists $\mathcal{A}_0 \subseteq_{fin} \mathcal{A}$ such that, for all $\mathcal{B}$, if $\mathcal{A}_0 \subseteq \mathcal{B} \subseteq \mathcal{A}$ then:* $\mathrm{MEV}_{\mathcal{A}_0}(S, \mathcal{X}) = \mathrm{MEV}_{\mathcal{B}}(S, \mathcal{X})$.

We also show that MEV can always be extracted from a *finite* mempool. This follows from the continuity of κ, and in particular from its consequence Proposition 2(5), which ensures that each transaction in the sequence used to obtain the max gain can be deduced from a finite subset of the mempool.

Proposition 8 (Mempool finiteness) *For all $\mathcal{A}$, $\mathcal{X}$ and S, there exists some $\mathcal{X}_0 \subseteq_{fin} \mathcal{X}$ such that* $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}_0) = \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X})$.

Not all MEV is always considered an attack: e.g., the MEV that derives from arbitrage on AMMs and liquidations on Lending Pools is rather considered an incentive for users to keep the contract aligned with its ideal functionality. In these cases, the MEV is extracted without using the mempool. To isolate the part of MEV that is agreeably considered an attack, we remove from the overall $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X})$ the part that can be extracted without knowledge of the mempool, i.e., $\mathrm{MEV}_{\mathcal{A}}(S, \emptyset)$. We dub this new notion as "bad MEV".

Definition 7 (Bad MEV) *The "bad MEV" extractable* by a set of actors $\mathcal{A}$ *from a mempool $\mathcal{X}$ in a state S is given by:*

$$\mathrm{MEV}^{\mathrm{bad}}{}_{\mathcal{A}}(S, \mathcal{X}) = \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) - \mathrm{MEV}_{\mathcal{A}}(S, \emptyset) \tag{7}$$

Proposition 9 *All the previous results about* MEV*, except Proposition 6, also hold for* $\mathrm{MEV}^{\mathrm{bad}}$*. Furthermore,* $\mathrm{MEV}^{\mathrm{bad}}{}_{\mathcal{A}}(S, \mathcal{X}) \leq \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X})$.

3.3 Universal MEV

Definition 5 parameterises MEV over a set of actors $\mathcal{A}$. In this way, the same state $(S, \mathcal{X})$ could admit different MEVs for different sets of actors. This dependency on the set of actors contrasts with the practice, where the actual identity of miners or validators is unrelated to their ability to extract MEV.

For instance, consider the `Whitelist` contract in Figure 2. In any state S where `M` has at least $1 : \mathtt{T}$ and the contract has $n : \mathtt{T}$ with $n > 0$, any set of actors $\mathcal{B}$ including `A` has MEV. More precisely, $\mathrm{MEV}_{\mathcal{B}}(S, \mathcal{X}) = n \cdot \$\mathbf{1}_{\mathtt{T}}$. However, this way of extracting MEV is *not* considered an attack in practice, since the

```
contract Whitelist {
  pay(a pays 1:T) { require a==A; transfer(a,balance(T):T); }
}
contract Blacklist {
  pay(a pays 1:T) { require a!=A; transfer(a,balance(T):T); }
}
```

Fig. 2. `Whitelist` and `Blacklist` contracts.

```
contract Bank {
  deposit(a pays amt:T) { // a sends amt:T to the contract
    if acct[a]==null then acct[a]=amt else acct[a]=acct[a]+amt
  }
  xfer(a sig,amt,b) { // a transfers amt:T to b
    require acct[a]>=amt && acct[b]!=null && amt>0;
    acct[a]=acct[a]-amt; acct[b]=acct[b]+amt
  }
  wdraw(a sig,amt) { // a withdraws amt:T
    require acct[a]>=amt && amt>0;
    acct[a]=acct[a]-amt; transfer(a,amt:T);
  }
}
```

Fig. 3. A `Bank` contract.

recipient of the tokens is not arbitrary, but an actor (`A`) who is *hard-coded* in the contract. By contrast, the contract `Blacklist` is attackable, provided that the adversary $\mathcal{B}$ includes some $\mathtt{M} \neq \mathtt{A}$ who has at least $1:\mathtt{T}$. The fact that the hard-coded actor `A` cannot extract MEV is irrelevant, since the adversary can easily create a pseudonym which is different from the blacklisted ones.

As another example of a non-attack, consider the `Bank` contract in Figure 3, which allows users to deposit and withdraw tokens, and to transfer them to others. Let S be a state where `A` has deposited $n:\mathtt{T}$ in the contract, while the balances of the other users are zero. We have that $\mathrm{MEV}_{\{\mathtt{A}\}}(S, \emptyset) = n \cdot \$\mathbf{1}_{\mathtt{T}}$ and in general any set of actors including `A` can extract MEV. However, even this way of extracting MEV is *not* considered an attack, since any adversary $\mathcal{B}$ not including `A` has $\mathrm{MEV}_{\mathcal{B}}(S, \mathcal{X}) = 0$, for every mempool $\mathcal{X}$ (unless $\mathcal{X}$ includes an explicit `xfer` from `A` to $\mathcal{B}$). Unlike in the `Blacklist` example, the `Bank` has no hard-coded names, but some names become bound in the contract states upon transactions. For instance, after `A` has deposited $n:\mathtt{T}$, we have $\mathtt{Bank}[n:\mathtt{T}, \mathtt{acct} = \{\mathtt{A} \mapsto n\}]$.

In general, if the ability to extract MEV is subject to the existence of specific actors in the set $\mathcal{A}$, this is *not* considered an attack. Even when the identity of actors is immaterial, the amount of tokens in their wallets is not: e.g., actors may lose the capability of extracting MEV when spoiled from their tokens. Therefore, we consider as universal MEV the max gain that can be extracted by the adversary after a suitable *redistribution* of tokens in wallets. To stay on the safe side, we always consider the optimal token redistribution for the adversary. In this way, we never consider a state S as MEV-free just because the adversary has not enough tokens in S to carry the attack: this would be unsafe, since the attack could have been possible in a state S' where the adversary has redistributed the

tokens in their wallets. Formally, a *token redistribution* is a relation $S \approx_\$ S'$ which holds when all the tokens in the wallets in S are reassigned in S'.

Definition 8 (Token redistribution) *Let $S = (W, \mathtt{C})$, $S' = (W', \mathtt{C}')$. We write $S \approx_\$ S'$ when $W(\mathbb{A}) = W'(\mathbb{A})$ and $\mathtt{C} = \mathtt{C}'$.*

We now formalise the key notion of *universal MEV*. States with no universal MEV are called *MEV-free*.

Definition 9 (Universal MEV) *The* universal MEV *extractable from a mempool $\mathcal{X}$ in a state S is given by:*

$$\mathrm{MEV}(S, \mathcal{X}) = \min_{\mathcal{B}\ \mathit{cofinite}} \max_{\substack{\mathcal{A} \subseteq \mathcal{B} \\ S \approx_\$ S'}} \mathrm{MEV}_{\mathcal{A}}(S', \mathcal{X}) \tag{8}$$

We say that $(S, \mathcal{X})$ is MEV-free *when $\mathrm{MEV}(S, \mathcal{X}) = 0$. We define $\mathrm{MEV}^{\mathrm{bad}}(S, \mathcal{X})$ and $\mathrm{MEV}^{\mathrm{bad}}$-free similarly.*

To ensure that the identities of actors extracting MEV are immaterial, in (8) we take the *minimum* w.r.t. all sets $\mathcal{B}$ of actors. We restrict to *infinite* sets $\mathcal{B}$ to grant the attacker an unbounded amount of fresh identities, which can be used to avoid the ones handled in a special way by the contract. More specifically, since real-world contracts treat, in each state, only a finite number of actors as special, we let $\mathcal{B}$ range over *cofinite* sets. Once the set $\mathcal{B}$ of adversaries is fixed, we take the *maximum* MEV of $\mathcal{A} \subseteq \mathcal{B}$ w.r.t. all the possible token redistributions. In this way, we ensure that the adversary has enough tokens to carry the attack, if any. Note that (8) follows the *minimax* principle of game theoretic definitions. Intuitively, the honest players choose $\mathcal{B}$ in the min so to prevent the adversary from using privileged identities. Then, the adversary chooses, in the max, the identities $\mathcal{A}$ from $\mathcal{B}$ which are actually used for the attack: this allows the adversary to remove the actors with negative gain.

The universal MEV is always defined under a strengthened \$-boundedness assumption which considers token redistributions:

$$\forall S_0 \in \mathbb{S}_0.\ \exists n.\ \forall S.\ \ S_0 \to_{\approx_\$} \cdots \to_{\approx_\$} S \implies \$\omega_{\mathbb{A}}(S) < n \tag{9}$$

where the relation $\to_{\approx_\$}$ allows to redistribute tokens at each step.

Proposition 10 *For all $\mathcal{X}$ and S satisfying (9), $\mathrm{MEV}(S, \mathcal{X})$ and $\mathrm{MEV}^{\mathrm{bad}}(S, \mathcal{X})$ are defined and have a non-negative value.*

The following two theorems establish that universal MEV is monotonic with respect to mempools and wallets.

Theorem 1 *If $\mathcal{X} \subseteq \mathcal{X}'$ then $\mathrm{MEV}(S, \mathcal{X}) \leq \mathrm{MEV}(S, \mathcal{X}')$ (similarly for $\mathrm{MEV}^{\mathrm{bad}}$).*

Theorem 2 *Let $S = (W, \mathtt{C})$ and let $S + W_\Delta$ be $(W + W_\Delta, \mathtt{C})$. If the contract is wallet-monotonic, then: $\mathrm{MEV}(S, \mathcal{X}) \leq \mathrm{MEV}(S + W_\Delta, \mathcal{X})$.*

Table 1. MEV analysis of our benchmark of contracts.

Contract	MEV-free?	MEV^{bad}-free?	Contract	MEV-free?	MEV^{bad}-free?
Bad HTLC	✗	✗	Crowdfund (B.3)	✓	✓
HTLC (B.1)	✓	✓	AMM (B.4)	✗	✗
Whitelist	✓	✓	Price Bet ([9])	✗	✓
Blacklist	✗	✓	Naïve bounty (B.5)	✗	✗
Bank	✓	✓	Bounty (B.5)	✓	✓
Coin Pusher (B.2)	✗	✗	Lending Pool (B.6)	✗	✗

3.4 Proving MEV-freedom

We now apply our theory to study MEV-freedom of contracts. Because of space constraints, we show here just a few examples. More complex contracts, including crowdfunding, bounties, AMMs, and Lending Pools are analyzed in Appendix B. Here we just summarize the results of our analysis in Table 1. In the column "MEV-free?", we mark with a ✗ those cases where a (universal) MEV attack exists with respect to some state and mempool, and with ✓ when no such attack exist. The column "MEV^{bad}-free?" uses a similar notation.

The ✗ in the AMM contract is witnessed by a sandwich attack in a state where the AMM is in balanced state (i.e., the internal exchange rate equals to that of an external page oracle). This is a case of "bad" MEV, since the attack exploits the mempool. When the AMM is unbalanced, there is a case of "legit" MEV, i.e., anyone can perform arbitrage and have a positive MEV with an empty mempool. The ✗ in the Lending Pool contract is witnessed by the attack that exploits a mempool transaction that updates the interest rate. The attacker back-runs this transaction in a state where some borrowers become undercollateralized, and so liquidates part of their debt, obtaining their minted tokens with a bonus. The Price Bet contract in [9] shows an interesting case where (universal) MEV is extractable without exploiting the mempool. In this contract, a player bets on the future exchange rate between two tokens, determined through an AMM that is used as a price oracle. The attacker bets on a given price, then unbalances the AMM to obtain the desired exchange rate, and finally re-balances the AMM. In this way, the attacker can win the bet, extracting MEV. Note that this attack is possible whenever in the state there are enough tokens to unbalance the AMM as required. The Bounty contract, which rewards the first user who submits the solution to a puzzle, is a paradigmatic case where a naïve implementation leads to "bad" MEV attacks that makes the adversary able to steal a submitted solution. Fixing the contract requires to devise a non-trivial commit-reveal protocol (as done in Appendix B.5), through which we eventually achieve MEV-freedom.

Theorem 3 `Whitelist` *is* MEV*-free, for all* S *and* $\mathcal{X}$*.*

Proof. Let S be such that `Whitelist` contains $n : \mathtt{T}$ with $n > 0$. Let $\mathcal{X}$ be arbitrary, and let $\mathcal{B}$ be a cofinite set *not* including `A`. For all $\mathcal{A} \subseteq \mathcal{B}$ and for all redistributions $S \approx_{\$} S'$, we have that $\text{MEV}_{\mathcal{A}}(S', \mathcal{X}) = 0$, hence the max in (8) is zero. Therefore, $\text{MEV}(S, \mathcal{X}) = 0$: indeed, any reachable state is MEV-free in any mempool. Note that taking the min w.r.t. all cofinite $\mathcal{B}$ in Definition 9 is

instrumental to exclude from the potential adversaries those actors which are assigned a privileged role by the contract, as A in Whitelist. In this way, adversaries cannot exploit their identities to extract MEV.

Theorem 4 Blacklist *is* not MEV-*free, for all* $\mathcal{X}$ *and all* S *where both wallets and the contract contain at least* $1:\mathtt{T}$.

Proof. Let $\mathcal{B}$ be any cofinite set of actors, let $\mathcal{X}$ be an arbitrary mempool, and assume that Blacklist contains $n:\mathtt{T}$ with $n > 0$. Let $\mathcal{A} = \mathcal{B}$, and let $S \approx_\$ S'$ assign at least $1:\mathtt{T}$ to some $\mathtt{M} \neq \mathtt{A}$ in $\mathcal{A}$. This maximizes $\mathrm{MEV}_{\mathcal{A}}(S', \mathcal{X}) = n \cdot \$\mathbf{1}_{\mathtt{T}}$. Therefore, taking the min w.r.t. all cofinite sets of actors yields $\mathrm{MEV}(S, \mathcal{X}) = n \cdot \$\mathbf{1}_{\mathtt{T}}$, and so the contract is *not* MEV-free, as expected. Note that some redistributions would lead to zero MEV: e.g., this is the case when all tokens are assigned to the blacklisted A. To avoid this issue, the max in (8) allows to consider the MEV resulting from the most favourable redistribution for the adversary. Note that computing the min w.r.t. all *cofinite* $\mathcal{B}$ ensures that we can always assign tokens to non-blacklisted actors.

Theorem 5 Bank *is* MEV-*free, for all* S *and finite* $\mathcal{X}$.

Proof. Let $\mathcal{B}$ be cofinite and not including any actor in $\mathcal{X}$ or in the contract state (which can only mention finitely many actors). Observe that $\mathrm{MEV}_{\mathcal{A}}(S', \mathcal{X}) = 0$, for any $\mathcal{A} \subseteq \mathcal{B}$ and token redistribution $S \approx_\$ S'$. Indeed, there are only two ways for $\mathcal{A}$ to extract tokens from the contract: via a wdraw, or a xfer followed by a wdraw. Now, since the contract state does not mention actors in $\mathcal{A}$, wdraw transactions fired from $\mathcal{A}$ are invalid (unless they are preceded by a deposit, but this would lead to a non-positive gain). Since the mempool $\mathcal{X}$ does not mention actors in $\mathcal{B}$, xfer transactions to actors in $\mathcal{A} \subseteq \mathcal{B}$ are not forgeable by $\mathcal{A}$. Therefore, the max in (8) is zero, and so any contract state is MEV-free.

4 Related work

The first (partial) formal definition of MEV was given by Babel, Daian, Kelkar and Juels [9]. Transliterated into our notation, it is:

$$\mathrm{MEV}^{\mathrm{BDKJ}}_{\mathtt{A}}(S) = \max\{\gamma_{\mathtt{A}}(S, \mathcal{Y}) \mid \mathcal{Y} \in \mathrm{Blocks}(\mathtt{A}, S)\} \tag{10}$$

where $\mathrm{Blocks}(\mathtt{A}, S)$ represents the set of all valid blocks that A can construct in S, and A is allowed to own multiple wallets. A first key difference w.r.t. our work is that, while $\mathrm{Blocks}(\mathtt{A}, S)$ is not specified in [9], we provide an axiomatization of this set in Definition 4. Notably, our axiomatization allows us to prove key properties of MEV, as monotonicity (w.r.t. mempools and wallets) and actors/mempool finiteness. The dependence of $\mathrm{Blocks}(\mathtt{A}, S)$ on the mempool is left implicit in [9], while we make the mempool a parameter of MEV. This allows our attackers to craft transactions by combining their private knowledge with that of the mempool, as in the attack to the BadHTLC in Example 1. Instead, in [9]

the transactions in Blocks($\mathtt{A}, S$) are either generated by $\mathtt{A}$ using only her private knowledge, or taken from the (implicit) mempool.

Another key difference is that [9] does not provide a notion of *universal* MEV, i.e., the MEV that can be extracted by anyone, regardless of their identity and current token balance. Indeed, this is the kind of MEV which is most relevant in practice. The intuition of [9] is to compute $\mathrm{MEV}_{\mathtt{A}}^{\mathrm{BDKJ}}(S)$ w.r.t. an actor $\mathtt{A}$ who is "external" to the contract. However, the intuition is not supported by a formalization, which is not straightforward to achieve in general. Instead, our Definition 9 exactly characterises this universal MEV, making it identity-agnostic and token-agnostic.

Before ours, a version of universal MEV was proposed by Salles [37], and it has the following form:

$$\mathrm{MEV}^{\mathrm{Salles}}(S) = \min_{\mathtt{A} \in \mathbb{A}} \mathrm{MEV}_{\mathtt{A}}^{\mathrm{BDKJ}}(S) \tag{11}$$

By taking the minimum over all actors, (11) no longer depends on the identity of the attacker. As noted in [37], a drawback is that such definition classifies as MEV-free contracts that intuitively are not: e.g., the `Blacklist` contract would be considered MEV-free, since performing the attack requires upfront costs, that not all actors can afford. Note instead that Theorem 4 correctly classifies `Blacklist` as *not* MEV-free. A fix proposed in [37,31] is to parameterise MEV by a constant n, which restricts the set of attackers to those who own at least n tokens:

$$\mathrm{MEV}^{\mathrm{Salles}}(S, n) = \min \left\{ \mathrm{MEV}_{\mathtt{A}}^{\mathrm{BDKJ}}(S) \;\middle|\; \mathtt{A} \in \mathbb{A}, W(\mathtt{A}) \geq n \right\} \tag{12}$$

where $W(\mathtt{A})$ is the number of tokens in $\mathtt{A}$'s wallet. We note that also this fix has drawbacks. A first issue is that the set of actors who own at least n tokens in a state S is always *finite*. Hence, a contract that blacklists all these actors preventing them to withdraw tokens would be MEV-free according to $\mathrm{MEV}^{\mathrm{Salles}}(S, n)$. Instead, Definition 9 correctly classifies it as *not* MEV-free, since for all cofinite $\mathcal{B}$, the tokens can be *redistributed* to some $\mathcal{A} \subseteq \mathcal{B}$ who are not blacklisted, and can then extract MEV. Another issue of making the min in (12) range over all actors is the following. Consider a variant of `Blacklist` where calling `pay` requires zero tokens. Since the min in (12) must take also the hard-coded $\mathtt{A}$ into account, and $\mathtt{A}$'s MEV is zero (since she is blacklisted) then the min would be 0, and so (12) would incorrectly classify the contract as MEV-free. Instead, our redistribution game allows us to rule out such $\mathtt{A}$.

The notion of MEV in [15] is based on ours, but it uses an alternative approach to make MEV independent from the wealth of adversaries: rather than using a token redistribution, it takes the maximum MEV over all possible user wallets. Unlike ours, [15] does not provide a model of the adversarial knowledge.

Using redistributions like ours is also helpful to solve another issue of [37]: namely, blacklisting could also be based on the number of tokens held in wallets, e.g., preventing actors with more than 100 tokens from extracting MEV. In this case, $\mathrm{MEV}^{\mathrm{Salles}}(S, n)$ would be zero for all n, since the minimum would also take into account the blacklisted actors with zero MEV, while our notion correctly classifies the contract as *not* MEV-free.

As discussed in Section 3.3, our notion of universal MEV is game-theoretic. Another approach based on game theory — but with substantially different goals — is followed in [31]. This work models the priority gas auction arising from MEV extraction as a game, and studies the Nash equilibria ensuring that adversaries have the same MEV opportunities. Our goal instead is to formalize MEV so to analyse contracts w.r.t. MEV attacks.

5 Limitations and conclusions

While designing our MEV model, we strove to capture the most important aspects of MEV in common smart contracts. However, our model still has some limitations, which we discuss below.

Long-range attacks Our notion of MEV models the value extractable by adversaries in a single block: indeed, in Definition 5 we allow the adversary to perform a sequence $\mathcal{Y}$ of transactions (which models the block), but we neglect additional transactions happening after $\mathcal{Y}$. This does not consider long-range attacks spanning across multiple blocks. For instance, the adversary could perform some contract actions in one block that do not extract any MEV immediately, but affect the state of a time-based contract that will eventually give MEV. Precisely addressing long-range attacks would require some extensions to our theory. First, the notion of wealth should take into account token price fluctuations, which are irrelevant within a single block. Second, the knowledge κ should also depend on the blockchain state, e.g., to take revealed secrets into account. Third, MEV should depend on the strategies of the honest actors, i.e., on the transactions that they would send to the mempool in a given state. Actually, not considering these strategies, and just assuming that any actor is always willing to perform any contract action, would result in a gross over-approximation of MEV. An additional complication is that in certain contracts, like e.g., in gambling games, such strategies could be probabilistic. There, defining MEV in terms of the best possible future state for the adversary would again provide an over-approximation of MEV, since it would assume an unrealistically lucky adversary. For instance, consider a guessing game where an actor commits to a secret number, and then the adversary must guess its parity to win. Taking the *maximal* gain over all possible futures effectively provides the adversary with the knowledge of the secret.

Computational adversaries Our notion of adversarial knowledge in Definition 4 is *sharp*: any piece of data is either known to the adversary or completely inaccessible to them. This assumption is common in symbolic models of cryptographic protocols, but it does not always perfectly model the real-world adversaries. Indeed, an adversary could be able to obtain some data but only at the cost of a long computation. For instance, a contract could require the adversary to solve a moderately hard cryptographic puzzle to extract MEV. Modeling this kind of computational adversaries would require to refine the notions of adversarial knowledge and MEV to take costs into account.

Cost of MEV Our notion of universal MEV evaluates the MEV that can be extracted by an arbitrarily wealthy adversary. To this purpose, Definition 9 uses token redistributions, which allow the adversary to use *all* tokens in the state, even those belonging to honest actors. In this way, we are effectively assuming that the adversary always has, either in their wallet or by buying them from honest actors, all the tokens needed to carry the attack. An alternative definition, which does not require to grab the tokens of honest actors, would be to allow the adversary to mint the tokens needed in the attack, similarly to the definition of MEV^∞ in [15]. In scenarios where tokens are used as credentials to perform given actions, we could restrict token redistribution to avoid giving such special tokens to the adversary. We also remark that transaction fees do not contribute to MEV, similarly to [9,37]. Encompassing fees would allow to declassify as MEV attacks those where the fees needed to carry the attack exceed the adversarial gain. In practice, fees may be quite costly in private mempools like Flashbots [41]. Adversaries can extract MEV also by front-running a transaction in the mempool so to increase the amount of gas needed to validate it.

Good *vs.* bad MEV There is an open debate within the community about what exactly constitutes MEV, and how to separate "good MEV" from "bad MEV" [24,12,27,39,32,20]. In the absence of an agreement about these notions, our definition of "bad MEV" in Definition 7 formally captures some of the arguments used in this debate. In particular, we classify as "good" the MEV obtained through arbitrage in AMMs, since it does not exploit the mempool. Furthermore, we classify the liquidations on Lending Pools obtained without exploiting the mempool as "good" MEV. These classifications seem coherent with discussions in the community: MEV is good when it is an incentive for *any* user to perform actions (e.g., arbitrage, liquidations) that serve to the purpose of the protocol (e.g., aligning prices for AMMs, repaying loans for Lending Pools). Instead, we classify as "bad" the MEV resulting from sandwich attacks to AMMs [44] and from liquidations that back-run interest-accruing transactions in Lending Pools: this is correct in our view, because these attacks require the privileges of block proposers (by contrast, plain arbitrages and liquidations can be performed by any user with sufficient tokens).

We stress that not all the intuitively bad MEVs are classified as such by our Definition 7: this is the case, e.g., of the DAO attack [2], where MEV results from a bug in the contract implementation. On the other side, we are not aware of any real-world contracts that have "bad MEV" (according to our definition), but where the MEV is considered beneficial to the contract functionality.

Private order flows The "no shared secret" axiom in Definition 4 forbids actors to share private information (e.g., keys). It also rules out *private order flows* [25]: if the same order flow is sent to `A` and `B`, they might infer some common transactions that are not public knowledge. This simplifying assumption can be relaxed by making variables `A` represent secrets rather than actors, and modelling actors as the set of secrets $\mathcal{A}$ that they know.

Acknowledgments Work partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU, and by PRIN 2022 PNRR project DeLiCE (F53D23009130001).

References

1. MEV-explore: Flashbots transparency dashboard. explore.flashbots.net, accessed October 11th, 2024
2. Understanding the DAO attack (June 2016), http://www.coindesk.com/understanding-dao-hack-journalists/
3. Balancer whitepaper (2019), https://balancer.finance/whitepaper/
4. Aave website (2020), https://www.aave.com
5. Curve token pair implementation (2021), https://github.com/curvefi/curve-contract/blob/a1b5a797790d3f5ef12b0e358892a0ce47c12f85/contracts/pool-templates/base/SwapTemplateBase.vy
6. Uniswap token pair implementation (2021), https://github.com/Uniswap/uniswap-v2-core/blob/4dd59067c76dea4a0e8e4bfdda41877a6b16dedc/contracts/UniswapV2Pair.sol
7. Angeris, G., Chitra, T.: Improved price oracles: Constant Function Market Makers. In: ACM Conference on Advances in Financial Technologies (AFT). pp. 80–91. ACM (2020). https://doi.org/10.1145/3419614.3423251, https://arxiv.org/abs/2003.10001
8. Angeris, G., Kao, H.T., Chiang, R., Noyes, C., Chitra, T.: An analysis of Uniswap markets. Cryptoeconomic Systems **1**(1) (2021). https://doi.org/10.21428/58320208.c9738e64
9. Babel, K., Daian, P., Kelkar, M., Juels, A.: Clockwork finance: Automated analysis of economic security in smart contracts. In: IEEE Symposium on Security and Privacy. pp. 622–639. IEEE Computer Society (2023). https://doi.org/10.1109/SP46215.2023.00036
10. Babel, K., Javaheripi, M., Ji, Y., Kelkar, M., Koushanfar, F., Juels, A.: Lanturn: Measuring economic security of smart contracts through adaptive learning. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 1212–1226. ACM (2023). https://doi.org/10.1145/3576915.3623204
11. Babel, K., Jean-Louis, N., Ji, Y., Misra, U., Kelkar, M., Mudiyanselage, K.Y., Miller, A., Juels, A.: PROF: protected order flow in a profit-seeking world. CoRR **abs/2408.02303** (2024). https://doi.org/10.48550/ARXIV.2408.02303
12. Barczentewicz, M.: MEV on Ethereum: A policy analysis (2023). https://doi.org/http://dx.doi.org/10.2139/ssrn.4332703
13. Bartoletti, M., Chiang, J.H., Lluch-Lafuente, A.: Maximizing extractable value from Automated Market Makers. In: Financial Cryptography. LNCS, vol. 13411, pp. 3–19. Springer (2022). https://doi.org/10.1007/978-3-031-18283-9_1
14. Bartoletti, M., Chiang, J.H., Lluch-Lafuente, A.: A theory of Automated Market Makers in DeFi. Logical Methods in Computer Science **18**(4) (2022). https://doi.org/10.46298/lmcs-18(4:12)2022
15. Bartoletti, M., Marchesin, R., Zunino, R.: DeFi composability as MEV non-interference. In: Financial Cryptography. LNCS, vol. 14744. Springer (2024)
16. Baum, C., yu Chiang, J.H., David, B., Frederiksen, T.K., Gentile, L.: SoK: Mitigation of front-running in decentralized finance. Cryptology ePrint Archive, Report 2021/1628 (2021), https://ia.cr/2021/1628

17. Baum, C., David, B., Frederiksen, T.K.: P2DEX: privacy-preserving decentralized cryptocurrency exchange. In: Applied Cryptography and Network Security (ACNS). LNCS, vol. 12726, pp. 163–194. Springer (2021). https://doi.org/10.1007/978-3-030-78372-3_7
18. Breidenbach, L., Daian, P., Tramèr, F., Juels, A.: Enter the Hydra: Towards principled bug bounties and exploit-resistant smart contracts. In: USENIX Security Symposium. pp. 1335–1352. USENIX Association (2019)
19. Canidio, A., Danos, V.: Commitment against front-running attacks. Manag. Sci. **70**(7), 4429–4440 (2024). https://doi.org/10.1287/MNSC.2023.01239
20. Chiplunkar, A., Gosselin, S.: A new game in town (2023), https://frontier.tech/a-new-game-in-town
21. Ciampi, M., Ishaq, M., Magdon-Ismail, M., Ostrovsky, R., Zikas, V.: Fairmm: A fast and frontrunning-resistant crypto market-maker. In: Cyber Security, Cryptology, and Machine Learning (CSCML). LNCS, vol. 13301, pp. 428–446. Springer (2022). https://doi.org/10.1007/978-3-031-07689-3_31
22. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: IEEE Symp. on Security and Privacy. pp. 910–927. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00040
23. Eskandari, S., Moosavi, S., Clark, J.: SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In: Financial Cryptography. pp. 170–189. Springer (2020). https://doi.org/10.1007/978-3-030-43725-1_13
24. Flashbots: Develop an MEV taxonomy (2021), https://github.com/flashbots/mev-research/issues/24
25. Gupta, T., Pai, M.M., Resnick, M.: The centralizing effects of private order flow on proposer-builder separation. In: Advances in Financial Technologies (AFT). LIPIcs, vol. 282, pp. 20:1–20:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.AFT.2023.20
26. Heimbach, L., Wattenhofer, R.: Sok: Preventing transaction reordering manipulations in decentralized finance. In: Advances in Financial Technologies (2022)
27. Ji, Y., Grimmelmann, J.: Regulatory implications of MEV mitigations. In: Financial Cryptography Workshops. LNCS, vol. 14746, pp. 335–363. Springer (2024). https://doi.org/10.1007/978-3-031-69231-4_21
28. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for Byzantine consensus. In: Advances in Cryptology (CRYPTO). LNCS, vol. 12172, pp. 451–480. Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_16
29. Kulkarni, K., Diamandis, T., Chitra, T.: Towards a theory of Maximal Extractable Value I: constant function market makers. CoRR **abs/2207.11835** (2022). https://doi.org/10.48550/arXiv.2207.11835
30. Li, Z., Pournaras, E.: Sok: Consensus for fair message ordering. CoRR **abs/2411.09981** (2024). https://doi.org/10.48550/ARXIV.2411.09981
31. Mazorra, B., Reynolds, M., Daza, V.: Price of MEV: towards a game theoretical approach to MEV. In: ACM CCS Workshop on Decentralized Finance and Security. pp. 15–22. ACM (2022). https://doi.org/10.1145/3560832.3563433
32. Monoceros Venture: The MEV book: A comprehensive guide to Maximal Extractable Value (2024), https://www.monoceros.com/insights/maximal-extractable-value-book
33. Öz, B., Sui, D., Thiery, T., Matthes, F.: Who wins Ethereum block building auctions and why? In: Advances in Financial Technologies (AFT). LIPIcs, vol. 316, pp. 22:1–22:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/LIPICS.AFT.2024.22

34. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? In: IEEE Symp. on Security and Privacy. pp. 198–214. IEEE (2022). https://doi.org/10.1109/SP46214.2022.9833734
35. Qin, K., Zhou, L., Livshits, B., Gervais, A.: Attacking the DeFi ecosystem with Flash Loans for fun and profit. In: Financial Cryptography. LNCS, vol. 12674, pp. 3–32. Springer (2021). https://doi.org/10.1007/978-3-662-64322-8_1
36. Raikwar, M., Polyanskii, N., Müller, S.: Fairness notions in DAG-based DLTs. In: Blockchain Research & Applications for Innovative Networks and Services (BRAINS). pp. 1–8. IEEE (2023). https://doi.org/10.1109/BRAINS59668.2023.10316937
37. Salles, A.: On the formalization of MEV (2021), https://writings.flashbots.net/research/formalization-mev
38. Torres, C.F., Camino, R., State, R.: Frontrunner Jones and the Raiders of the Dark Forest: An empirical study of frontrunning on the Ethereum blockchain. In: USENIX Security Symposium. pp. 1343–1359 (2021)
39. Torres, C.F., Mamuti, A., Weintraub, B., Nita-Rotaru, C., Shinde, S.: Rolling in the shadows: Analyzing the extraction of MEV across layer-2 rollups. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 2591–2605. ACM (2024). https://doi.org/10.1145/3658644.3690259
40. Wahrstätter, A., Zhou, L., Qin, K., Svetinovic, D., Gervais, A.: Time to bribe: Measuring block construction market. CoRR **abs/2305.16468** (2023). https://doi.org/10.48550/ARXIV.2305.16468
41. Weintraub, B., Torres, C.F., Nita-Rotaru, C., State, R.: A Flash(Bot) in the pan: Measuring maximal extractable value in private pools. In: ACM Internet Measurement Conference. p. 458–471. ACM (2022). https://doi.org/10.1145/3517745.3561448
42. Werner, S., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., Knottenbelt, W.J.: SoK: Decentralized Finance (DeFi). In: ACM Conference on Advances in Financial Technologies, (AFT). pp. 30–46. ACM (2022). https://doi.org/10.1145/3558535.3559780
43. Zhou, L., Qin, K., Cully, A., Livshits, B., Gervais, A.: On the just-in-time discovery of profit-generating transactions in DeFi protocols. In: IEEE Symp. on Security and Privacy. pp. 919–936. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00113
44. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-Frequency Trading on Decentralized On-Chain Exchanges. In: IEEE Symp. on Security and Privacy. pp. 428–445. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00027

A A concrete contract language

We instantiate our abstract model with a simple contract language, dubbed TxScript. The language is heavily inspired by Solidity — and indeed we could have as well used Solidity itself to sketch our examples. Still, we opt for introducing a new language for coherence with the spirit of this paper: to formally prove properties of contracts, e.g., the presence or absence of MEV, we need a language with a formal semantics. To ease formal definitions and reasoning, our TxScript aims at minimality, so we drop all the features of Solidity that are inessential to the understanding of MEV. Despite this simplification, TxScript is still expressive enough to express real-world use cases like those found in DeFi.

A TxScript contract is a finite set of procedures of the form:

$$\texttt{f}(\overrightarrow{par_g})\{s\}$$

where `f` is the procedure name, $\overrightarrow{par_g}$ is the sequence of formal parameters, and s is the procedure body (see Figure 4). We assume that all the procedures in a contract have distinct names. Statements and expressions extend those of a loop-free imperative language with a few domain-specific constructs:

- `balance(T)` is the balance of tokens of type `T` deposited in the contract;
- `transfer(A,`n`:T)` transfers n units of `T` from the contract to `A`;
- `require` e rolls-back the transaction if condition e is false.

Transactions have the form $\texttt{f}(\overrightarrow{txarg})$, where the sequence of *actual* parameters may include, besides constants, the term `A pays` n`:T`, representing a transfer of n units of token `T` from `A` to the contract upon the procedure call.[6] Note that `A pays` n`:T` involves a signature of `A` on the transaction that authorize the transfer of tokens from her wallet to the contract. When we are only interested in the signature, and not in the transfer of tokens, we just write `A sig`, interpreting it as syntactic sugar for `A pays` 0`:T`. Transactions nonces $@n$ are used to prevent double-spending attacks: all transactions in the blockchain must have distinct nonces. We omit transaction nonces in examples.

To improve readability, when we want to fix some parameters in a procedure, instead of writing:

```
commit(a pays x:t,b,c) {
    require a==A && x==1 && t==BTC && b==B; ...
}
```

we just hard-code constants in the formal parameters, e.g.:

```
    commit(A pays 1:BTC,B,c) { ... }
```

[6] This mechanism generalises the one provided by Ethereum to transfer tokens upon contract calls. In Ethereum, a contract call involves a single transfer of *ether* from the caller to the contract. In TxScript, instead, a single transaction can involve multiple transfers of tokens (of any type) from the actors who authorise the transaction.

c ::=	`contract` C{ $\overrightarrow{p}$ }	**Contract**
p ::=	f($\overrightarrow{parg}$){s}	**Procedure**
parg ::=		**Argument**
	x	variable
|	a `pays` x:t	token input
|	a `sig`	signature
s ::=		**Statement**
	`skip`	skip
	`require` e	require condition
|	$x = e$	assignment
|	$x[e_1] = e_2$	map update
|	`transfer`(e_1, e_2:e_3)	token output
|	$s_1; s_2$	sequence
|	`if` e `then` s_1 `else` s_2	conditional
e ::=		**Expression**
	`null`	undefined
|	n | `A` | `T`	constants
|	x	variables
|	$e_1[e_2]$	map lookup
|	$e_1 \circ e_2$	operation
|	`balance`(e)	number of tokens of type e
|	`H`($e_1, \ldots, e_n$)	collision-resistant hash
X ::=	f($\overrightarrow{txarg}$)@n	**Transaction**
txarg ::=		**Transaction argument**
	n | `A` | `T`	constants
|	`A` `pays` n:`T`	token input
|	`A` `sig`	signature

Fig. 4. Syntax of contracts and transactions.

Example 2. Recall the HTLC from Figure 6, and let:

$$\mathtt{A}[1:\mathtt{T}] \mid \mathtt{B}[0:\mathtt{T}] \mid \mathtt{HTLC}[\sigma_0]$$

be an initial state where A owns $1:\mathtt{T}$, B owns nothing, and HTLC is in the initial state σ_0 where the balance is empty and all the variables are set to their default values (as in Solidity). Assume that $\mathcal{A}$ chooses a secret s, and computes its hash $H(s) = h$. Upon firing $\mathtt{commit}(\mathtt{A}\ \mathtt{pays}\ 1:\mathtt{T}, \mathtt{B}, h)$ the state takes a transition to:

$$\mathtt{A}[0:\mathtt{T}] \mid \mathtt{B}[0:\mathtt{T}] \mid \mathtt{HTLC}[\sigma_0\{^{\mathtt{A}}/_{\mathtt{xa}}, ^{\mathtt{B}}/_{\mathtt{xb}}, ^{h}/_{\mathtt{yc}}\}; 1:\mathtt{T}]$$

Now, upon firing $\mathtt{reveal}(rev(r_{\mathtt{A}}, b))$, the state evolves to:

$$\mathtt{A}[1:\mathtt{T}] \mid \mathtt{B}[0:\mathtt{T}] \mid \mathtt{HTLC}[\sigma_0\{^{\mathtt{A}}/_{\mathtt{xa}}, ^{\mathtt{B}}/_{\mathtt{xb}}, ^{h}/_{\mathtt{yc}}\}; 0:\mathtt{T}]$$

In this state, the committed secret has been revealed, and the committer A has redeemed her deposit. ◇

Semantics We assume a set **Var** of variables (ranged over by $x, y, \ldots$), partitioned in two subsets: **PVar** for the variables used in procedure arguments, and **SVar** for the state variables, which are further partitioned into base variables and map variables. We use t for variable token types, and a for variable actors. We assume that the variables used in the LHS of any assignment and in map updates are in **SVar**.

Contract states are pairs of the form $(\sigma, \mathcal{X})$. The component σ is a total map in $\mathbf{SVar} \cup \mathbb{T} \to \mathbf{Val}$, where **Val** is the universe of values, comprising base values **BVal**, and total maps from base values to values. Base values are natural numbers ($\mathbb{N}$), actors ($\mathbb{A}$), tokens ($\mathbb{T}$), and the singleton $\{null\}$. We embed booleans into $\mathbb{N}$ as usual.

The component $\mathcal{X}$ of the contract state records the set of all transactions executed so far, and it is used to prevent the double-spending of transactions in the mempool.[7] A contract state $(\sigma, \mathcal{X})$ is *initial* when $\sigma(\mathtt{T}) = 0$ for all $\mathtt{T} \in \mathbb{T}$, $\sigma(x) = null$ for all base variables, $\sigma(x) = \lambda b.\, null$ for all map variables, and $\mathcal{X} = \emptyset$.

We will often omit transaction nonces: if the same procedure with the same parameters is executed two or more times, we implicitly assume that all its transaction nonces are distinct.

The semantics of contracts is a labelled transition relation between blockchain states, with signature as in Definition 1. The transition relation $\mapsto$ is specified by the following rule, which updates the blockchain state when a valid transaction

[7] Blockchain platforms use similar mechanisms to avoid replay attacks which double spend a transaction. For instance, Algorand marks a transaction as invalid if it belongs to the set of transactions fired in the last 1000 rounds. In Ethereum, each transaction must be signed by its sender: the signature also includes a nonce, which is increased each time the sender broadcasts a transaction. In the blockchain, the nonces of the transactions from the same sender must be ordered, without skipping.

is fired:

$$\mathsf{X} = \mathtt{f}(\overrightarrow{txarg})@n \qquad \mathtt{f}(\overrightarrow{parg})\{s\} \in \mathtt{C} \qquad \mathsf{X} \notin \mathcal{X}$$

$$\frac{(\overrightarrow{parg})\rho = \overrightarrow{txarg} \qquad \langle \overrightarrow{txarg}, (W, \sigma)\rangle \Rightarrow_{arg} (W'', \sigma'') \qquad \langle s, (W'', \sigma'')\rangle_\rho \Rightarrow (W', \sigma')}{(W, (\sigma, \mathcal{X})) \overset{\mathsf{X}}{\longmapsto} (W', (\sigma', \mathcal{X} \cup \{\mathsf{X}\}))}$$

The rule defines a single state transition triggered by a (valid) transaction X. The condition $\mathsf{X} \notin \mathcal{X}$ in the first line of the rule premises ensures that the same transaction cannot be executed twice. The second line of the rule premises infers a substitution ρ to match the formal and the actual parameters of the called procedure. The condition $\langle \overrightarrow{txarg}, (W, \sigma)\rangle \Rightarrow_{arg} (W'', \sigma'')$ evaluates the transaction arguments (see below). Finally, the premise $\langle s, (W'', \sigma'')\rangle_\rho \Rightarrow (W', \sigma')$ evaluates the procedure statement s, producing a new blockchain state. Note that if some `require` commands in the statement s fail then $\langle s, (W'', \sigma'')\rangle_\rho \Rightarrow \bot$, hence the premise is false, and the rule does not apply.

The semantics of expressions in a state σ is standard, except for the wallet lookup, the semantics of which is defined as follows:

$$\frac{[\![e]\!]_{\rho,\sigma} = \mathtt{T}}{[\![\mathtt{balance}(e)]\!]_{\rho,\sigma} = \sigma(\mathtt{T})}$$

We assume a basic type system on expressions, which rules out operations between non-compatible types, like e.g., $\mathtt{A} + 1$, $\mathtt{T} + 1$, $\mathtt{balance}(\mathtt{A})$, $\mathtt{sec}(e)$ where e is anything but a reveal $rev(r, n)$.

The rule for transferring tokens from the contract to an actor is the following, where we use the standard notation $\sigma\{^{v}/_{x}\}$ to update a partial map σ at point x: namely, $\sigma\{^{v}/_{x}\}(x) = v$, while $\sigma\{^{v}/_{x}\}(y) = \sigma(y)$ for $y \neq x$.

$$\frac{[\![e_1]\!]_{\rho,\sigma} = \mathtt{A} \quad [\![e_2]\!]_{\rho,\sigma} = n \quad [\![e_3]\!]_{\rho,\sigma} = \mathtt{T} \quad \sigma(\mathtt{T}) \geq n}{\langle \mathtt{transfer}(e_1, e_2 : e_3), (W, \sigma)\rangle_\rho \Rightarrow (W\{^{W(\mathtt{A})+[n:\mathtt{T}]}/_{\mathtt{A}}\}, \sigma - [n:\mathtt{T}])}$$

The rules for evaluating `require` commands are the following:

$$\frac{[\![e]\!]_{\rho,\sigma} = \mathit{true}}{\langle \mathtt{require}\ e, (W, \sigma)\rangle_\rho \Rightarrow (W, \sigma)} \qquad \frac{[\![e]\!]_{\rho,\sigma} = \mathit{false}}{\langle \mathtt{require}\ e, (W, \sigma)\rangle_\rho \Rightarrow \bot}$$

The second rule deal with the case when the condition under the `require` is violated: in this case, the evaluation of the command yields the special value $\bot$, which represents an execution error.

The rule for evaluating actual parameters is the following:

$$\frac{W(\mathtt{A})(\mathtt{T}) \geq n \qquad \langle \overrightarrow{txarg}, (W\{^{W(\mathtt{A})-[n:\mathtt{T}]}/_{\mathtt{A}}\}, \sigma + [n:\mathtt{T}])\rangle \Rightarrow_{arg} (W', \sigma')}{\langle \mathtt{A}\ \mathtt{pays}\ n:\mathtt{T}; \overrightarrow{txarg}, (W, \sigma)\rangle \Rightarrow_{arg} (W', \sigma')}$$

The other rules are standard. The full set of rules is in Figure 5.

$$\frac{}{[\![\texttt{null}]\!]_{\rho,\sigma} = null} \qquad \frac{}{[\![v]\!]_{\rho,\sigma} = v} \qquad \frac{}{[\![x]\!]_{\rho,\sigma} = (\rho \cup \sigma)(x)} \qquad \frac{[\![e]\!]_{\rho,\sigma} = \mathtt{T}}{[\![\texttt{balance}(e)]\!]_{\rho,\sigma} = \sigma(\mathtt{T})}$$

$$\frac{[\![e_1]\!]_{\rho,\sigma} = f \in \mathbf{BVal} \to \mathbf{Val} \quad [\![e_2]\!]_{\rho,\sigma} = v \in \mathbf{BVal}}{[\![e_1[e_2]]\!]_{\rho,\sigma} = f(v)}$$

$$\frac{[\![e_1]\!]_{\rho,\sigma} = n_1 \quad [\![e_2]\!]_{\rho,\sigma} = n_2 \quad n_1 \circ n_2 = n \in \mathbb{N}}{[\![e_1 \circ e_2]\!]_{\rho,\sigma} = n}$$

$$\frac{[\![e_1]\!]_{\rho,\sigma} = v_1 \quad \cdots \quad [\![e_n]\!]_{\rho,\sigma} = v_n \quad H(v_1 \cdots v_n) = v}{[\![\mathtt{H}(e_1, \ldots, e_n)]\!]_{\rho,\sigma} = v}$$

$$\frac{}{\langle \texttt{skip}, (W, \sigma)\rangle_\rho \Rightarrow (W, \sigma)}$$

$$\frac{[\![e]\!]_{\rho,\sigma} = true}{\langle \texttt{require}\ e, (W, \sigma)\rangle_\rho \Rightarrow (W, \sigma)} \qquad \frac{[\![e]\!]_{\rho,\sigma} = false}{\langle \texttt{require}\ e, (W, \sigma)\rangle_\rho \Rightarrow \bot}$$

$$\frac{[\![e]\!]_{\rho,\sigma} = v}{\langle x = e, (W, \sigma)\rangle_\rho \Rightarrow (W, \sigma\{^v/_x\})}$$

$$\frac{[\![x]\!]_{\rho,\sigma} = f \quad [\![e_1]\!]_{\rho,\sigma} = k \quad [\![e_2]\!]_{\rho,\sigma} = v \quad f' = f\{^v/_k\}}{\langle x[e_1] = e_2, (W, \sigma)\rangle_\rho \Rightarrow (W, \sigma\{^{f'}/_x\})}$$

$$\frac{[\![e_1]\!]_{\rho,\sigma} = \mathtt{A} \quad [\![e_2]\!]_{\rho,\sigma} = n \quad [\![e_3]\!]_{\rho,\sigma} = \mathtt{T} \quad \sigma(\mathtt{T}) \geq n}{\langle \texttt{transfer}(e_1, e_2 : e_3), (W, \sigma)\rangle_\rho \Rightarrow (W\{^{W(\mathtt{A})+n:\mathtt{T}}/_{\mathtt{A}}\}, \sigma - n:\mathtt{T})}$$

$$\frac{\langle s_1, (W, \sigma)\rangle_\rho \Rightarrow (W'', \sigma'') \quad \langle s_2, (W'', \sigma'')\rangle_\rho \Rightarrow (W', \sigma')}{\langle s_1; s_2, (W, \sigma)\rangle_\rho \Rightarrow (W\{^{W(\mathtt{A})+n:\mathtt{T}}/_{\mathtt{A}}\}, \sigma - n:\mathtt{T})}$$

$$\frac{[\![e]\!]_{\rho,\sigma} = b \quad \langle s_b, (W, \sigma)\rangle_\rho \Rightarrow (W', \sigma')}{\langle \texttt{if}\ e\ \texttt{then}\ s_{true}\ \texttt{else}\ s_{false}, (W, \sigma)\rangle_\rho \Rightarrow (W', \sigma')}$$

$$\frac{}{\langle \varepsilon, (W, \sigma)\rangle \Rightarrow_{arg} (W, \sigma)} \qquad \frac{\langle \overrightarrow{txarg}, (W, \sigma)\rangle \Rightarrow_{arg} (W', \sigma')}{\langle v; \overrightarrow{txarg}, (W, \sigma)\rangle \Rightarrow_{arg} (W', \sigma')}$$

$$\frac{W(\mathtt{A})(\mathtt{T}) \geq n \quad \langle \overrightarrow{txarg}, (W\{^{W(\mathtt{A})-n:\mathtt{T}}/_{\mathtt{A}}\}, \sigma + n:\mathtt{T})\rangle \Rightarrow_{arg} (W', \sigma')}{\langle \mathtt{A}\ \texttt{pays}\ n:\mathtt{T}; \overrightarrow{txarg}, (W, \sigma)\rangle \Rightarrow_{arg} (W', \sigma')}$$

$$\mathtt{X} = \mathtt{f}(\overrightarrow{txarg})@n \qquad \mathtt{f}(\overrightarrow{parg})\{s\} \in \mathtt{C} \qquad \mathtt{X} \notin \mathcal{X}$$

$$\frac{(\overrightarrow{parg})\rho = \overrightarrow{txarg} \qquad \langle \overrightarrow{txarg}, (W, \sigma)\rangle \Rightarrow_{arg} (W'', \sigma'') \qquad \langle s, (W'', \sigma'')\rangle_\rho \Rightarrow (W', \sigma')}{(W, (\sigma, \mathcal{X})) \overset{\mathtt{X}}{\mapsto} (W', (\sigma', \mathcal{X} \cup \{\mathtt{X}\}))}$$

Fig. 5. Semantics of contracts.

```
contract HTLC {
  commit(a pays 1:T,b,c) { // a must send 1 token T to the contract
    require verifier==null; committer=a; verifier=b; commitment=c;
  }
  reveal(a sig,y) { // a must sign the transaction
    require a==committer && balance(T)>0 && H(y)==commitment;
    transfer(a,balance(T):T) // send the whole balance of tokens T to a
  }
  timeout(Oracle sig) { // Oracle must sign the transaction
    require balance(T)>0; // the contract must have some tokens T
    transfer(verifier,balance(T):T);
    verifier=null;
  }
}
```

Fig. 6. A Hash Time Locked Contract.

B Evaluation

We now assess the effectiveness of our MEV theory on a benchmark of real-world contracts.

B.1 HTLC

The `HTLC` contract in Figure 6 implements a fix to the `BadHTLC` version in Figure 1. The fix consists in constraining the reveal method to only transfer tokens to the committer. The fixed contract is MEV-free in any state and finite mempool.

B.2 Coin Pusher

The following example shows the case where extracting MEV requires to fire some transactions found in the mempool. The `CoinPusher` contract in Figure 7 transfers all its tokens to anyone who makes the balance exceed $100:\mathtt{T}$. Let S be a state where the contract has $0:\mathtt{T}$, `A` has $1:\mathtt{T}$, and `M` has $99:\mathtt{T}$. In the empty mempool, `M` has no MEV in S, since she has not enough balance to trigger the push of tokens from the contract. Instead, with a mempool $\mathcal{X} = \{\mathtt{play}(\mathtt{A}\ \mathtt{pays}\ 1:\mathtt{T})\}$, we have that `M` can fire the sequence $\mathtt{play}(\mathtt{A}\ \mathtt{pays}\ 1:\mathtt{T})\,\mathtt{play}(\mathtt{M}\ \mathtt{pays}\ 99:\mathtt{T}) \in \kappa_{\mathtt{M}}(\mathcal{X})^*$, obtaining $\mathrm{MEV}_{\{\mathtt{M}\}}(S,\mathcal{X}) = 1 \cdot \mathbf{1}_{\mathtt{T}}$.

```
contract CoinPusher {
  play(a pays x:T) {
    require x>0;
    if balance(T)>=100 then transfer(a,balance(T):T) else skip;
  }
}
```

Fig. 7. A `CoinPusher` contract.

```
contract Crowdfund {
  init(a,n) {
    require rcv==null;
    rcv=a; goal=n; isOpen=true;
  }
  donate(a pays x:T) {
    require isOpen && x>0;
    if amount[a]=null then amount[a]=x else amount[a]=amount[a]+x;
  }
  claim() {
    require balance(T)>=goal && isOpen;
    transfer(rcv,balance(T):T); rcv=null; isOpen=false;
  }
  timeout(Oracle sig) {
    isOpen=false;
  }
  refund(a sig) {
    require amount[a]>0 && !isOpen;
    transfer(a,amount[a]:T); amount[a]=0;
  }
}
```

Fig. 8. A crowdfunding contract.

B.3 Crowdfund

Consider the `Crowdfund` contract in Figure 8. Let S be a state where someone has donated 50 tokens (with a goal of 51):

$$\mathtt{B}[0{:}\mathtt{T}] \mid \mathtt{Crowdfund}[50{:}\mathtt{T}, \mathtt{rcv} = \mathtt{B}, \mathtt{goal} = 51, \mathtt{isOpen} = \mathit{true}] \mid \cdots$$

For `B` to have a positive MEV, the mempool must contain a transaction where some $\mathtt{A} \neq \mathtt{B}$ donates at least $1{:}\mathtt{T}$. For instance, if `A` has at least $10{:}\mathtt{T}$ in S, then `B` has MEV for $\mathcal{X} = \{\mathtt{donate}(\mathtt{A}\ \mathtt{pays}\ 10{:}\mathtt{T})\}$. MEV can be extracted by firing the sequence $\mathtt{donate}(\mathtt{A}\ \mathtt{pays}\ 10{:}\mathtt{T})\,\mathtt{claim}() \in \kappa_{\{\mathtt{B}\}}(\emptyset)$, resulting in $\mathrm{MEV}_{\{\mathtt{B}\}}(S, \mathcal{X}) = 60 \cdot \$\mathbf{1}_{\mathtt{T}} > 0$.

Note however that S is MEV-free for any mempool. Indeed, choosing a cofinite $\mathcal{B}$ *not* including `B` gives $\mathrm{MEV}_{\mathcal{A}}(S', \mathcal{X}) = 0$ for any $\mathcal{A} \subseteq \mathcal{B}$ and any $S \approx_{\$} S'$, and so the max in (8) is zero. This is coherent with the intuition: indeed, after the `init`, the identity of the actor who can extract tokens is singled out in the contract state, and it cannot be replaced by an arbitrary miner/validator. In general, `Crowdfund` is MEV-free after `init` in any finite mempool. The only case where the contract is not MEV-free is the (hardly realistic) one where `init` has not been performed yet, and the mempool contains a `donate`.

B.4 Automated Market Makers

We now formally prove the well-known fact that constant-product Automated Market Makers [7,8,14], a wide class of decentralized exchanges including mainstream platforms like Uniswap, Curve and Balancer [6,5,3], are *not* MEV-free.

Consider the `AMM` contract in Figure 9. Users can add reserves of $\mathtt{T}_0$ and $\mathtt{T}_1$ to the contract with `addliq` (preserving the reserves ratio), and exchange units of $\mathtt{T}_0$

```
contract AMM {
  addliq(a0 pays x0:T0,a1 pays x1:T1) {
    require balance(T0) * (balance(T1)-x1) == (balance(T0)-x0) * balance(T1)
  }
  swap0(a pays x:T0,ymin) {
    y = (x * balance(T1)) / balance(T0);
    require y>=ymin && y<balance(T1);
    transfer(a,y:T1);
  }
  swap1(a pays x:T1,ymin) {
    y = (x * balance(T0)) / balance(T1);
    require y>=ymin && y<balance(T0);
    transfer(a,y:T0);
  }
}
```

Fig. 9. A constant-product AMM contract.

with units of $\mathtt{T}_1$ with `swap0` and `swap1`. More specifically, $\mathtt{swap0}(\mathtt{A}\ \mathtt{pays}\ x{:}\mathtt{T}_0, y_{\min})$ allows A to send $x{:}\mathtt{T}_0$ to the contract, and receive at least $y_{\min}{:}\mathtt{T}_1$ in exchange. Symmetrically, $\mathtt{swap1}(\mathtt{A}\ \mathtt{pays}\ x{:}\mathtt{T}_1, y_{\min})$ allows A to exchange $x{:}\mathtt{T}_1$ for at least $y_{\min}{:}\mathtt{T}_0$.

We show how adversaries can extract MEV from the contract through the so-called *sandwich attack* [44]. Assume that the token prices are $\$\mathbf{1}_{\mathtt{T}_0} = 4$ $\$\mathbf{1}_{\mathtt{T}_1} = 9$. Let:

$$S = \mathtt{AMM}[6{:}\mathtt{T}_0, 6{:}\mathtt{T}_1] \mid \mathtt{A}[3{:}\mathtt{T}_0] \mid \cdots$$

Since the exchange rate given by the AMM ($1{:}\mathtt{T}_0$ for $1{:}\mathtt{T}_1$) is more convenient than the exchange rate given by the token prices ($9{:}\mathtt{T}_0$ for $4{:}\mathtt{T}_1$), A would have a positive gain by firing $\mathsf{X}_\mathtt{A} = \mathtt{swap1}(\mathtt{A}\ \mathtt{pays}\ 3{:}\mathtt{T}_1, 1)$ in the *current* state. Indeed, we would have:

$$S \xrightarrow{\mathsf{X}_\mathtt{A}} \mathtt{AMM}[9{:}\mathtt{T}_0, 4{:}\mathtt{T}_1] \mid \mathtt{A}[0{:}\mathtt{T}_0, 2{:}\mathtt{T}_1] \mid \cdots$$

and so A's gain would be $\gamma_\mathtt{A}(S, \mathsf{X}_\mathtt{A}) = 2 \cdot \$\mathbf{1}_{\mathtt{T}_1} - 3 \cdot \$\mathbf{1}_{\mathtt{T}_0} = 6 > 0$. In a sandwdich attack, the adversary has access to the mempool $\mathcal{X} = \{\mathsf{X}_\mathtt{A}\}$, and can have a positive gain to A's detriment. Let $\mathcal{B}$ be any cofinite adversary, and pick $\mathtt{M} \neq \mathtt{A}$ in $\mathcal{B}$. Assume a token redistribution which assigns $3{:}\mathtt{T}_0$ to M, and let $\mathcal{A} = \{\mathtt{M}\}$. We show that $\mathcal{A}$ has a positive MEV, and so $(S, \mathcal{X})$ is *not* MEV-free.

The idea of the sandwich attack is the following:

1. M front-runs $\mathsf{X}_\mathtt{A}$ to make the AMM reach an equilibrium, where the AMM exchange rate equals the exchange rate given by the external token prices. This is done through $\mathsf{X}_\mathtt{M} = \mathtt{swap0}(\mathtt{M}\ \mathtt{pays}\ 3{:}\mathtt{T}_0, 2)$;
2. then, M plays $\mathsf{X}_\mathtt{A}$. In this way, A will receive fewer units of $\mathtt{T}_1$ than she would have obtained in S. Indeed, since the AMM is in equilibrium after $\mathsf{X}_\mathtt{M}$, A will have a negative gain;
3. finally, M closes the sandwich with a transaction that makes the AMM reach again the equilibrium state. This is done through $\mathsf{X}'_\mathtt{M} = \mathtt{swap1}(\mathtt{M}\ \mathtt{pays}\ 1{:}\mathtt{T}_1, 3)$.

More precisely, we have the following computation:

$$
\begin{aligned}
S &\xrightarrow{\mathsf{X}_\mathtt{M}} \mathtt{AMM}[9{:}\mathtt{T}_0, 4{:}\mathtt{T}_1] \mid \mathtt{M}[0{:}\mathtt{T}_0, 2{:}\mathtt{T}_1] \mid \mathtt{A}[3{:}\mathtt{T}_0] \mid \cdots \\
&\xrightarrow{\mathsf{X}_\mathtt{A}} \mathtt{AMM}[12{:}\mathtt{T}_0, 3{:}\mathtt{T}_1] \mid \mathtt{M}[0{:}\mathtt{T}_0, 2{:}\mathtt{T}_1] \mid \mathtt{A}[0{:}\mathtt{T}_0, 1{:}\mathtt{T}_1] \mid \cdots \\
&\xrightarrow{\mathsf{X}'_\mathtt{M}} \mathtt{AMM}[9{:}\mathtt{T}_0, 4{:}\mathtt{T}_1] \mid \mathtt{M}[3{:}\mathtt{T}_0, 1{:}\mathtt{T}_1] \mid \mathtt{A}[0{:}\mathtt{T}_0, 1{:}\mathtt{T}_1] \mid \cdots
\end{aligned}
$$

The resulting gains for M and A are:

$$
\begin{aligned}
\gamma_\mathtt{A}(S, \mathsf{X}_\mathtt{M}\mathsf{X}_\mathtt{A}\mathsf{X}'_\mathtt{M}) &= 1 \cdot \$\mathbf{1}_{\mathtt{T}_1} - 3 \cdot \$\mathbf{1}_{\mathtt{T}_0} = -3 < 0 \\
\gamma_\mathtt{M}(S, \mathsf{X}_\mathtt{M}\mathsf{X}_\mathtt{A}\mathsf{X}'_\mathtt{M}) &= 1 \cdot \$\mathbf{1}_{\mathtt{T}_1} = 9 > 0
\end{aligned}
$$

Since this is the optimal strategy for M (as shown e.g., in [13]), we conclude that $\mathrm{MEV}(S, \mathcal{X}) = 9$.

B.5 Bounty contract

```
contract BadBounty {
  init(b pays n:T) { }
  claim(a,sent_sol) {
    require balance(T)>0;
    require <sent_sol is really the solution>;
    transfer(a,balance(T):T);
  }
}
```

Fig. 10. A MEV-leaking bounty contract.

Consider a bounty contract which rewards the first user who submits the solution to a puzzle. For simplicity we assume that the solution is unique, and that only a certain actor A can find the solution. We start with a naïve contract BadBounty in Figure 10. Here, A submits the solution by sending $\mathsf{X}_\mathtt{A} = \mathtt{claim}(\mathtt{A}, s)$, with her name and the solution s as parameters. In this state, where $\mathsf{X}_\mathtt{A}$ is in the mempool, the contract is *not* MEV-free. Indeed, an adversary M can front-run $\mathsf{X}_\mathtt{A}$ with $\mathtt{claim}(\mathtt{M}, s)$, which will release the bounty.

Fixing the contract to make it MEV-free requires some ingenuity. In the Bounty contract in Figure 11, users follow a commit-reveal protocol. First, they commit their name A together with the hash of the pair $(\mathtt{A}, s)$, where s is the bounty solution. Note that an adversary M can front-run the commit replaying the sniffed hashes together with her name M, but as we will see, this is not enough to extract MEV. All the commits are recorded in two maps usr and cmt: their ordering is controlled by the adversary. After A's commit is finalised on the blockchain, A calls ver to submit the actual solution s, making it public. When M sees this transaction in the mempool, she can front-run it with the commit of the pair $(\mathtt{M}, s)$, which is however recorded *after* A's commit in the maps. At

```
contract Bounty {
  init(b pays n:T) { }
  commit(a,sol_cmt) {
    require !found && balance(T)>0;
    usr[next] = a;
    cmt[next] = sol_cmt;
    next = next+1;
  }
  ver(sent_sol) {
    require !found && balance(T)>0;
    require <sent_sol is really the solution>;
    found = true;
    sol = sent_sol;
  }
  claim() {
    require found && i_claimed<next;
    if (cmt[i_claimed] == H(usr[i_claimed],sol))
    then transfer(usr[i_claimed],balance(T):T)
    else i_claimed = i_claimed+1;
  }
}
```

Fig. 11. A MEV-free bounty contract.

this point A repeatedly sends claim transactions until receiving the bounty. The claim procedure scans the maps from the first commit onwards, releasing the tokens to the *first* user B who has submitted the hash of pair $(\mathtt{B}, s)$. So, even though M has front-run A's commit with her own, she will not receive the bounty, since the hashed name does not correspond to the name in the usr map.

In any state S reached when A follows this protocol, and any mempool $\mathcal{X}$ which contains the next move of A, the contract is MEV-free. Indeed, $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) = 0$ whenever $\mathtt{A} \notin \mathcal{A}$, and so the max in (8) is zero.

B.6 Lending Pools

We analyse MEV in a Lending Pool contract inspired by Aave [4] (see Figure 12). In general, Lending Pools implement loan markets and feature complex economic mechanisms to incentivize users to deposit tokens and repay loans. To keep our presentation self-contained, our LP contract makes several simplifications w.r.t. mainstream implementations; these simplifications, however, are irrelevant to the analysis of MEV.

Users can deposit tokens in the LP, obtaining in return virtual tokens minted by the contract. More precisely, upon a deposit of $x:\mathtt{T}$ in a pool with reserves of $n:\mathtt{T}$, the user will receive $x \cdot \mathtt{X}(n)$ minted tokens, where the exchange rate $\mathtt{X}(n)$ is given by:

$$\mathtt{X}(n) = \begin{cases} 1 & \text{if } \mathtt{totM} = 0 \\ \frac{n + \mathtt{totD} \cdot \mathtt{ir}}{\mathtt{totM}} & \text{otherwise} \end{cases}$$

where totM is the total number of minted tokens, and $\mathtt{totD} \cdot \mathtt{ir}$ is the total amount of debt (more on this below). A first simplification here is that our LP manages a *single* token type T, while actual implementations allow e.g., a user

to lend tokens of a certain type and borrow tokens of another type. Although this simplification makes our LP not interesting for practical purposes, as said before it is immaterial for MEV. The use of minted tokens is twofold. On the one hand, they are an incentive to lend: users deposit speculating that the minted tokens will be redeemable for a value greater than the original deposit. On the other hand, they are used as a collateral when borrowing tokens: namely, users can obtain a loan only if they have enough *collateralization*, that is given by:

$$\mathtt{C}(\mathtt{A}, n) = \frac{\mathtt{minted}[\mathtt{A}] \cdot \mathtt{X}(n)}{\mathtt{debt}[\mathtt{A}] \cdot \mathtt{ir}}$$

where n is the reserve of T in the pool, $\mathtt{minted}[\mathtt{A}]$ is the amount of minted tokens owned by the user, and $\mathtt{debt}[\mathtt{A}]$ is the amount of A's debt. The borrow action requires that the user collateralization after the action is above a given threshold.

Users can redeem their minted tokens for units of T, where the actual amount is obtained by applying the exchange rate. Also the redeem requires that the collateralization is above the threshold.

An oracle can accrue interests on loans from time to time. This is done by multiplying the interest rate ir by the factor $\mathtt{Imul} > 1$. Note that interest accrual may make some borrowers undercollateralized, exposing them to liquidations. Namely, a liquidate action allows *anyone* to repay part of the debt of an undercollateralized user, and obtain as a reward part of their minted tokens. The multiplication factor $\mathtt{Rliq} > 1$ incentivizes liquidations.

We now show that our formalization can precisely characterise a typical MEV attack on Lending Pools [34]. For simplicity, in our example we use fractional values, rather than integers, and we assume that $\$\mathbf{1}_{\mathtt{T}} = 1$. Consider the state S reached after the sequence of transactions:

- $\mathtt{X}_0 = \mathtt{init}(\mathtt{O}\ \mathtt{pays}\ 0\!:\!\mathtt{T}, 1.5, 1.3, 1.5)$, that initializes the contract;
- $\mathtt{X}_1 = \mathtt{deposit}(\mathtt{M}\ \mathtt{pays}\ 100\!:\!\mathtt{T})$, where M deposits $100\!:\!\mathtt{T}$;
- $\mathtt{X}_2 = \mathtt{deposit}(\mathtt{B}\ \mathtt{pays}\ 50\!:\!\mathtt{T})$, where B deposits $50\!:\!\mathtt{T}$;
- $\mathtt{X}_3 = \mathtt{borrow}(\mathtt{B}\ \mathtt{pays}\ 0\!:\!\mathtt{T}, 30)$ where B borrows $30\!:\!\mathtt{T}$.

The state S is determined as follows. From the initial state:

$$S_0 \;=\; \mathtt{LP}[0\!:\!\mathtt{T}] \mid \mathtt{M}[200\!:\!\mathtt{T}] \mid \mathtt{B}[100\!:\!\mathtt{T}] \mid \cdots$$

firing $\mathtt{X}_0$ leads to the state:

$$S_1 = \mathtt{LP}[0\!:\!\mathtt{T}, {}^{1.5}/_{\mathtt{Cmin}}, {}^{1.3}/_{\mathtt{Rliq}}, {}^{1}/_{\mathtt{ir}}, {}^{1.5}/_{\mathtt{Imul}}, \cdots] \mid \cdots$$

where we summarize with $\cdots$ the parts of the state unaffected by the transition. Then, firing $\mathtt{X}_1$ increases by $100\!:\!\mathtt{T}$ the LP reserves, and decreases by 100 the units of T in M's wallet. The new state is:

$$S_2 = \mathtt{LP}[100\!:\!\mathtt{T}, {}^{100}/_{\mathtt{totM}}, {}^{\mathtt{M}\to 100}/_{\mathtt{minted}}, \cdots] \mid \mathtt{M}[100\!:\!\mathtt{T}] \mid \cdots$$

The map minted records that M now has 100 units of the minted token. Then, firing $\mathtt{X}_2$ performs B's deposit, leading to the state:

$$S_3 = \mathtt{LP}[150\!:\!\mathtt{T}, {}^{150}/_{\mathtt{totM}}, {}^{\mathtt{M}\to 100, \mathtt{B}\to 50}/_{\mathtt{minted}}, \cdots] \mid \mathtt{B}[50\!:\!\mathtt{T}] \mid \cdots$$

```
contract LP {
  fun X(n) { // exchange rate 1:T = X() minted T
    if totM==0 then return 1
    else return (n + totD*ir)/totM
  }
  fun C(a,n) { // collateralization of a
    return (minted[a]*X(n))/(debt[a]*ir)
  }
  init(a sig,c,r,m) {
    require balance(T)==0 && r>1 && m>1;
    Cmin = c;   // minimum collateralization
    Rliq = r;   // liquidation bonus
    ir   = 1;   // interest rate
    Imul = m;   // interest rate multiplier
    totD = 0;   // total debt
    totM = 0;   // total minted tokens
  }
  deposit(a pays x:T) {
    y = x/X(balance(T)-x); // received minted tokens
    minted[a] += y;
    totM += y;
  }
  borrow(a sig,x) {
    require balance(T)>x;
    transfer(a,x:T);
    debt[a] += x / ir;
    totD += x / ir;
    require C(a,balance(T))>=Cmin;
  }
  accrue(Oracle sig) {
    ir = ir * Imul;
  }
  repay(a pays x:T) {
    require debt[a]*ir>=x;
    debt[a] -= x / ir;
  }
  redeem(a sig,x) {
    y = x * X(balance(T));
    require minted[a]>=x && balance(T)>=y;
    transfer(a,y:T);
    minted[a] -= x;
    totM -= x;
    require C(a,balance(T))>=Cmin
  }
  liquidate(a pays x:T,b) {
    y = (x / X(balance(T)-x)) * Rliq;
    require debt[b]*ir>x;
    require C(b,balance(T)-x)<Cmin;
    require minted[b] >= y;
    minted[a] += y;
    minted[b] -= y;
    debt[b] -= x/ir;
    totD -= x/ir;
    require C(b,balance(T))<=Cmin;
  }
}
```

Fig. 12. A Lending Pool contract.

At this point, B fires X_3, borrowing $30:T$ from the LP. In the reached state S, these units are transferred from the contract to B's wallet, and the map debt records that B has a debt of 30 units:

$$S = \mathtt{LP}[120:T, {}^{150}/_{\mathtt{totM}}, {}^{30}/_{\mathtt{totD}}, {}^{B\to 30}/_{\mathtt{debt}}, \cdots] \mid \mathtt{B}[80:T] \mid \cdots$$

Note that the borrow transaction is valid because B's collateralization in S is still above the threshold Cmin: more precisely, the exchange rate in S is $\mathtt{X}(120) = (120+30)/150 = 1$, and B's collateralization is $\mathtt{C}(\mathtt{B}, 120) = 50/30 = 1.67 > \mathtt{Cmin} = 1.5$.

The attack starts when the adversary M observes in the mempool a transaction $X_4 = \mathtt{accrue}(\mathtt{Oracle\ pays}\ 0:T)$ sent by the oracle to update the interest rate. Indeed, after X_4 is executed, B becomes undercollateralized, and so anyone can liquidate part of B's debt, and obtain B's minted tokens with a bonus. To perform the attack, M back-runs X_4 with a sequence of two transactions:

- $X_5 = \mathtt{liquidate}(\mathtt{M\ pays}\ 40:T, \mathtt{B})$, where M obtains minted tokens from B's collateral, at a discounted price;
- $X_6 = \mathtt{redeem}(\mathtt{M\ pays}\ 0:T, 45)$, where M redeems all her minted tokens for T units.

The sequence $X_4X_5X_6$ allows M to extract MEV from the contract. The state resulting after the attack is obtained as follows. The transaction X_4 just updates the interest rate, leading to:

$$S_4 = \mathtt{LP}[120:T, {}^{150}/_{\mathtt{totM}}, {}^{B\to 30}/_{\mathtt{debt}}, {}^{1.5}/_{\mathtt{ir}}] \mid \mathtt{B}[80:T] \mid \cdots$$

Then, the liquidation X_5 transfers $40:T$ from M to the LP, and updates the maps debt and minted:

$$S_5 = \mathtt{LP}[160:T, {}^{3}/_{\mathtt{totD}}, {}^{B\to 3}/_{\mathtt{debt}}, {}^{M\to 147, B\to 3}/_{\mathtt{minted}}, \cdots] \mid \mathtt{M}[60:T] \mid \cdots$$

Finally, M can redeem all her minted tokens with X_6:

$$S_6 = \mathtt{LP}[0:T, {}^{M\to 0, B\to 3}/_{\mathtt{minted}}, \cdots] \mid \mathtt{M}[220:T] \mid \cdots$$

Summing up, M's gain is $(220-200)\cdot \$\mathbf{1}_T = 20\cdot \$\mathbf{1}_T$. Note that there is no way to extract more tokens from the contract (in the last state, the reserves are empty), and that the identity of the adversary is immaterial for the attack. Therefore, $\mathrm{MEV}(S, \{X_4\}) = 20\cdot \$\mathbf{1}_T$.

C Supplementary results and proofs

C.1 Blockchain model (Section 2)

We detail below the proofs of the statements in Section 2, together with additional results.

The finite tokens axiom ensures that the wallet of any set of actors $\mathcal{A}$, defined as the pointwise addition of functions:

$$\omega_{\mathcal{A}}(S) \;=\; \textstyle\sum_{\mathtt{A}\in\mathcal{A}} \omega_{\mathtt{A}}(S)$$

is always defined, and that it has a non-zero amount of tokens only for a *finite* set of token types.

Proposition 11 *For all (possibly infinite) $\mathcal{A}$ and S, $\omega_{\mathcal{A}}(S)$ is defined and has a finite support.*

Proof. Direct consequence of (1).

Furthermore, for any possibly infinite $\mathcal{A}$, there exists a least *finite* subset $\mathcal{A}_0$ which contains exactly the tokens of $\mathcal{A}$. This is also true for all subsets $\mathcal{B}$ of $\mathcal{A}$ including $\mathcal{A}_0$.

Proposition 12 *For all S, $\mathcal{A}$, there exists the least $\mathcal{A}_0 \subseteq_{fin} \mathcal{A}$ such that, for all $\mathcal{B}$, if $\mathcal{A}_0 \subseteq \mathcal{B} \subseteq \mathcal{A}$ then $\omega_{\mathcal{A}_0}(S) = \omega_{\mathcal{B}}(S)$.*

Proof. By definition, $\omega_{\mathcal{A}}(S)\mathtt{T} = \sum_{\mathtt{A}\in\mathcal{A}} \omega_{\mathtt{A}}(S)\mathtt{T}$. By (1), the quantity $\omega_{\mathtt{A}}(S)\mathtt{T}$ is non-zero for only finitely many $\mathtt{A}$ and $\mathtt{T}$. Therefore, the set $\mathcal{A}_0 = \{\mathtt{A} \mid \exists \mathtt{T}.\ \omega_{\mathtt{A}}(S)\mathtt{T} \neq 0\}$ is finite, and $\omega_{\mathcal{A}\setminus\mathcal{A}_0}(S)\mathtt{T} = 0$ for all $\mathtt{T}$. Consequently, $\omega_{\mathcal{A}}(S)\mathtt{T} = \omega_{\mathcal{A}_0}(S)\mathtt{T} \leq \omega_{\mathcal{B}}(S)\mathtt{T} \leq \omega_{\mathcal{A}}(S)\mathtt{T}$. Note that, by construction, $\mathcal{A}_0$ is the least subset of $\mathcal{A}$ preserving the tokens. □

Hereafter, we denote by $\mathbb{N}^{(\mathbb{T})}$ the set of finite-support functions from $\mathbb{T}$ to $\mathbb{N}$, like those resulting from $\omega_{\mathcal{A}}(S)$.

We can extend (3) to the sum of a countable set of wallets:

Proposition 13 *Let $(w_i)_i$ be a countable set of wallets in $\mathbb{N}^{(\mathbb{T})}$ such that $\sum_{i,\mathtt{T}} w_i(\mathtt{T}) \in \mathbb{N}$. Then:*

$$\$\sum_i w_i \;=\; \sum_i \$w_i \;=\; \sum_{i,\mathtt{T}} w_i(\mathtt{T}) \cdot \$\mathbf{1}_{\mathtt{T}}$$

Proof. Since $\sum_i w_i$ is in $\mathbb{T} \to \mathbb{N}$, by (2) we can write:

$$\sum_i w_i \;=\; \sum_{\mathtt{T}} \Big(\sum_i w_i\Big)(\mathtt{T}) \cdot \mathbf{1}_{\mathtt{T}} \;=\; \sum_{\mathtt{T},i} w_i(\mathtt{T}) \cdot \mathbf{1}_{\mathtt{T}} \tag{13}$$

By hypothesis, $w_i(\mathtt{T})$ is almost always zero, so the rhs in (13) is a *finite* sum. Then:

$$
\begin{aligned}
\$\sum_i w_i &= \$\sum_{\mathtt{T},i} w_i(\mathtt{T}) \cdot \mathbf{1}_{\mathtt{T}} && \text{by (13)} \\
&= \sum_{\mathtt{T},i} \$\big(w_i(\mathtt{T}) \cdot \mathbf{1}_{\mathtt{T}}\big) && \text{additivity, finite sum} \\
&= \sum_{\mathtt{T},i} w_i(\mathtt{T}) \cdot \$\mathbf{1}_{\mathtt{T}} && \text{additivity, finite sum} \\
&= \sum_i \sum_{\mathtt{T}} w_i(\mathtt{T}) \cdot \$\mathbf{1}_{\mathtt{T}} \\
&= \sum_i \sum_{\mathtt{T}} \$\big(w_i(\mathtt{T}) \cdot \mathbf{1}_{\mathtt{T}}\big) && \text{additivity, finite sum} \\
&= \sum_i \$\sum_{\mathtt{T}} w_i(\mathtt{T}) \cdot \mathbf{1}_{\mathtt{T}} && \text{additivity, finite sum} \\
&= \sum_i \$w_i && \text{by (2)}
\end{aligned}
$$

□

Note that, since $\mathbb{A}$ is countable, so is any subset $\mathcal{A}$ of $\mathbb{A}$. Therefore, we have the following corollary of Proposition 13.

Proposition 14 *For all S and $\mathcal{A}$, $\$\omega_{\mathcal{A}}(S) = \sum_{\mathtt{A}\in\mathcal{A}} \$\omega_{\mathtt{A}}(S)$.*

We can extend Proposition 12 to wealth, by ensuring that, in any reachable state, only *finitely* many actors have a non-zero wealth:

Proposition 15 *For all S, $\mathcal{A}$, there exists the least $\mathcal{A}_0 \subseteq_{fin} \mathcal{A}$ such that, for all $\mathcal{B}$, if $\mathcal{A}_0 \subseteq \mathcal{B} \subseteq \mathcal{A}$ then $\$\omega_{\mathcal{A}_0}(S) = \$\omega_{\mathcal{B}}(S)$.*

Proof. Let $\mathcal{A}_1 = \{\mathtt{A}_1, \ldots, \mathtt{A}_n\}$ be the finite set given by Proposition 12, for which we have: $\omega_{\mathcal{A}}(S) = \omega_{\mathcal{A}_1}(S) = \sum_{i=1}^{n} \omega_{\mathtt{A}_i}(S)$. Since this is a finite sum, by additivity of \$ we obtain: $\$\omega_{\mathcal{A}}(S) = \$\omega_{\mathcal{A}_1}(S) = \sum_{i=1}^{n} \$\omega_{\mathtt{A}_i}(S)$. Now, let $\mathcal{A}_0 = \{\mathtt{A} \in \mathcal{A}_1 \mid \$\omega_{\mathtt{A}}(S) \neq 0\}$. Since $\mathcal{A}_0$ is finite, we have $\$\omega_{\mathcal{A}}(S) = \$\omega_{\mathcal{A}_1}(S) = \$\omega_{\mathcal{A}_0}(S)$. Now, let $\mathcal{A}_0 \subseteq \mathcal{B} \subseteq \mathcal{A}$. Since \$ is additive, it is also monotonic, and so $\$\omega_{\mathcal{A}}(S) = \$\omega_{\mathcal{A}_0}(S) \leq \$\omega_{\mathcal{B}}(S) \leq \$\omega_{\mathcal{A}}(S)$. Note that, by construction, $\mathcal{A}_0$ is the least set satisfying the statement.

Proof of Proposition 1

We must prove the following statements:

(1) $\gamma_{\mathcal{A}}(S, \mathfrak{X})$ is defined and has a finite integer value
(2) $\gamma_{\mathcal{A}}(S, \mathfrak{X}) = \sum_{\mathtt{A}\in\mathcal{A}} \gamma_{\mathtt{A}}(S, \mathfrak{X})$
(3) there exists $\mathcal{A}_0 \subseteq_{fin} \mathcal{A}$ such that, for all $\mathcal{B}$, if $\mathcal{A}_0 \subseteq \mathcal{B} \subseteq \mathcal{A}$ then $\gamma_{\mathcal{A}_0}(S, \mathfrak{X}) = \gamma_{\mathcal{B}}(S, \mathfrak{X})$.

Item (1) follows from Proposition 11.
Item (2) follows from Propositions 11 and 14.
For Item (3), let $S \xrightarrow{\mathfrak{X}} S'$. By Definition 3, we have that $\gamma_{\mathcal{A}}(S, \mathfrak{X}) = \$\omega_{\mathcal{A}}(S') - \$\omega_{\mathcal{A}}(S)$. By Proposition 15, there exist $\mathcal{A}_1, \mathcal{A}_2 \subseteq_{fin} \mathcal{A}$ such that:

$$\forall \mathcal{B} : \mathcal{A}_1 \subseteq \mathcal{B} \subseteq \mathcal{A} \implies \$\omega_{\mathcal{A}_1}(S) = \$\omega_{\mathcal{B}}(S) \tag{14}$$
$$\forall \mathcal{B} : \mathcal{A}_2 \subseteq \mathcal{B} \subseteq \mathcal{A} \implies \$\omega_{\mathcal{A}_2}(S') = \$\omega_{\mathcal{B}}(S') \tag{15}$$

Let $\mathcal{A}_0 = \mathcal{A}_1 \cup \mathcal{A}_2$, and let $\mathcal{B}$ be such that $\mathcal{A}_0 \subseteq \mathcal{B} \subseteq \mathcal{A}$. Therefore:

$$\begin{aligned} \gamma_{\mathcal{B}}(S, \mathfrak{X}) &= \$\omega_{\mathcal{B}}(S') - \$\omega_{\mathcal{B}}(S) && \text{by Definition 3} \\ &= \$\omega_{\mathcal{A}_0}(S') - \$\omega_{\mathcal{A}_0}(S) && \text{by (14) and (15)} \\ &= \gamma_{\mathcal{A}_0}(S, \mathfrak{X}) \end{aligned}$$ □

C.2 Adversarial knowledge (Section 3.1)

Proof of Proposition 2

For (1), if $\kappa_{\mathcal{A}}(\emptyset) = \kappa_{\mathcal{B}}(\emptyset)$, then by the private knowledge axiom $\mathcal{A} \subseteq \mathcal{B}$ and $\mathcal{B} \subseteq \mathcal{A}$, and so $\mathcal{A} = \mathcal{B}$. Item (2) follows from no shared secrets and monotonicity. The leftmost inclusion of item (3) follows from extensivity and monotonicity, while the rightmost inclusion is given by:

$$\begin{aligned} \kappa_{\mathcal{A}}(\kappa_{\mathcal{B}}(\mathfrak{X})) &\subseteq \kappa_{\mathcal{A}\cup\mathcal{B}}(\kappa_{\mathcal{A}\cup\mathcal{B}}(\mathfrak{X})) && \text{monotonicity, twice} \\ &\subseteq \kappa_{\mathcal{A}\cup\mathcal{B}}(\mathfrak{X}) && \text{idempotence} \end{aligned}$$

For (4), we have:

$$\begin{aligned} \kappa_{\mathcal{B}}(\kappa_{\mathcal{A}}(\mathcal{Y}) \cup \mathfrak{X}) &\subseteq \kappa_{\mathcal{B}}(\kappa_{\mathcal{A}}(\mathfrak{X}) \cup \mathfrak{X}) && \text{monotonicity, } \mathcal{Y} \subseteq \mathfrak{X} \\ &= \kappa_{\mathcal{B}}(\kappa_{\mathcal{A}}(\mathfrak{X})) && \text{extensivity} \\ &\subseteq \kappa_{\mathcal{A}}(\kappa_{\mathcal{A}}(\mathfrak{X})) && \text{monotonicity, } \mathcal{B} \subseteq \mathcal{A} \\ &= \kappa_{\mathcal{A}}(\mathfrak{X}) && \text{idempotence} \end{aligned}$$

For Item (5), let $\mathfrak{X} = \bigcup_i \mathfrak{X}_i$, where $\{\mathfrak{X}_i\}_i$ is an increasing chain of finite sets. By continuity, we have that:

$$\mathsf{X} \in \kappa_{\mathcal{A}}(\mathfrak{X}) = \kappa_{\mathcal{A}}(\textstyle\bigcup_i \mathfrak{X}_i) = \bigcup_i \kappa_{\mathcal{A}}(\mathfrak{X}_i)$$

Therefore, there exists some n such that $\mathsf{X} \in \kappa_{\mathcal{A}}(\mathfrak{X}_n)$. The thesis follows from the fact that $\mathfrak{X}_n$ is a finite subset of $\mathfrak{X}$. □

Proof of Proposition 16

Item (1) is straightforward by Definition 10.
Item (2) is a direct consequence of extensivity.

For (3), w.l.o.g., take a minimal finite set of actors $\mathcal{A}_0$ such that $\mathcal{X} \subseteq \kappa_{\mathcal{A}_0}(\emptyset)$ (this is always possible, since finite sets are well ordered by inclusion). We prove that $\mathcal{A}_0 = \min\{\mathcal{B} \mid \mathcal{X} \subseteq \kappa_{\mathcal{B}}(\emptyset)\}$. By contradiction, let $\mathcal{A}$ be such that $\mathcal{X} \subseteq \kappa_{\mathcal{A}}(\emptyset)$ and $\mathcal{A}_0 \not\subseteq \mathcal{A}$. Let $\mathcal{B} = \mathcal{A}_0 \cap \mathcal{A}$. We have that:

$$\begin{aligned} \mathcal{X} &\subseteq \kappa_{\mathcal{A}_0}(\emptyset) \cap \kappa_{\mathcal{A}}(\emptyset) && \mathcal{X} \subseteq \kappa_{\mathcal{A}_0}(\emptyset),\ \mathcal{X} \subseteq \kappa_{\mathcal{A}}(\emptyset) \\ &\subseteq \kappa_{\mathcal{A}_0 \cap \mathcal{A}}(\emptyset) && \text{no shared secrets} \\ &= \kappa_{\mathcal{B}}(\emptyset) && \mathcal{B} = \mathcal{A}_0 \cap \mathcal{A} \end{aligned}$$

Since $\mathcal{A}_0 \not\subseteq \mathcal{A}$, then $\mathcal{B} \subset \mathcal{A}_0$, and so $\mathcal{A}_0$ is not minimal — contradiction.

For (4), by the finite causes property, there exists a finite $\mathcal{A}_0$ such that $\mathcal{X} \subseteq \kappa_{\mathcal{A}_0}(\emptyset)$. The thesis follows by Item (3). □

C.3 MEV (Section 3)

Proof of Proposition 3

By Definition 4, $\kappa_{\mathcal{A}}(\mathcal{X})$ is defined. By \$-boundedness, $\gamma_{\mathcal{A}}(S, \mathfrak{X})$ is defined and bounded for all $\mathfrak{X}$. The existence of the maximum in Definition 5 follows from the fact that each non-empty upper-bounded subset of $\mathbb{N}$ admits a maximum. This maximum exists even though $\kappa_{\mathtt{A}}(-)^*$ generates an infinite set of sequences, of unbounded (but finite) length. The MEV is non-negative since the empty sequence of transactions gives a zero gain. □

We denote by $\alpha(\mathcal{X})$ the least set of actors who are needed to craft all the transactions in $\mathcal{X}$. When $\alpha(\mathcal{X}) = \emptyset$, then $\mathcal{X}$ can be created by anyone, without requiring any private knowledge.

Definition 10 (Transaction authoriser) *For all $\mathcal{X}$, we define:*

$$\alpha(\mathcal{X}) \;=\; \min\{\mathcal{A} \mid \mathcal{X} \subseteq \kappa_{\mathcal{A}}(\emptyset)\}$$

when such a minimum exists.

Proposition 16 establishes some properties of authorisers. Item (2) states that the authorisers of $\mathcal{X}$ can generate at least $\mathcal{X}$. Items (3) and (4) give sufficient conditions for $\alpha(\mathcal{X})$ to be defined: indeed, $\alpha(\mathcal{X})$ may be undefined for some $\mathcal{X}$, since the set of $\mathcal{A}$ such that $\mathcal{X} \subseteq \kappa_{\mathcal{A}}(\emptyset)$ may not have a minimum. Item (3) guarantees that $\alpha(\mathcal{X})$ is defined (and finite) whenever $\mathcal{X}$ is inferred from an empty knowledge. Item (4) gives the same guarantee when $\mathcal{X}$ is finite.

Proposition 16 *For all $\mathcal{X}$ and $\mathcal{A}$, we have that:*

(1) $\alpha(\emptyset) = \emptyset$
(2) *if $\alpha(\mathcal{X})$ is defined, then $\mathcal{X} \subseteq \kappa_{\alpha(\mathcal{X})}(\emptyset)$*
(3) *if $\mathcal{X} \subseteq \kappa_{\mathcal{A}}(\emptyset)$ with $\mathcal{A}$ finite, $\alpha(\mathcal{X})$ is defined and finite*
(4) *if $\mathcal{X}$ is finite, then $\alpha(\mathcal{X})$ is defined and finite.*

For a TxScript transaction $\mathsf{X} = \mathtt{f}(\overrightarrow{txarg})$, the authorizers are the $\mathtt{A}$ such that $\overrightarrow{txarg}$ contains $\mathtt{A}$ `pays` $n\,\mathtt{:T}$, for some $n \in \mathbb{N}$ and $\mathtt{T} \in \mathbb{T}$, or involves a secret s generated by $\mathtt{A}$. For instance, in `BadHTLC` (Example 1) we have: $\alpha(\mathsf{X}_{\mathtt{A}}) = \{\mathtt{A}\}$, $\alpha(\mathsf{X}_{\mathtt{M}}) = \{\mathtt{M}, \mathtt{A}\}$, and $\alpha(\mathsf{Y}_{\mathtt{M}}) = \{\mathtt{M}, \mathtt{Oracle}\}$.

A special case of Proposition 4 is when the actors in $\mathcal{A}$ have enough knowledge to generate the set $\mathcal{X}$ by themselves. In this case, the MEV extractable by $\mathcal{A}$ is that of a displacement attack:

Corollary 1 *If $\alpha(\mathcal{X}) \subseteq \mathcal{A}$, then $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) = \mathrm{MEV}_{\mathcal{A}}(S, \emptyset)$.*

Proof. Consequence of Proposition 4, using the inequality:

$$\begin{aligned}
\kappa_{\mathcal{A}}(\mathcal{X}) &\subseteq \kappa_{\mathcal{A}}(\kappa_{\alpha(\mathcal{X})}(\emptyset)) && \text{Proposition 16(2)} \\
&\subseteq \kappa_{\mathcal{A} \cup \alpha(\mathcal{X})}(\emptyset) && \text{Proposition 2(3)} \\
&\subseteq \kappa_{\mathcal{A}}(\emptyset) && \alpha(\mathcal{X}) \subseteq \mathcal{A}
\end{aligned}$$

□

Proof of Proposition 4

We have that $\mathcal{X} = \mathcal{Y} \cup \mathcal{Z}$, where $\mathcal{Y} = \mathcal{X} \setminus \kappa_{\mathcal{A}}(\emptyset)$ and $\mathcal{Z} = \mathcal{X} \cap \kappa_{\mathcal{A}}(\emptyset)$. We show that $\kappa_{\mathcal{A}}(\mathcal{X}) = \kappa_{\mathcal{A}}(\mathcal{Y})$. The inclusion $\supseteq$ follows by monotonicity of κ. For $\subseteq$, we have that:

$$\begin{aligned}
\kappa_{\mathcal{A}}(\mathcal{X}) &= \kappa_{\mathcal{A}}(\mathcal{Y} \cup \mathcal{Z}) && \mathcal{X} = \mathcal{Y} \cup \mathcal{Z} \\
&\subseteq \kappa_{\mathcal{A}}(\mathcal{Y} \cup \kappa_{\mathcal{A}}(\emptyset)) && \text{monotonicity, } \mathcal{Z} \subseteq \kappa_{\mathcal{A}}(\emptyset) \\
&\subseteq \kappa_{\mathcal{A}}(\mathcal{Y} \cup \kappa_{\mathcal{A}}(\mathcal{Y})) && \text{monotonicity} \\
&= \kappa_{\mathcal{A}}(\kappa_{\mathcal{A}}(\mathcal{Y})) && \text{extensivity} \\
&= \kappa_{\mathcal{A}}(\mathcal{Y}) && \text{idempotence}
\end{aligned}$$

□

Proof of Proposition 5

Let $\boldsymbol{\mathcal{X}} \in \kappa_{\mathcal{A}}(\mathcal{X})^*$ maximize $\gamma_{\mathcal{A}}(S, \cdot)$. By monotonicity of κ we have that $\kappa_{\mathcal{A}}(\mathcal{X}) \subseteq \kappa_{\mathcal{A}'}(\mathcal{X}')$. Hence, $\boldsymbol{\mathcal{X}} \in \kappa_{\mathcal{A}'}(\mathcal{X}')^*$. The thesis follows from Proposition 1(2). □

Proof of Proposition 7

Let $\mathcal{B} \subseteq \mathcal{A}$. By Definition 5, we have that:

$$\mathrm{MEV}_{\mathcal{B}}(S, \mathcal{X}) \;=\; \gamma_{\mathcal{B}}(S, \boldsymbol{\mathcal{X}}) \qquad \text{for some } \boldsymbol{\mathcal{X}} \in \kappa_{\mathcal{B}}(\mathcal{X})^*$$

Since $\boldsymbol{\mathcal{X}}$ is made of a finite set of transactions, by the finite causes property of κ there exists a *finite* $\mathcal{C}$ such that $\boldsymbol{\mathcal{X}} \in \kappa_{\mathcal{C}}(\emptyset)^*$. Then, by monotonicity of κ it follows that $\boldsymbol{\mathcal{X}} \in \kappa_{\mathcal{C}}(\emptyset)^* \subseteq \kappa_{\mathcal{C}}(\mathcal{X})^*$. Let $\mathcal{C}_0 = \mathcal{C} \cap \mathcal{B}$. By the "no shared secrets" property of κ:

$$\begin{aligned}
\boldsymbol{\mathcal{X}} \in \kappa_{\mathcal{C}}(\mathcal{X})^* \cap \kappa_{\mathcal{B}}(\mathcal{X})^* &= (\kappa_{\mathcal{C}}(\mathcal{X}) \cap \kappa_{\mathcal{B}}(\mathcal{X}))^* \\
&\subseteq \kappa_{\mathcal{C} \cap \mathcal{B}}(\mathcal{X})^* = \kappa_{\mathcal{C}_0}(\mathcal{X})^*
\end{aligned}$$

By Proposition 1(3), there exists a finite $\mathcal{C}_1 \subseteq_{fin} \mathcal{B}$ such that $\gamma_{\mathcal{B}}(S, \mathfrak{X}) = \gamma_{\mathcal{C}_1}(S, \mathfrak{X})$. Let $\mathcal{A}_0 = \mathcal{C}_0 \cup \mathcal{C}_1 \subseteq \mathcal{B}$. We have that:

$$\gamma_{\mathcal{B}}(S, \mathfrak{X}) = \gamma_{\mathcal{C}_1}(S, \mathfrak{X}) = \gamma_{\mathcal{A}_0}(S, \mathfrak{X})$$

Since $\mathfrak{X} \in \kappa_{\mathcal{C}_0}(\mathcal{X})^*$, the monotonicity of κ implies $\mathfrak{X} \in \kappa_{\mathcal{A}_0}(\mathcal{X})^*$. Summing up:

$$\mathrm{MEV}_{\mathcal{B}}(S, \mathcal{X}) \;=\; \gamma_{\mathcal{B}}(S, \mathfrak{X}) \;=\; \gamma_{\mathcal{A}_0}(S, \mathfrak{X}) \;=\; \mathrm{MEV}_{\mathcal{A}_0}(S, \mathcal{X}) \qquad \square$$

Proof of Proposition 8

By Definition 5, we have that:

$$\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) \;=\; \gamma_{\mathcal{A}}(S, \mathfrak{X}) \qquad \text{for some } \mathfrak{X} = \mathsf{X}_1 \cdots \mathsf{X}_n \in \kappa_{\mathcal{A}}(\mathcal{X})^*$$

By Proposition 2(5), for all $i \in 1..n$ there exists $\mathcal{X}_i \subseteq_{fin} \mathcal{X}$ such that $\mathsf{X}_i \in \kappa_{\mathcal{A}}(\mathcal{X}_i)$. Let $\mathcal{X}_0 = \bigcup_{i \in 1..n} \mathcal{X}_i \subseteq_{fin} \mathcal{X}$. By monotonicity of κ, we have $\mathsf{X}_i \in \kappa_{\mathcal{A}}(\mathcal{X}_0)$ for all $i \in 1..n$. Therefore, $\mathfrak{X} \in \kappa_{\mathcal{A}}(\mathcal{X}_0)^*$. $\square$

C.4 Bad MEV (Section 3)

Proposition 17 *$\mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}}(S, \mathcal{X})$ is defined and has a non-negative value.*

Proof. Immediate from Proposition 3, Proposition 5, and Definition 7. $\square$

Proposition 18 $\mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}}(S, \mathcal{X}) = \mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}}(S, \mathcal{X} \setminus \kappa_{\mathcal{A}}(\emptyset))$

Proof. We have that:

$$\begin{aligned}
\mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}}(S, \mathcal{X}) &= \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) - \mathrm{MEV}_{\mathcal{A}}(S, \emptyset) && \text{by Definition 7} \\
&= \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X} \setminus \kappa_{\mathcal{A}}(\emptyset)) - \mathrm{MEV}_{\mathcal{A}}(S, \emptyset) && \text{by Proposition 4} \\
&= \mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}}(S, \mathcal{X} \setminus \kappa_{\mathcal{A}}(\emptyset)) && \text{by Definition 7} \qquad \square
\end{aligned}$$

Proposition 19 *If $\mathcal{X} \subseteq \mathcal{X}'$, then* $\mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}}(S, \mathcal{X}) \leq \mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}}(S, \mathcal{X}')$.

Proof. We have that:

$$\begin{aligned}
\mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}}(S, \mathcal{X}) &= \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) - \mathrm{MEV}_{\mathcal{A}}(S, \emptyset) && \text{by Definition 7} \\
&\leq \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}') - \mathrm{MEV}_{\mathcal{A}}(S, \emptyset) && \text{by Proposition 5} \\
&= \mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}}(S, \mathcal{X}') && \text{by Definition 7} \qquad \square
\end{aligned}$$

Proposition 20 *For all $\mathcal{A}$, $\mathcal{X}$ and S, there exists $\mathcal{A}_0 \subseteq_{fin} \mathcal{A}$ such that, for all $\mathcal{B}$, if $\mathcal{A}_0 \subseteq \mathcal{B} \subseteq \mathcal{A}$ then:* $\mathrm{MEV}^{\mathrm{bad}}_{\mathcal{A}_0}(S, \mathcal{X}) = \mathrm{MEV}^{\mathrm{bad}}_{\mathcal{B}}(S, \mathcal{X})$.

Proof. By Proposition 7 (applied twice), there exist $\mathcal{A}_0 \subseteq_{fin} \mathcal{A}$ and $\mathcal{A}_1 \subseteq_{fin} \mathcal{A}$ such that, for all $\mathcal{B}_0$ and $\mathcal{B}_1$:

$$\mathcal{A}_0 \subseteq \mathcal{B}_0 \subseteq \mathcal{A} \implies \mathrm{MEV}_{\mathcal{A}_0}(S, \mathcal{X}) = \mathrm{MEV}_{\mathcal{B}_0}(S, \mathcal{X}) \tag{16}$$

$$\mathcal{A}_1 \subseteq \mathcal{B}_1 \subseteq \mathcal{A} \implies \mathrm{MEV}_{\mathcal{A}_1}(S, \emptyset) = \mathrm{MEV}_{\mathcal{B}_1}(S, \emptyset) \tag{17}$$

Let $\mathcal{A}_2 = \mathcal{A}_0 \cup \mathcal{A}_1$, which is finite. Then, for all $\mathcal{B}$ such that $\mathcal{A}_2 \subseteq_{fin} \mathcal{B} \subseteq \mathcal{A}$:

$$\begin{aligned}
\mathrm{MEV}^{\mathrm{bad}}{}_{\mathcal{A}}(S, \mathcal{X}) &= \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) - \mathrm{MEV}_{\mathcal{A}}(S, \emptyset) && \text{by Definition 7} \\
&= \mathrm{MEV}_{\mathcal{B}}(S, \mathcal{X}) - \mathrm{MEV}_{\mathcal{B}}(S, \emptyset) && \text{by (16),(17)} \\
&= \mathrm{MEV}^{\mathrm{bad}}{}_{\mathcal{B}}(S, \mathcal{X}) && \square
\end{aligned}$$

Proposition 21 *For all $\mathcal{A}$, $\mathcal{X}$ and S, there exists $\mathcal{X}_0 \subseteq_{fin} \mathcal{X}$ such that* $\mathrm{MEV}^{\mathrm{bad}}{}_{\mathcal{A}}(S, \mathcal{X}_0) = \mathrm{MEV}^{\mathrm{bad}}{}_{\mathcal{A}}(S, \mathcal{X})$.

Proof. By Proposition 21, there exists $\mathcal{X}_0 \subseteq_{fin} \mathcal{X}$ such that $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}_0) = \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X})$. Therefore, by Definition 7:

$$\begin{aligned}
\mathrm{MEV}^{\mathrm{bad}}{}_{\mathcal{A}}(S, \mathcal{X}) &= \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) - \mathrm{MEV}_{\mathcal{A}}(S, \emptyset) \\
&= \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}_0) - \mathrm{MEV}_{\mathcal{A}}(S, \emptyset) \\
&= \mathrm{MEV}^{\mathrm{bad}}{}_{\mathcal{A}}(S, \mathcal{X}_0) && \square
\end{aligned}$$

Proof of Proposition 9

Consequence of Proposition 17, Proposition 18, Proposition 19, Proposition 20, Proposition 21. □

C.5 Preservation under renaming

In general, renaming actors in $\mathcal{A}$ may affect $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X})$. However, in most real-world contracts there exist sets of actors that are *indistinguishable* from each other when looking at their interaction capabilities with the contract. We formalise as *clusters* these sets of actors, and we show in Theorem 6 that MEV is invariant w.r.t. renaming actors in a cluster.

Definition 11 formalises the effect of renaming a set of actors $\mathcal{A}$ on states. We say that ρ is a *renaming of* $\mathcal{A}$ if $\rho \in \mathcal{A} \to \mathcal{A}$ is a permutation. Applying ρ to a blockchain state (W, C) only affects the wallet state, leaving the contract state unaltered. The renamed wallet state $W\rho$ is defined as follows.

Definition 11 *For a wallet state W and renaming ρ of $\mathcal{A}$, let:*

$$(W\rho)(\mathtt{A}) = \begin{cases} W(\rho^{-1}(\mathtt{A})) & \text{if } \mathtt{A} \in \mathcal{A} \\ W(\mathtt{A}) & \text{otherwise} \end{cases}$$

We define the renaming of (W, C) as $(W, C)\rho = (W\rho, C)$.

As a basic example, for $W = \mathtt{A}_0[0:\mathtt{T}] \mid \mathtt{A}_1[1:\mathtt{T}] \mid \mathtt{A}_2[2:\mathtt{T}]$ and $\rho = \{\mathtt{A}_1/\mathtt{A}_0, \mathtt{A}_2/\mathtt{A}_1, \mathtt{A}_0/\mathtt{A}_2\}$ is a renaming of $\{\mathtt{A}_0, \mathtt{A}_1, \mathtt{A}_2\}$, then $W\rho = \mathtt{A}_1[0:\mathtt{T}] \mid \mathtt{A}_2[1:\mathtt{T}] \mid \mathtt{A}_0[2:\mathtt{T}]$.

Intuitively, a set $\mathcal{A}$ is a cluster when, for each renaming ρ of $\mathcal{A}$ and each subset $\mathcal{B}$ of $\mathcal{A}$, if $\mathcal{B}$ can fire a sequence of transactions leading to a given wallet state, then also $\rho(\mathcal{B})$ can achieve the same effect, up-to the renaming ρ.

Definition 12 (Cluster of actors) *A set of actors $\mathcal{A} \subseteq \mathbb{A}$ is a* cluster in $(S, \mathcal{X})$ *iff, for all renamings ρ_0, ρ_1 of $\mathcal{A}$, and for all $\mathcal{B} \subseteq \mathcal{A}$, S', $\boldsymbol{\mathcal{X}}$, and $\mathcal{Y} \subseteq \mathcal{X}$, if*

$$S\rho_0 \xrightarrow{\boldsymbol{\mathcal{X}}} S' \qquad \text{with } \boldsymbol{\mathcal{X}} \in \kappa_{\rho_0(\mathcal{B})}(\mathcal{Y})^*$$

then there exist $\boldsymbol{\mathcal{X}'}$ and R' such that:

$$S\rho_1 \xrightarrow{\boldsymbol{\mathcal{X}'}} R' \text{ with } \boldsymbol{\mathcal{X}'} \in \kappa_{\rho_1(\mathcal{B})}(\mathcal{Y})^* \text{ and } \omega(R') = \omega(S')\rho_0^{-1}\rho_1$$

We establish below two basic properties of clusters. First, any subset of a cluster is still a cluster; second, if $\mathcal{A}$ is a cluster in S then it is also a cluster of any $\mathcal{A}$-renaming of S.

Lemma 1 *Let $\mathcal{A}$ be a cluster in $(S, \mathcal{X})$. Then:*

(1) *if $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{X}' \subseteq \mathcal{X}$, then $\mathcal{A}'$ is a cluster in $(S, \mathcal{X}')$;*
(2) *for all renamings ρ of $\mathcal{A}$, $\mathcal{A}$ is a cluster in $(S\rho, \mathcal{X})$.*

Proof. For item (1), let $\mathcal{A}' \subseteq \mathcal{A}$, let $\mathcal{X}' \subseteq \mathcal{X}$, and let ρ_0, ρ_1 be renamings of $\mathcal{A}'$. For $i \in \{0, 1\}$, let:

$$\rho_i' \;=\; \lambda x.\ \textit{if } x \in \mathcal{A}' \textit{ then } \rho_i(x) \textit{ else } x$$

Clearly, ρ_i' is a renaming of $\mathcal{A}$ and $S\rho_i = S\rho_i'$ for $i \in \{0, 1\}$. Let $\mathcal{B} \subseteq \mathcal{A}'$, $\mathcal{Y} \subseteq \mathcal{X}'$, and S', $\boldsymbol{\mathcal{X}}$ as in the hypotheses. Since $\mathcal{B} \subseteq \mathcal{A}$, $\mathcal{Y} \subseteq \mathcal{X}$, and $\rho_i(\mathcal{B}) = \rho_i'(\mathcal{B})$, the other premises remain true: hence, we can exploit the fact that $\mathcal{A}$ is a cluster in $(S, \mathcal{X})$ to prove that $\mathcal{A}'$ is a cluster in $(S, \mathcal{X}')$.

For item (2), let ρ, ρ_0, ρ_1 be renamings of $\mathcal{A}$. Note that also $\rho_i \circ \rho$ is a renaming of $\mathcal{A}$ for $i \in \{0, 1\}$. Let $\mathcal{B} \subseteq \mathcal{A}$, let $\mathcal{Y} \subseteq \mathcal{X}$, and let $(S\rho)\rho_0 \xrightarrow{\boldsymbol{\mathcal{X}}} S'$, with $\boldsymbol{\mathcal{X}} \in \kappa_{\rho_0(\mathcal{B})}(\mathcal{Y})^*$. We have that $(S\rho)\rho_0 = S(\rho_0 \circ \rho)$ and, for $\mathcal{B}' = \rho^{-1}(\mathcal{B})$, $\boldsymbol{\mathcal{X}} \in \kappa_{(\rho_0 \circ \rho)(\mathcal{B}')}(\mathcal{Y})^*$. Let $\rho_0' = \rho_0 \circ \rho$ and $\rho_1' = \rho_1 \circ \rho$. Since $\mathcal{A}$ is a cluster in $(S, \mathcal{X})$, then there exist $\boldsymbol{\mathcal{X}'}$ and R' such that:

1. $S\rho_1' \xrightarrow{\boldsymbol{\mathcal{X}'}} R'$
2. $\boldsymbol{\mathcal{X}'} \in \kappa_{\rho_1'(\mathcal{B}')}(\mathcal{Y})^*$
3. $\omega(S')(\rho_1' \circ \rho_0'^{-1}) = \omega(R')$.

The thesis follows from:

1. $S\rho_1' = S(\rho_1 \circ \rho) = (S\rho)\rho_1$
2. $\rho_1'(\mathcal{B}') = (\rho_1 \circ \rho)(\rho^{-1}(\mathcal{B})) = \rho_1(\mathcal{B})$

3. Since $\rho_0 = \rho_0' \circ \rho^{-1}$ and $\rho_1 = \rho_1' \circ \rho^{-1}$, then

$$\begin{aligned}\omega(S')(\rho_1 \circ \rho_0^{-1}) &= \omega(S')((\rho_1' \circ \rho^{-1}) \circ (\rho_0' \circ \rho^{-1})^{-1}) \\ &= \omega(S')((\rho_1' \circ \rho^{-1}) \circ (\rho \circ \rho_0'^{-1})) \\ &= \omega(S')(\rho_1' \circ \rho_0'^{-1}) \\ &= \omega(R')\end{aligned}$$

□

Example 3. In the `CoinPusher` contract in Figure 7, for any S and for $\mathcal{X} = \{\texttt{play}(\texttt{A}_i\ \texttt{pays}\ n_i\!:\!\texttt{T})\}_{i=1..n}$, we have that $\mathcal{C} = \mathbb{A} \setminus \{\texttt{A}_i\}_{i=1..n}$ is a cluster in $(S, \mathcal{X})$. To see how a sequence $\mathcal{X}$ is transformed into an equivalent $\mathcal{X}'$, consider the case $n = 1$, $\texttt{A}_1 = \texttt{A}$ and $n_1 = 99$. Let $\mathbb{A} = \{\texttt{A}_i\}_{i\in\mathbb{N}}$, let $\mathcal{B} \subseteq \mathcal{C}$, $\mathcal{Y} \subseteq \mathcal{X}$, and let ρ_0, ρ_1 be renamings of $\mathcal{C}$. Any $\mathcal{X} \in \kappa_{\rho_0(\mathcal{B})}(\mathcal{Y})^*$ is a sequence of transactions of the form $\texttt{play}(\texttt{B}_i\ \texttt{pays}\ v_i\!:\!\texttt{T})$, with $\texttt{B}_i \in \rho_0(\mathcal{B}) \cup \{\texttt{A}\}$. We craft $\mathcal{X}' \in \kappa_{\rho_1(\mathcal{B})}(\mathcal{Y})^*$ from $\mathcal{X}$, by renaming with $\rho_0^{-1}\rho_1$ the actors in $\mathcal{X}$. For instance, consider a state S where the contract has 0 tokens, and:

$$S = \texttt{A}_0[99\!:\!\texttt{T}] \mid \texttt{A}_1[1\!:\!\texttt{T}] \mid \texttt{A}_2[0\!:\!\texttt{T}] \mid \texttt{A}_3[0\!:\!\texttt{T}] \mid \cdots$$

Let $\rho_0 = \{\texttt{A}_1/\texttt{A}_0, \texttt{A}_2/\texttt{A}_1, \texttt{A}_0/\texttt{A}_2\}$, let $\mathcal{B} = \{\texttt{A}_0, \texttt{A}_1\}$, and let

$$\mathcal{X} = \texttt{play}(\texttt{A}_1\ \texttt{pays}\ 99\!:\!\texttt{T})\ \texttt{play}(\texttt{A}_2\ \texttt{pays}\ 1\!:\!\texttt{T}) \in \kappa_{\rho_0(\mathcal{B})}(\mathcal{Y})$$

Now, let $\rho_1 = \{\texttt{A}_3/\texttt{A}_0, \texttt{A}_0/\texttt{A}_1, \texttt{A}_1/\texttt{A}_3\}$. We define $\mathcal{X}'$ as:

$$\mathcal{X}' = \texttt{play}(\texttt{A}_3\ \texttt{pays}\ 99\!:\!\texttt{T})\ \texttt{play}(\texttt{A}_0\ \texttt{pays}\ 1\!:\!\texttt{T}) \in \kappa_{\rho_1(\mathcal{B})}(\mathcal{Y})$$

Firing $\mathcal{X}$ and $\mathcal{X}'$, respectively, from $S\rho_0$ and $S\rho_1$ leads to:

$$\begin{aligned}S\rho_0 &\xrightarrow{\mathcal{X}} \texttt{A}_0[0\!:\!\texttt{T}] \mid \texttt{A}_1[0\!:\!\texttt{T}] \mid \texttt{A}_2[100\!:\!\texttt{T}] \mid \texttt{A}_3[0\!:\!\texttt{T}] \mid \cdots \\ S\rho_1 &\xrightarrow{\mathcal{X}'} \texttt{A}_0[100\!:\!\texttt{T}] \mid \texttt{A}_1[0\!:\!\texttt{T}] \mid \texttt{A}_2[0\!:\!\texttt{T}] \mid \texttt{A}_3[0\!:\!\texttt{T}] \mid \cdots\end{aligned}$$

and the two states are equivalent up-to renaming. ◇

Example 4. The contract `DoubleAuth` in Figure 13 allows anyone to withdraw once 2 out of 3 actors among `A`, `B` and the owner give their authorization. The contract has two hard-coded actors `A` and `B` who have a privileged status (they can authorise the `withdraw`), and so cannot be replaced by others. Further, once `DoubleAuth` has been initialised, also the owner (say, `O`) acquires the same privilege, and so also `O` is not replaceable by others. We prove that neither `A` nor `B` nor `O` can belong to large enough clusters.

Let S be the state reached upon firing $\texttt{init}(\texttt{O}\ \texttt{pays}\ 1\!:\!\texttt{T})$ from an initial state S_0 such that $\omega_{\texttt{O}}(S_0) = \mathbf{1}_{\texttt{T}}$ and $\omega_a(S_0)(t) = 0$ for all $(a, t) \neq (\texttt{O}, \texttt{T})$. We prove that no cluster with size greater than 3 can include `A`, `B` or `O`.

By contradiction, assume that $\mathcal{A}$ is a cluster in S with $|\mathcal{A}| > 3$ and $\texttt{A} \in \mathcal{A}$. Let ρ_0 be the identity, and let ρ_1 be such that $\rho_1(\texttt{A}) = \texttt{M}$, where $\texttt{M} \in \mathcal{A} \setminus \{\texttt{A}, \texttt{B}, \texttt{O}\}$

```
contract DoubleAuth {
  init(a pays x:T) {
    require owner==null;
    owner=a;
  }
  auth1(a sig) {
    require owner!=null && c1==null;
    require a==A || a==B || a==owner;
    c1=a;
  }
  auth2(a sig) {
    require c1!=null && c2=null && c1!=a;
    require a==A || a==B || a==owner;
    c2=a;
  }
  withdraw(a sig) {
    require c1!=null && c2!=null;
    transfer(a,balance(T):T);
  }
}
```

Fig. 13. A contract requiring two authorizations.

(such an M always exists by the hypothesis on the size of $\mathcal{A}$). Let $\mathcal{B} = \{\mathtt{A}\} \subseteq \mathcal{A}$, and let $\mathcal{X} = \mathtt{auth1}(\mathtt{A}\ \mathtt{pays}\ 0{:}\mathtt{T})\ \mathtt{auth2}(\mathtt{B}\ \mathtt{pays}\ 0{:}\mathtt{T})\ \mathtt{withdraw}(\mathtt{A}\ \mathtt{pays}\ 0{:}\mathtt{T})$ be a sequence in $\kappa_{\mathcal{A}}(\mathcal{Y})^*$ for some $\mathcal{Y}$ with $\alpha(\mathcal{Y}) \cap \mathcal{A} = \emptyset$ and $\mathtt{O} \notin \alpha(\mathcal{Y})$ (this can always be satisfied by choosing $\mathcal{Y} = \emptyset$ if $\mathtt{B} \in \mathcal{A}$, and $\mathcal{Y} = \{\mathtt{B}\}$ otherwise). Since $\mathcal{A}$ is a cluster in S, there must exist $\mathcal{X}'$ and R' satisfying Definition 12. Firing the withdraw requires $\alpha(\mathcal{Y}) \cup \rho_1(\mathcal{A})$ to contain at least two distinct actors among A, B and O, while by the hypotheses above we have that $\alpha(\mathcal{Y}) \cup \rho_1(\mathcal{A}) \subseteq \{\mathtt{B}, \mathtt{M}\}$. Therefore, $\mathcal{X}'$ cannot contain a valid withdraw, and so M has no tokens in R'. Instead, upon firing $\mathcal{X}$ we have that A owns $1 : \mathtt{T}$. This violates Definition 12, which requires $\omega(R')\mathtt{M} = (\omega(S')\rho^{-1})\mathtt{M} = \omega(S')\rho(\mathtt{M}) = \omega(S')\mathtt{A}$. Proving that B or O cannot belong to clusters of size greater than 3 in S is done similarly. ◇

Example 5. Recall the contract CoinPusher in Figure 7. For any S and for $\mathcal{X} = \{\mathtt{play}(\mathtt{A}_i\ \mathtt{pays}\ n_i{:}\mathtt{T})\}_{i=1..n}$, we have that $\mathcal{C} = \mathbb{A} \setminus \{\mathtt{A}_i\}_{i=1..n}$ is a cluster in $(S, \mathcal{X})$. ◇

The following theorem establishes that renaming the actors in a cluster preserves their MEV.

Theorem 6 (Preservation under renaming) *Let $\mathcal{A}$ be a cluster in $(S, \mathcal{X})$ and let ρ be a renaming of $\mathcal{A}$. For all $\mathcal{B} \subseteq \mathcal{A}$:*

$$\mathrm{MEV}_{\mathcal{B}}(S, \mathcal{X}) \;=\; \mathrm{MEV}_{\rho(\mathcal{B})}(S\rho, \mathcal{X})$$

Proof. Since $\mathcal{A}$ is a cluster in $(S, \mathcal{X})$, then by Lemma 1(2), $\mathcal{A}$ is also a cluster in $(S\rho, \mathcal{X})$. Hence, by symmetry it is enough to prove $\leq$ between MEVs, which is implied by:

$$\max \gamma_{\mathcal{B}}(S, \kappa_{\mathcal{B}}(\mathcal{X})^*) \leq \max \gamma_{\rho(\mathcal{B})}(S\rho, \kappa_{\rho(\mathcal{B})}(\mathcal{X})^*) \tag{18}$$

To prove (18), let $S \xrightarrow{\mathfrak{X}} S'$, where $\mathfrak{X} \in \kappa_{\mathcal{B}}(\mathcal{X})^*$ maximises the gain of $\mathcal{B}$. Since $\mathcal{B} \subseteq \mathcal{A}$, by Definition 12 (choosing the identity function for the first renaming ρ_0 and ρ for the second renaming ρ_1), there exist $\mathfrak{X}'$ and R' such that $S\rho \xrightarrow{\mathfrak{X}'} R'$ with $\mathfrak{X}' \in \kappa_{\rho(\mathcal{B})}(\mathcal{X})^*$ and $\omega(S')\rho = \omega(R')$. Then, $\omega_{\rho(\mathcal{B})}(R') = \omega_{\mathcal{B}}(S')$. From this we obtain:

$$\begin{aligned}\max \gamma_{\mathcal{B}}(S, \kappa_{\mathcal{B}}(\mathcal{X})^*) &= \gamma_{\mathcal{B}}(S, \mathfrak{X}) \\ &= \gamma_{\rho(\mathcal{B})}(S\rho, \mathfrak{X}') \\ &\leq \max \gamma_{\rho(\mathcal{B})}(S\rho, \kappa_{\rho(\mathcal{B})}(\mathcal{X})^*)\end{aligned}$$ □

The following proposition shows that infinite clusters with a positive MEV are also MEV-attackers. This is an important sanity check for our definition of MEV-attacker, since actors in the same cluster can be renamed without affecting their MEV.

Proposition 22 *Let $\mathcal{A}$ be an infinite cluster of $(S, \mathcal{X})$ such that $\mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) > 0$. Then, $\mathcal{A}$ is a MEV-attacker in $(S, \mathcal{X})$.*

Proof. By Proposition 7, there exists some $\mathcal{A}_0 \subseteq_{fin} \mathcal{A}$ such that $\mathrm{MEV}_{\mathcal{A}_0}(S, \mathcal{X}) = \mathrm{MEV}_{\mathcal{B}}(S, \mathcal{X}) = \mathrm{MEV}_{\mathcal{A}}(S, \mathcal{X}) > 0$ for all $\mathcal{A}_0 \subseteq \mathcal{B} \subseteq \mathcal{A}$. Let $\mathcal{B}$ be an infinite subset of $\mathcal{A}$. Take ρ be any renaming of $\mathcal{A}$ such that $\mathcal{A}_0 \subseteq \rho(\mathcal{B}) \subseteq \mathcal{A}$ and $\mathcal{A} \setminus \rho(\mathcal{B})$ has no tokens in S (note that such a renaming always exists, whether $\mathcal{B}$ is cofinite or not). Then, $\mathrm{MEV}_{\rho(\mathcal{B})}(S, \mathcal{X}) > 0$. Since $\mathcal{A}$ is a cluster and ρ^{-1} is a renaming of $\mathcal{A}$, by Theorem 6 we have:

$$\mathrm{MEV}_{\mathcal{B}}(S\rho^{-1}, \mathcal{X}) = \mathrm{MEV}_{\rho^{-1}(\rho(\mathcal{B}))}(S\rho^{-1}, \mathcal{X}) = \mathrm{MEV}_{\rho(\mathcal{B})}(S, \mathcal{X}) > 0$$

To conclude, we note that $S\rho^{-1}$ is a $(\mathcal{A}, \mathcal{B})$-wallet redistribution of S. Item 4 of Definition 8 is trivial; for the other items, we must show that the tokens of $\mathcal{A}$ in S are tokens of $\mathcal{B}$ in $S\rho^{-1}$. In particular, it suffices to prove that $\mathcal{A} \setminus \mathcal{B}$ has no tokens in $S\rho^{-1}$. By contradiction, let $\mathtt{A} \in \mathcal{A} \setminus \mathcal{B}$ have tokens in $S\rho^{-1}$. This implies that $\rho(\mathtt{A}) \in \rho(\mathcal{A} \setminus \mathcal{B}) = \mathcal{A} \setminus \rho(\mathcal{B})$ must have tokens in $S\rho^{-1}\rho = S$. This contradicts the assumption that $\mathcal{A} \setminus \rho(\mathcal{B})$ has no tokens in S.

C.6 Universal MEV (Section 3.3)

Proof of Theorem 1

Consequence of Proposition 5 and Proposition 19.

Proof of Theorem 2

It is sufficient to prove that for all $\mathcal{A}$ and for all $S \approx_{\$} S'$, there exists some W'_{Δ} and token redistribution $S + W_{\Delta} \approx_{\$} S' + W'_{\Delta}$ such that $\mathrm{MEV}_{\mathcal{A}}(S', \mathcal{X}) \leq \mathrm{MEV}_{\mathcal{A}}(S' + W'_{\Delta}, \mathcal{X})$. It is easy to transform the first token redistribution into the second one: it suffices to arbitrarily reassign the tokens in W_{Δ}. By Theorem 2, $\mathrm{MEV}_{\mathcal{A}}(S', \mathcal{X}) \leq \mathrm{MEV}_{\mathcal{A}}(S' + W'_{\Delta}, \mathcal{X})$.