DeFi composability as MEV non-interference

Massimo Bartoletti[1], Riccardo Marchesin[2], Roberto Zunino[2]

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

Abstract. Complex DeFi services are usually constructed by composing a variety of simpler smart contracts. The permissionless nature of the blockchains where these smart contracts are executed makes DeFi services exposed to security risks, since adversaries can target any of the underlying contracts to economically damage the compound service. We introduce a new notion of secure composability of smart contracts, which ensures that adversaries cannot economically harm the compound contract by interfering with its dependencies.

1 Introduction

Decentralized Finance (DeFi) is often touted as the "money Lego" for its ability to build complex financial services from a variety of simpler components [3,25]. Recent empirical analyses of the DeFi ecosystem show that DeFi protocols are heavily intertwined in practice, giving rise to complex interactions [19,20]. This complexity has a drawback, in that adversaries can exploit unintended forms of interaction among protocols to obtain an economic profit to the detriment of the users [14,18]. These security risks are further exacerbated by novel financial services allowing users to easily create arbitrary compositions of DeFi protocols [4]. The key unanswered question is: *when is a DeFi composition secure?*

Despite the clear practical relevance of this question, the research on DeFi composability is surprisingly limited. To the best of our knowledge, the only notion of secure DeFi composition in the scientific literature is the one introduced by Babel, Daian, Kelkar and Juels in their recent "Clockwork finance" paper [7]. There, the focus is on attacks where adversaries can exploit newly deployed contracts to increase their profit opportunities. Accordingly, their criterion is that contracts Δ are composable in a blockchain state S if adding Δ to S does not give the adversary a "significantly higher" Maximal Extractable Value (MEV). In formulae, denoting by $\mathrm{MEV}(S)$ the maximal value that adversaries can extract from S, and with $S \mid \Delta$ the blockchain state S extended with the contracts Δ, the composability criterion of Babel *et al.* is expressed as:

$$\Delta \text{ is } \varepsilon\text{-composable in } S \quad \text{iff} \quad \mathrm{MEV}(S \mid \Delta) \leq (1+\varepsilon)\ \mathrm{MEV}(S) \tag{1}$$

where ε parameterises the "not significantly higher" condition above. For example, let Δ be a contract allowing users to bet on the price of a token, relying on an Automated Market Maker (AMM) in S as a price oracle. If the adversary

has sufficient capital in S, (1) would correctly classify Δ as *not* 0-composable in S, because the adversary can produce enough price volatility in the AMM to always win the bet, and so to extract more MEV in $S \mid \Delta$ than in S.

Limitations of ε-composability We argue that the ε-composability of Babel *et al.* has some drawbacks. First, using the MEV of the *whole* state S as a baseline for the comparison makes it difficult to interpret the concrete security guarantee of ε-composability. For instance, one may be induced to believe that deploying a contract Δ in S is secure after finding that 0-composability holds. However, (1) only ensures that the MEV of $S \mid \Delta$ is bounded by that of S, while it does not say *from which contracts* this MEV is extracted. For instance, assume that the Total Locked Value (TLV) of Δ and the MEV of S are equal, and that extracting MEV from Δ blocks the MEV opportunities in S (as in Example 5). Even though Δ is 0-composable with S, attacking $S \mid \Delta$ can make Δ lose its *whole* TLV — not what the designer of Δ would reasonably consider secure.

Another drawback of using the *global* MEV as a baseline is that (1) classifies as *not* composable contracts that have intended MEV, even when they have no interference at all with the rest of the system. For instance, let Δ be an airdrop contract allowing anyone to withdraw its whole balance. Although Δ has no dependencies of any kind with the rest of the system, it is not 0-composable with any S, since $\mathrm{MEV}(S \mid \Delta)$ is greater than $\mathrm{MEV}(S)$. Trying to find values of ε for which ε-composability holds is impractical, and at the extreme, if $\mathrm{MEV}(S)$ is zero, then Δ is not ε-composable with S for any $\varepsilon \geq 0$.

Relying on global MEV also poses usability and algorithmic issues. First, a definition of composability that requires to compute a function of the *whole* blockchain state S (which is usually quite large) is hardly efficiently computable. Second, when ε-composability is violated because of additional MEV extracted from S, it is unclear which countermeasures could be taken against the attack, since the attacked contracts in S cannot be amended or removed.

A new security notion: MEV non-interference These arguments suggest to explore composability notions not relying on the global MEV. Our insight comes from *non-interference*, a notion studied by the information security community since the 1980s [17,23,22,8,16]. In the classic setting, non-interference requires that adversaries interacting with a software system cannot observe private data. More precisely, the property holds when public outputs (that can be observed by adversaries) are not affected by private (confidential) inputs. We adapt this notion to the DeFi setting, by requiring that the MEV extractable from Δ (the public output) is not affected by the dependencies of Δ (the private inputs). This means that adversaries cannot extract more value from Δ using *any* contract in $S \mid \Delta$ than they could extract by using Δ only: therefore, interacting with other contracts in the blockchain (including the dependencies of Δ) gives no advantage to the adversary. To the best of our knowledge, this is the first time non-interference principles are applied to the DeFi setting.

Our new security notion, that we dub *MEV non-interference*, relies on novel contributions to the theory of MEV. In particular, we introduce *local MEV*, a

Table 1: Summary of notation.

$\mathtt{A}, \mathtt{B}$	User accounts	$\mathcal{A}, \mathcal{B}$	Sets of [user|contract] accounts
$\mathtt{C}, \mathtt{D}$	Contract accounts	$\dagger\Gamma$	Contract accounts in Γ
$\mathcal{C}, \mathcal{D}$	Sets of contract accounts	$\mathit{deps}(\mathcal{C})$	Dependencies of $\mathcal{C}$
$\mathtt{T}, \mathtt{T}'$	Token types	$\$\mathbf{1}_{\mathtt{T}}$	Price of $\mathtt{T}$
$\mathtt{X}, \mathtt{X}'$	Transactions	$\vec{\mathtt{X}}$	Sequence of transactions
S, S'	Blockchain states	$\$_{\mathcal{A}}(S)$	Wealth of $\mathcal{A}$ in S
W, W'	Wallet states	$\gamma_{\mathcal{A}}(S, \vec{\mathtt{X}})$	$\mathcal{A}$'s gain upon firing $\vec{\mathtt{X}}$ in S
Γ, Δ	Contract states	$\mathfrak{M}$	Set of adversaries

new metric of economic attacks to smart contracts that measures the maximal economic loss that adversaries can cause to a *given* set of contracts (whereas global MEV only applies to the *whole* blockchain state). We study two versions of local MEV, which assume different adversary models: $\mathrm{MEV}(S, \mathcal{C})$ is the maximal loss of contracts $\mathcal{C}$ in S under adversaries whose wealth is *fixed* by S; instead, $\mathrm{MEV}^{\infty}(S, \mathcal{C})$ assumes adversaries with an *unbounded* capital to carry the attack (in practice, flash loans make this kind of adversary quite realistic). One of our main theoretical contributions is that computing $\mathrm{MEV}^{\infty}(S, \mathcal{C})$ just requires to know $\mathcal{C}$ and their dependencies (Theorem 1). This gives MEV^{∞} an algorithmic advantage over global MEV, which depends on the whole blockchain state.

We define MEV non-interference in two versions: $S \not\rightsquigarrow \Delta$ means that interacting with S does not give *bounded-wealth* adversaries more opportunities to cause a loss to the contracts Δ; furthermore, $\Gamma \not\rightsquigarrow^{\infty} \Delta$ does the same for *unbounded-wealth* adversaries, where Γ is the state S after the wallets have been removed.

We argue that MEV non-interference naturally captures the security property a developer would like to verify before deploying new contracts. By relying on the *local* MEV that can be extracted from Δ (instead of the global MEV), our notion overcomes the drawbacks of ε-composability. First, while contracts with intended MEV always break ε-composability, they possibly enjoy MEV non-interference: e.g., this is the case of the above-mentioned airdrop contract, which is MEV non-interfering w.r.t. any S. Second, for the unbounded-wealth version we prove that deciding $\Gamma \not\rightsquigarrow^{\infty} \Delta$ only requires to consider Δ and their dependencies in Γ (Theorem 4). We exploit this result to show that $\Gamma \not\rightsquigarrow^{\infty} \Delta$ is resistant to adversarial contracts: i.e., MEV non-interference still holds when an adversary deploys some contracts Δ' before Δ (by contrast, both ε-composability and $\not\rightsquigarrow$ are *not* resistant). We give sufficient conditions for both versions of MEV non-interference (Theorems 2 and 3), and we apply them to study typical compositions of DeFi protocols (Table 3).

Because of space constraints, we provide the proofs of all our results and the pseudo-code for our DeFi examples in a separated Appendix.

2 Blockchain model

We fix a model of account-based blockchains and smart contracts *à la* Ethereum. To keep our theory simple, we abstract from consensus features (e.g., the gas mechanism), and we rule out some problematic behaviours (e.g., reentrancy).

Blockchain states We assume a set $\mathbb{T}$ of *token types* ($\mathtt{T}, \mathtt{T}', \ldots$) and a countably infinite set $\mathbb{A}$ of *accounts*. Accounts are partitioned into *user accounts* $\mathtt{A}, \mathtt{B}, \ldots \in \mathbb{A}_u$ (representing the so-called *externally owned accounts* in Ethereum) and *contract accounts* $\mathtt{C}, \mathtt{D}, \ldots \in \mathbb{A}_c$. We designate a subset $\mathcal{M}$ of user accounts as adversaries[3], including e.g. block proposers and MEV searchers. The state of a user account is a map $w \in \mathbb{T} \to \mathbb{N}$ from tokens to non-negative integers, i.e. a ***wallet*** that quantifies the tokens in the account. The state of a contract account is a pair (w, σ), where w is a wallet and σ is a key-value store. ***Blockchain states*** $S, S', \ldots$ are finite maps from accounts to their states, where user wallets include at least $\mathcal{M}$'s. We use the operator $\mid$ to deconstruct a blockchain state into its components, writing e.g.:

$$S = \mathtt{A}[1{:}\,\mathtt{T}, 2{:}\,\mathtt{T}_2] \mid \mathtt{M}[0{:}\,\mathtt{T}] \mid \mathtt{C}[1{:}\,\mathtt{T}, \mathtt{owner} = \mathtt{A}]$$

for a blockchain state where the user account $\mathtt{A}$ stores 1 unit of token $\mathtt{T}$ and two units of token $\mathtt{T}_2$, the user account $\mathtt{M}$ has zero tokens, and the contract $\mathtt{C}$ stores 1 unit of $\mathtt{T}$ and has a key-value store mapping `owner` to $\mathtt{A}$.

Contracts We do not rely on a specific contract language: we just assume that contracts have an associated set of methods (like e.g., in Solidity).[4] A method can: (i) receive parameters and tokens from the caller, (ii) update the contract wallet and state, (iii) transfer tokens to user accounts, (iv) call other contracts (possibly transferring tokens along with the call), (v) return values and transfer tokens to the caller, (vi) abort. As usual, a method cannot drain tokens from other accounts: the only ways for a contract to receive tokens are (i) from a caller invoking one of its methods, or (ii) by calling a method of another contract that sends tokens to its caller. For simplicity, we assume that a contract $\mathtt{C}$ can only call methods of contracts deployed before it. Formally, defining "$\mathtt{C}$ *is called by* $\mathtt{D}$" when some method of $\mathtt{D}$ calls some method of $\mathtt{C}$, we are requiring that the transitive and reflexive closure $\sqsubseteq$ of this relation is a partial order. We also assume that blockchain states contain all the *dependencies* of their contracts: formally, if $\mathcal{C}$ are the contracts in S, we require that $\mathit{deps}(\mathcal{C}) = \{\mathtt{C}' \mid \exists \mathtt{C} \in \mathcal{C}.\ \mathtt{C}' \sqsubseteq \mathtt{C}\}$ are in S. States satisfying these assumptions are said *well-formed*: all states mentioned in our results (either in hypothesis or thesis) are always well-formed.

[3] In practice, given a blockchain state it would be safe to say that $\mathcal{M}$ are the accounts never mentioned in the contract states and code. Modelling $\mathcal{M}$ as a system parameter is a simplification, which avoids to make our definitions depend on a specific contract language. It would be possible to remove the parameter $\mathcal{M}$, at the cost of an increased complexity of the definitions and statements (see e.g. the adversarial MEV in [12]).

[4] For simplicity, we forbid direct transfer of assets between users: this is not a limitation, since these transfers can always be routed by suitable contracts.

Although well-formedness makes our model cleaner than Ethereum, preventing some problematic behaviours like reentrant calls [21], we only need it on the analysed contracts Δ and their dependencies (see Section 5). We write $S = W \mid \Gamma$ for a blockchain state S composed of user wallets W and contract states Γ. We can deconstruct wallets, writing $S = W \mid W' \mid \Gamma$ when $\operatorname{dom} W$ and $\operatorname{dom} W'$ are disjoint, as well as contract states, writing $S = W \mid \Gamma \mid \Delta$. We denote by $\dagger\Gamma$ the set of contract accounts in Γ (i.e. $\dagger\Gamma = \operatorname{dom}\Gamma$), and let $\mathit{deps}(\Delta) = \mathit{deps}(\dagger\Delta)$. Finally, we assume that contracts cannot inspect the state of other accounts, including users' wallets and the state of other contracts.[5] Formally, we are requiring that each transaction enabled in S produces the same effect in a "richer" state $S' \geq_\$ S$ containing more tokens in users' wallets (Definition B.2 in [11]).

Transactions We model contracts behaviour as a deterministic transition relation $\to$ between blockchain states, where state transitions are triggered by ***transactions*** $\mathsf{X}, \mathsf{X}', \ldots$. A transaction is a call to a contract method, written $\mathtt{A}\colon \mathtt{C}.\mathtt{f}(\mathtt{args})$, where $\mathtt{A}$ is the user signing the transaction, $\mathtt{C}$ is the called contract, $\mathtt{f}$ is the called method, and $\mathtt{args}$ is the list of actual parameters, which can also include transfers of tokens from $\mathtt{A}$ to $\mathtt{C}$. Invalid transactions are rolled-back, i.e. $\to$ preserves the state. There are various reasons for invalidity, e.g. the called method aborts, a token transfer without the needed tokens is attempted, etc. Given $\mathsf{X} = \mathtt{A}\colon \mathtt{C}.\mathtt{f}(\mathtt{args})$, we write $\mathit{callee}(\mathsf{X})$ for the target contract $\mathtt{C}$. Methods can refer to $\mathtt{A}$ via the identifier $\mathtt{origin}$ and to the caller (contract or user) account via $\mathtt{sender}$ (corresponding, resp., to $\mathtt{tx.origin}$ and $\mathtt{msg.sender}$ in Solidity).

Example: an Automated Market Maker In our examples, we specify contracts in pseudo-code. Its syntax is similar to Solidity, with some extra features: (i) the expression $\#\mathtt{T}$ denotes the number of tokens $\mathtt{T}$ stored in the contract; (ii) the formal parameter $?\,x\colon \mathtt{T}$ requires the $\mathtt{sender}$ to transfer some tokens $\mathtt{T}$ to the contract along with the call (the unsigned integer variable x generalises Solidity's $\mathtt{msg.value}$ to multi-tokens); (iii) the command $\mathtt{a}!\,e\colon \mathtt{T}$ transfers e units of $\mathtt{T}$ from the contract to account $\mathtt{a}$, where e is an expression, and $\mathtt{a}$ could be either a user account or the method $\mathtt{sender}$). We exemplify pseudo-code in Figure 1, specifying an Automated Market Maker inspired by Uniswap v2 [5,6,9,26]. Users can add liquidity, query the token pair and the exchange rate, and swap units of $\mathtt{T}_0$ with units of $\mathtt{T}_1$ or vice-versa. More specifically, $\mathtt{A}\colon \mathtt{swap}(?\,x\colon \mathtt{T}_0, y_{\min})$ allows $\mathtt{A}$ to send $x\colon \mathtt{T}_0$ to the contract, and receive at least $y_{\min}\colon \mathtt{T}_1$ in exchange. Symmetrically, $\mathtt{A}\colon \mathtt{swap}(?\,x\colon \mathtt{T}_1, y_{\min})$ allows $\mathtt{A}$ to exchange $x\colon \mathtt{T}_1$ for at least $y_{\min}\colon \mathtt{T}_0$.

Wealth and gain Measuring the effect of an attack requires to estimate the ***wealth*** of the adversary before and after the attack. We denote by $\$_{\mathcal{A}}(S)$ the wealth of accounts $\mathcal{A}$ in S. Such wealth is given by the weighted sum of the tokens in $\mathcal{A}$'s wallets, where the weights are the token prices. We denote by $\$\mathbf{1}_{\mathtt{T}}$

[5] These are not restrictions in practice. To make a contract depend on a user's wallet, we can require the users to transfer tokens along with contract calls. To make it depend on the state of other contracts, we can access it through getter methods.

```
contract AMM {
  addLiq(?x0:T0,?x1:T1) { // add liquidity to the AMM
    require #T0 * (#T1-x1) == (#T0-x0) * #T1 }
  getTokens() { return (T0,T1) }     // token pair
  getRate(t) {                       // exchange rate
    if (t==T0) return #T0/#T1        // r:T0 for 1:T1
    else if (t==T1) return #T1/#T0   // r:T1 for 1:T0
    else abort }
  swap(?x:t,ymin) {
    if (t==T0)
      { y=(x*#T1)/#T0; require ymin<=y<#T1; sender!y:T1 }
    else if (t==T1)
      { y=(x*#T0)/#T1; require ymin<=y<#T0; sender!y:T0 }
    else abort }
}
```

Fig. 1: A constant-product AMM contract.

the (strictly positive) price of token type $\mathtt{T}$. This implicitly assumes that token prices are constant, since they do not depend on the blockchain state.[6]

Definition 1 (Wealth). *The wealth of $\mathcal{A} \subseteq \mathbb{A}$ in $S = W \mid \Gamma$ is given by:*

$$\$_{\mathcal{A}}(S) \;=\; \sum_{\mathtt{A} \in \mathcal{A} \cap \mathrm{dom}\, W,\, \mathtt{T}} W(\mathtt{A})(\mathtt{T}) \cdot \$\mathbf{1}_{\mathtt{T}} \;+\; \sum_{\mathtt{C} \in \mathcal{A} \cap \mathrm{dom}\, \Gamma,\, \mathtt{T}} \mathit{fst}(\Gamma(\mathtt{C}))(\mathtt{T}) \cdot \$\mathbf{1}_{\mathtt{T}} \qquad (2)$$

To rule out ill-formed states with an infinite amount of tokens, we require blockchain states to enjoy the *finite tokens axiom*, i.e. $\sum_{\mathtt{A},\mathtt{T}} S(\mathtt{A})(\mathtt{T}) \in \mathbb{N}$. This makes the global wealth always finite. The success of attacks is measured as *gain*, i.e. the difference of the attackers' wealth before and after the attack.

Definition 2 (Gain). *The gain of $\mathcal{A} \subseteq \mathbb{A}$ upon firing a transactions sequence $\vec{\mathtt{x}}$ in S is given by $\gamma_{\mathcal{A}}(S, \vec{\mathtt{x}}) = \$_{\mathcal{A}}(S') - \$_{\mathcal{A}}(S)$ if $S \xrightarrow{\vec{\mathtt{x}}} S'$.*

3 Local MEV

The MEV in a state S is the maximum gain of the adversary $\mathcal{M}$ upon firing a sequence of transactions constructed by $\mathcal{M}$ [7,12]. If we denote by $\kappa(\mathcal{M}, P)$ the set of transactions craftable by $\mathcal{M}$ using a mempool P, and by $\kappa(\mathcal{M}, P)^*$ their finite sequences, the MEV in a blockchain state S can be formalised as:

$$\mathrm{MEV}(S, P) = \max \left\{ \gamma_{\mathcal{M}}(S, \vec{\mathtt{x}}) \;\middle|\; \vec{\mathtt{x}} \in \kappa(\mathcal{M}, P)^* \right\} \qquad (3)$$

[6] This simplifying assumption allows local MEV to neglect the parts of the state that could affect token prices. A more realistic handling of token prices would require to extend the model with a function that determines the token prices in a given state.

This notion measures the value that adversaries can extract from *any* contract in the blockchain and transfer to their wallets. To study composability, following the intuitions in Section 1 we need instead to check if the contracts that will be deployed can have a loss when adversaries manipulate their dependencies. Therefore, our notion of MEV diverges from (3) in four aspects:

1. We measure the value that adversaries can extract from a *given* set of contracts $\mathcal{C}$. This means that only the tokens extracted from the contracts in $\mathcal{C}$ contribute to the extractable value, while the tokens grabbed from other contracts do not count (while they would count for the *global* MEV in (3)).
2. We count as MEV *all* the tokens that $\mathcal{M}$ can remove from $\mathcal{C}$, regardless of whether $\mathcal{M}$ can transfer them to their wallets. Namely, while (3) maximises $\mathcal{M}$'s gain $\gamma_{\mathcal{M}}$, our notion maximises $\mathcal{C}$'s *loss* $-\gamma_{\mathcal{C}}$. This is because we want our composability to provide security also against *irrational adversaries*, who try to damage contracts without necessarily making a profit.
3. We parameterise our MEV w.r.t. the set $\mathcal{D}$ of contracts callable by $\mathcal{M}$. This allows us to rework the distinction between private and public information in language-based non-interference. There, the idea is that public outputs are not affected by private inputs, i.e. restricting inputs to the private ones must preserve the public outputs. In our context, we rephrase this by requiring that the MEV extractable from $\mathcal{C}$ (the public output) is not affected when restricting the set of callable contracts to a subset $\mathcal{D}$ (the private inputs).
4. We assume that the mempool is empty, just writing $\kappa(\mathcal{M})$. We do so because to define secure composability we are concerned about the MEV extractable by exploiting new contracts, and not that extractable from the mempool. Note that the mempool is instead considered in the MEV notions in [7,12].

We call this new notion ***local MEV***, and we denote it by $\mathrm{MEV}_{\mathcal{D}}(S, \mathcal{C})$.

Definition 3 (Local MEV). *Let* $\kappa_{\mathcal{D}}(\mathcal{M}) = \{\mathsf{X} \in \kappa(\mathcal{M}) \mid \mathit{callee}(\mathsf{X}) \subseteq \mathcal{D}\}$ *be the set of transactions craftable by* $\mathcal{M}$ *and targeting contracts in* $\mathcal{D}$*. We define:*

$$\mathrm{MEV}_{\mathcal{D}}(S, \mathcal{C}) = \max\left\{-\gamma_{\mathcal{C}}(S, \vec{\mathsf{X}}) \;\middle|\; \vec{\mathsf{X}} \in \kappa_{\mathcal{D}}(\mathcal{M})^*\right\} \tag{4}$$

Hereafter, we abbreviate $\mathrm{MEV}_{\mathbb{A}_c}(S, \mathcal{C})$ *as* $\mathrm{MEV}(S, \mathcal{C})$.

Example 1. Consider two instances of the AMM contract in Figure 1, to swap respectively the pairs $(\mathtt{T}_0, \mathtt{T}_1)$ and $(\mathtt{T}_1, \mathtt{T}_2)$. Let $\$\mathbf{1}_{\mathtt{T}_0} = \$\mathbf{1}_{\mathtt{T}_1} = \$\mathbf{1}_{\mathtt{T}_2} = 1$, and let $S = \mathtt{M}[3{:}\,\mathtt{T}_0] \mid \mathtt{AMM1}[6{:}\,\mathtt{T}_0, 6{:}\,\mathtt{T}_1] \mid \mathtt{AMM2}[4{:}\,\mathtt{T}_1, 9{:}\,\mathtt{T}_2]$, where $\mathtt{M}$ is the adversary. We want to compute the local MEV w.r.t. $\mathcal{C} = \{\mathtt{AMM2}\}$. Recall that we are assuming the mempool to be empty. In the unrestricted case (i.e., $\mathcal{D}$ is the universe), the trace $\vec{\mathsf{X}}$ that maximises the loss of $\mathtt{AMM2}$ is the following:

$$\begin{aligned} S &\xrightarrow{\mathtt{M}:\mathtt{AMM1.swap}(?\,3:\mathtt{T}_0,0)} \mathtt{M}[2{:}\,\mathtt{T}_1] \mid \mathtt{AMM1}[9{:}\,\mathtt{T}_0, 4{:}\,\mathtt{T}_1] \mid \mathtt{AMM2}[4{:}\,\mathtt{T}_1, 9{:}\,\mathtt{T}_2] \\ &\xrightarrow{\mathtt{M}:\mathtt{AMM2.swap}(?\,2:\mathtt{T}_1,0)} \mathtt{M}[3{:}\,\mathtt{T}_2] \mid \mathtt{AMM1}[9{:}\,\mathtt{T}_0, 4{:}\,\mathtt{T}_1] \mid \mathtt{AMM2}[6{:}\,\mathtt{T}_1, 6{:}\,\mathtt{T}_2] \end{aligned}$$

We have that $-\gamma_{\{\texttt{AMM2}\}}(S, \vec{\chi}) = 1$, hence $\mathrm{MEV}(S, \{\texttt{AMM2}\}) = 1$. Instead, when $\texttt{M}$ is restricted to use $\mathcal{D} = \{\texttt{AMM2}\}$, it has no way to obtain the tokens $\texttt{T}_1$ that are needed to extract value from $\texttt{AMM2}$: therefore, $\mathrm{MEV}_{\{\texttt{AMM2}\}}(S, \{\texttt{AMM2}\}) = 0$. $\diamond$

Lemma 1 establishes some useful properties of local MEV. Item 1 studies some border cases: in particular, when the set of observed contracts $\mathcal{C}$ is the universe, local MEV over-approximates global MEV. Items 2 and 3 state that widening the restricted contracts $\mathcal{D}$ or the contract state potentially increases the local MEV[7]. Item 4 allows to restrict the set of contract accounts $\mathcal{C}$ and $\mathcal{D}$ to those occurring in the blockchain state Γ (i.e., $\dagger\Gamma$). Item 5 states that the local MEV extractable from $\mathcal{C}$ is non-negative and bounded by the wealth of $\mathcal{C}$.

Lemma 1 (Basic properties of MEV). *For all S, $\mathcal{C}, \mathcal{D} \subseteq \mathbb{A}_c$:*

1. $\mathrm{MEV}_{\mathcal{D}}(S, \emptyset) = \mathrm{MEV}_{\emptyset}(S, \mathcal{C}) = 0$, $\mathrm{MEV}_{\mathbb{A}_c}(S, \mathbb{A}_c) \geq \mathrm{MEV}(S)$
2. *if* $\mathcal{D} \subseteq \mathcal{D}'$, *then* $\mathrm{MEV}_{\mathcal{D}}(S, \mathcal{C}) \leq \mathrm{MEV}_{\mathcal{D}'}(S, \mathcal{C})$
3. $\mathrm{MEV}_{\mathcal{D}}(W \mid \Gamma, \mathcal{C}) \leq \mathrm{MEV}_{\mathcal{D}}(W \mid \Gamma', \mathcal{C})$, *where* $\Gamma' \,|_{\mathrm{dom}\,\Gamma} = \Gamma$
4. $\mathrm{MEV}_{\mathcal{D}}(W \mid \Gamma, \mathcal{C}) = \mathrm{MEV}_{\mathcal{D}}(W \mid \Gamma, \mathcal{C} \cap \dagger\Gamma) = \mathrm{MEV}_{\mathcal{D} \cap \dagger\Gamma}(W \mid \Gamma, \mathcal{C})$
5. $0 \leq \mathrm{MEV}_{\mathcal{D}}(S, \mathcal{C}) \leq \$_{\mathcal{C}}(S)$

Lemma 2 states that the only user wallets that need to be taken into account to estimate the MEV are those of the adversary (Item 1). This is because $\mathcal{M}$ has no way to force other users to spend their tokens in the attack sequence.[8] Furthermore, wealthier adversaries may potentially extract more MEV (Item 2).

Lemma 2 (MEV and adversaries' wallets).

1. *if* $\mathrm{dom}\, W_{\mathcal{M}} = \mathcal{M}$, *then* $\mathrm{MEV}_{\mathcal{D}}(W_{\mathcal{M}} \mid W \mid \Gamma, \mathcal{C}) = \mathrm{MEV}_{\mathcal{D}}(W_{\mathcal{M}} \mid \Gamma, \mathcal{C})$
2. *if* $S \leq_{\$} S'$, *then* $\mathrm{MEV}_{\mathcal{D}}(S, \mathcal{C}) \leq \mathrm{MEV}_{\mathcal{D}}(S', \mathcal{C})$

Since the overall amount of tokens held in contract wallets is limited, the MEV no longer increases when the adversary is rich enough (Lemma 3). Formally, we prove that there exists a threshold adversary wallet $W_{\mathcal{M}}$ yielding the same MEV as any richer adversary wallet. Together with item 2 of Lemma 2, this ensures $W_{\mathcal{M}}$ yields the maximum MEV over any adversary wallet.

Lemma 3 (Stability). *For all $\mathcal{C}$, $\mathcal{D}$, Γ, there exists an adversary wallet $W_{\mathcal{M}}$ such that* $\mathrm{MEV}_{\mathcal{D}}(W_{\mathcal{M}} \mid \Gamma, \mathcal{C}) = \mathrm{MEV}_{\mathcal{D}}(W'_{\mathcal{M}} \mid \Gamma, \mathcal{C})$ *for all* $W'_{\mathcal{M}} \geq_{\$} W_{\mathcal{M}}$.

Taking the maximum MEV over all possible user wallets (which always exists by Lemma 3) we then introduce a notion of MEV that does *not* depend on user wallets at all (including $\mathcal{M}$'s). The new notion, dubbed MEV^{∞}, reflects the fact

[7] Instead, $\mathcal{C} \subseteq \mathcal{C}'$ does not imply $\mathrm{MEV}_{\mathcal{D}}(S, \mathcal{C}) \leq \mathrm{MEV}_{\mathcal{D}}(S, \mathcal{C}')$, because maximising the loss of a contract may increase the gain of another one (see Example B.1 in [11]).

[8] This property would not hold if the adversary could play user transactions in the mempool, since their validity depends on the wallets of the users who signed them. However, when studying contract composability the question is whether a new contract can be attacked by exploiting its dependencies, so the mempool is irrelevant.

that the capital required from the adversary is not a barrier in practice. E.g., as already noted in [7], the adversary can apply for a flash loan (a risk-free operation) to obtain the capital needed to extract the full MEV.[9] Consequently, MEV^∞ is the most robust estimation of the value extractable by the adversary.

Definition 4 (Local MEV of wealthy adversaries). *For all* $\mathcal{C}, \mathcal{D}, \Gamma$*, let:*

$$\mathrm{MEV}^\infty_{\mathcal{D}}(\Gamma, \mathcal{C}) = \max_{W} \mathrm{MEV}_{\mathcal{D}}(W \mid \Gamma, \mathcal{C}) \tag{5}$$

Besides satisfying the basic properties of local MEV seen before (Lemma B.1 in [11]), MEV^∞ enjoys a key property: to compute the MEV^∞ of $\mathcal{C}$ in Γ, we can ignore all the contracts in Γ except the dependencies of $\mathcal{C}$. This is not true for MEV, because non-wealthy adversaries may need to interact with other contracts to obtain the tokens needed to extract value from $\mathcal{C}$. Formally, we define the *stripping* of Γ w.r.t. $\mathcal{C}$ (in symbols, $\Gamma \restriction_{\mathcal{C}}$) as the restriction of Γ to the domain $\mathit{deps}(\mathcal{C})$. Note that when Γ is well-formed, then also $\Gamma \restriction_{\mathcal{C}}$ is well-formed. Theorem 1 gives sufficient conditions under which we can strip from Γ all the non-dependencies of $\mathcal{C}$, preserving $\mathrm{MEV}^\infty_{\mathcal{D}}(\Gamma, \mathcal{C})$. The first condition is that contract methods are not aware of the identity of the `sender`, being only able to use it as a recipient of token transfers: we refer to this by saying that contracts are *sender-agnostic* (see Def. B.4 in [11]). The second condition ensures that $\mathcal{D}$ contains enough contracts to reproduce attacks in the stripped state.

Theorem 1 (State stripping). $\mathrm{MEV}^\infty_{\mathcal{D}}(\Gamma, \mathcal{C}) = \mathrm{MEV}^\infty_{\mathcal{D}}(\Gamma \restriction_{\mathcal{C}}, \mathcal{C})$ *holds if the contracts* $\mathcal{C}' = \mathit{deps}(\mathcal{C}) \cap \mathit{deps}(\mathcal{D} \setminus \mathit{deps}(\mathcal{C}))$ *satisfy: (i)* $\mathcal{C}'$ *are sender-agnostic, and (ii)* $\mathcal{C}' \subseteq \mathcal{D}$*. In particular, (ii) holds if* $\mathcal{D} = \mathbb{A}_c$ *or* $\mathcal{D} = \mathcal{C}$*.*

If any of the conditions (i) or (ii) do not hold, then adversaries might not be able to perform an attack on $\Gamma \restriction_{\mathcal{C}}$ with the same effect as an attack on Γ, and so they might extract less MEV^∞ in $\Gamma \restriction_{\mathcal{C}}$ than in Γ (see Ex. B.2, B.3 in [11]).

The following corollary of Theorem 1 states another key property of MEV^∞: extending the state with new contracts does not affect the MEV extractable from the old contracts.[10] This is the basis to prove that $\not\rightsquigarrow^\infty$ is resistant to adversarial contracts (Corollary 2).

Corollary 1. $\mathrm{MEV}^\infty(\Gamma, \mathcal{C}) = \mathrm{MEV}^\infty(\Gamma \mid \Delta, \mathcal{C})$ *for all* $\mathcal{C} \subseteq \dagger\Gamma$*.*

Corollary 1 highlights a drawback of the ε-composability in [7], which compares the global MEV in S with that in $S \mid \Delta$, where new contracts Δ have been deployed. Corollary 1 states that for wealthy adversaries, the two MEVs may only differ in the value extractable from Δ, and so Δ is 0-composable with S only if Δ has zero MEV. Hence, contracts that have intended MEV are not composable for [7], even if they have no interactions at all with the context.

[9] This is true for *atomic* attacks, where the adversary extracts MEV through a single transaction. If extracting MEV requires multiple transactions, like e.g. in sandwich attacks exploiting transactions in the mempool, flash loans are not possible.

[10] This property relies on assumptions (e.g., non-circularity of contract dependencies) that hold in our blockchain model, but may not hold in some concrete platforms. For instance, in Ethereum one can craft contracts that exploit reentrancy to extract MEV from already existing contracts, as in the infamous DAO attack [1].

```
contract Airdrop {
  constructor(?x:t) { tout=t }      // deposit any token t
  withdraw() { sender!#tout:tout } // any user withdraws
}
contract Exchange {
  constructor(?x:t1,t2,r) {
    require r>0; rate=r; tout=t1; tin=t2; owner=origin }
  getTokens() { return (tin,tout) }
  getRate() { return rate }
  setRate(newRate) { require origin==owner; rate=newRate }
  swap(?x:t) {                  // receives x units of tin
    require t==tin && #tout>=x*rate;
    sender!x*rate:tout }        // sends x*rate units of tout
}
```

Fig. 2: An airdrop and an exchange contract.

4 MEV non-interference

To formalise contract composability, we start by defining a relation $S \not\rightsquigarrow \Delta$ between blockchain states $S = W \mid \Gamma$ and contract states Δ. Intuitively, $S \not\rightsquigarrow \Delta$ means that the adversary cannot leverage S to extract more MEV from Δ than it would be possible by interacting with Δ alone. We will then say that S is *MEV non-interfering* with Δ. Note that $S \not\rightsquigarrow \Delta$ may or may not hold depending on the wealth of the adversary in S, as we have already observed in Lemma 2 that MEV depends on the adversary's wallets. As noted before, assuming bounds on the capital available to the adversary may be unsafe, and consequently we introduced in Definition 4 a notion of MEV that does not make assumptions on the adversary's wealth. Based on this notion, we will study later in this section another relation $\Gamma \not\rightsquigarrow^{\infty} \Delta$, which holds when contracts Γ do not interfere with the MEV extractable from Δ *regardless* of the adversary's wealth.

Formally, MEV non-interference $S \not\rightsquigarrow \Delta$ holds when the MEV extractable from the contract accounts in Δ (i.e., $\dagger\Delta$) using *any* contract in $S \mid \Delta$ is exactly the same MEV that can be extracted using *only* the contracts in Δ. We write $S \rightsquigarrow \Delta$ when $S \not\rightsquigarrow \Delta$ does not hold.[11]

Definition 5 (MEV non-interference). *A state S is* MEV non-interfering *with Δ, in symbols $S \not\rightsquigarrow \Delta$, when* $\mathrm{MEV}(S \mid \Delta, \dagger\Delta) = \mathrm{MEV}_{\dagger\Delta}(S \mid \Delta, \dagger\Delta)$.[12]

The following example discriminates MEV non-interference from Babel *et al.*'s ε-composability, showing that a contract with intended MEV but no interactions with the context enjoys MEV non-interference, but it is not ε-composable.

[11] Note that these definitions only apply when $S \mid \Delta$ is a well-formed blockchain state.
[12] Note that $\geq$ always holds by Lemma 1, so the definition only requires to check $\leq$.

```
contract Bet_oracle {
  constructor(?x:ETH,t,r,d) {
    require t!=ETH && oracle.getTokens()==(ETH,t);
    tok=t; rate=r; owner=origin; deadline=d }
  bet(?x:ETH) {
    require player==null && x==#ETH;
    player=origin }
  win() {
    require block.num<=deadline && origin==player;
    require oracle.getRate(ETH)>rate;
    player!#ETH:ETH }
  close() {
    require block.num>deadline && origin==owner;
    owner!#ETH:ETH }
}
```

Fig. 3: A bet contract relying on an external price oracle.

Example 2. Consider the airdrop contract in Figure 2, let S be any blockchain state, and let $\Delta = \texttt{Airdrop}[n{:}\,\texttt{T}, \texttt{tout} = \texttt{T}]$. Babel *et al.*'s ε-composability requires $\mathrm{MEV}(S \mid \Delta) \leq (1+\varepsilon)\mathrm{MEV}(S)$. In $S \mid \Delta$, the adversary can extract $n{:}\,\texttt{T}$ from Airdrop, and possibly use these tokens to extract more MEV from S. Therefore, $\mathrm{MEV}(S \mid \Delta) \geq n \cdot \$\mathbf{1}_{\texttt{T}} + \mathrm{MEV}(S)$, meaning that, for large enough n, the airdrop is *not* ε-composable with S. By contrast, $\mathrm{MEV}(S \mid \Delta, \{\texttt{Airdrop}\}) = n \cdot \$\mathbf{1}_{\texttt{T}} = \mathrm{MEV}_{\{\texttt{Airdrop}\}}(S \mid \Delta, \{\texttt{Airdrop}\})$, and so $S \not\rightsquigarrow \Delta$. This correctly reflects the fact that using the contracts in S does not give $\mathcal{M}$ any way to damage the airdrop. $\diamond$

We now discuss contract conditions that may break MEV non-interference. Clearly, *contract dependencies* (i.e., a contract in Δ that calls a contract in S) are a possible cause of MEV interferences $S \rightsquigarrow \Delta$. Example 3 shows the issue through a classical DeFi composition, where a bet contract uses an AMM as a price oracle [7]. Another source of MEV interference is when the contracts in S and Δ have *token dependencies* (i.e., a contract in Δ outputs tokens which can be used as input to contract in S, or vice-versa). Example 4 shows that this can cause MEV interferences, even when S and Δ have no contract dependencies.

Example 3. The Bet contract in Figure 3 allows anyone to bet on the exchange rate between a token and ETH. The contract is parameterised over an oracle contract that is queried for the token price. Bet receives the initial pot from the owner upon deployment. To join, a player must pay an amount of ETH equal to the pot. Before the deadline, the pot can be withdrawn by the player if the oracle exchange rate is greater than the bet rate. After the deadline, the pot goes to the owner. Consider an instance of Bet using an AMM as price oracle, and let:

$$S = \texttt{M}[310{:}\,\texttt{ETH}] \mid \texttt{AMM}[600{:}\,\texttt{ETH}, 600{:}\,\texttt{T}] \mid \texttt{block.num} = n - k \mid \cdots$$
$$\Delta = \texttt{Bet}[10{:}\,\texttt{ETH}, \texttt{tok} = \texttt{T}, \texttt{rate} = 3, \texttt{owner} = \texttt{A}, \texttt{deadline} = n]$$

Note that the current oracle exchange rate is 1, while winning the bet requires to make it exceed 3. Since the deadline has not passed in S ($\texttt{block.num} = n-k < n$) and the adversary $\texttt{M}$ is rich enough, she can fire the following sequence:

$$
\begin{aligned}
S \mid \Delta \xrightarrow{\texttt{M:Bet.bet}(?\,10{:}\texttt{ETH})} & \quad \texttt{M}[300{:}\,\texttt{ETH}] \mid \texttt{AMM}[600{:}\,\texttt{ETH}, 600{:}\,\texttt{T}] \mid \texttt{Bet}[20{:}\,\texttt{ETH}, \cdots] \\
\xrightarrow{\texttt{M:AMM.swap}(?\,300{:}\texttt{ETH},0)} & \quad \texttt{M}[200{:}\,\texttt{T}] \mid \texttt{AMM}[900{:}\,\texttt{ETH}, 400{:}\,\texttt{T}] \mid \texttt{Bet}[20{:}\,\texttt{ETH}, \cdots] \\
\xrightarrow{\texttt{M:Bet.win}()} & \quad \texttt{M}[20{:}\,\texttt{ETH}, 200{:}\,\texttt{T}] \mid \texttt{AMM}[900{:}\,\texttt{ETH}, 400{:}\,\texttt{T}] \mid \texttt{Bet}[0{:}\,\texttt{ETH}, \cdots] \\
\xrightarrow{\texttt{M:AMM.swap}(?\,200{:}\texttt{T},0)} & \quad \texttt{M}[320{:}\,\texttt{ETH}] \mid \texttt{AMM}[600{:}\,\texttt{ETH}, 600{:}\,\texttt{T}] \mid \texttt{Bet}[0{:}\,\texttt{ETH}, \cdots]
\end{aligned}
$$

When $\texttt{M}$ can interact with $\texttt{Bet}$'s dependency $\texttt{AMM}$, the loss of $\texttt{Bet}$ is $10{:}\,\texttt{ETH}$, hence $\text{MEV}(S \mid \Delta, \{\texttt{Bet}\}) = 10 \cdot \$\mathbf{1}_{\texttt{ETH}}$. Instead, when $\texttt{M}$ can only use $\texttt{Bet}$, $\text{MEV}_{\{\texttt{Bet}\}}(S \mid \Delta, \{\texttt{Bet}\}) = 0$. Therefore, S is MEV interfering with the $\texttt{Bet}$ contract. Note that a poorer $\texttt{M}$ may not have enough $\texttt{ETH}$ to produce the short-term volatility in the oracle exchange rate. Later in Definition 6 we will introduce a notion of MEV non-interference that does not depend on $\texttt{M}$'s wealth. ◇

Example 4. To show that MEV interference can happen even in the absence of contract dependencies, consider $\texttt{Airdrop}$ and $\texttt{Exchange}$ in Figure 2, and let:

$$
\begin{aligned}
S &= \texttt{M}[0{:}\,\texttt{T}] \mid \texttt{Airdrop}[1{:}\,\texttt{T}, \texttt{tout} = \texttt{T}] \\
\Delta &= \texttt{Exchange}[10{:}\,\texttt{ETH}, \texttt{tin} = \texttt{T}, \texttt{tout} = \texttt{ETH}, \texttt{rate} = 10, \texttt{owner} = \texttt{B}]
\end{aligned}
$$

The unrestricted MEV of $\texttt{Exchange}$ is $10 \cdot \$\mathbf{1}_{\texttt{ETH}}$, since $\texttt{M}$ can first extract $1{:}\,\texttt{T}$ from the airdrop, and then use the exchange, draining $10 \cdot \$\mathbf{1}_{\texttt{ETH}}$. Instead, its restricted MEV is zero, since $\texttt{M}$ cannot obtain the needed $1{:}\,\texttt{T}$. Hence, $S \rightsquigarrow \Delta$. ◇

Theorem 2 devises sufficient conditions for MEV non-interference. Condition (1) states that $S \not\rightsquigarrow \Delta$ holds whenever the new contracts Δ have zero MEV: a special case is when Δ have no tokens, by Lemma 1(5). Condition (2) requires contract and token independence. Formally, Γ and Δ are *contract independent* when their dependencies are disjoint, i.e. $\mathit{deps}(\Gamma) \cap \mathit{deps}(\Delta) = \emptyset$, and they are *token independent* in S when the token types that can be received by Γ in S are disjoint from those that can be sent by Δ in S, and vice-versa (see Definition B.5 in [11].) For instance, in Example 4 the $\texttt{Airdrop}$ has no input tokens, and it has $\texttt{T}$ as output token, while $\texttt{Exchange}$ has $\texttt{T}$ as input token and $\texttt{ETH}$ as output token. Since $\texttt{T}$ is both in the outputs of $\texttt{Airdrop}$ and in the inputs of $\texttt{Exchange}$, the two contracts are *not* token independent. Condition (3) relaxes condition (2), allowing contract dependence provided that the dependencies of Δ occurring in Γ cannot be exploited by adversaries. Formally, we require that Δ is *stable w.r.t. moves of* $\mathcal{M}$ *on* Γ, i.e. any transaction craftable by $\mathcal{M}$ and targeting a contract in Γ does not affect the observable behaviour of methods of Γ called from Δ. The observable behaviour includes the returned values, the transferred tokens and the aborts of methods that can be called from Δ.

Theorem 2 (Sufficient conditions for $\not\rightsquigarrow$). *Let $S = W \mid \Gamma$. Each of the following conditions implies $S \not\rightsquigarrow \Delta$: (1)* $\text{MEV}(S \mid \Delta, \dagger\Delta) = 0$ *(2)* *Γ and Δ*

are token independent in $S \mid \Delta$ *and contract independent (3)* Γ *and* Δ *are token independent in* $S \mid \Delta$ *and* Δ *is stable w.r.t. moves of* $\mathcal{M}$ *on* Γ.

Note that neither contract independence nor token independence nor stability are necessary conditions for $\not\rightsquigarrow$. For instance, a contract that provides the best swap between two AMMs (see Figure A.1 in [11]) has both contract dependencies (on the two called AMMs) and token dependencies (their underlying tokens), but nonetheless it enjoys MEV non-interference, because it has zero MEV (indeed, the contract balance is always zero, so there is nothing to extract)[13].

MEV non-interference against wealthy adversaries In general, $S \not\rightsquigarrow \Delta$ does not imply $S \mid \Gamma \not\rightsquigarrow \Delta$, even if Γ and Δ are contract independent. This means that the notion studied so far (similarly to ε-composability of [7]) gives no security guarantees against attacks where adversaries manage to deploy contracts Γ *before* Δ. For instance, consider the contract $\Gamma = \texttt{Airdrop}[1{:}\,\texttt{T}, \texttt{tout} = \texttt{T}]$ and the contract $\Delta = \texttt{Bet}_{\texttt{AMM}}[10{:}\,\texttt{ETH}, \texttt{tok} = \texttt{T}, \texttt{rate} = 3, \texttt{owner} = \texttt{A}, \texttt{deadline} = n]$, and let S be empty. Since $\texttt{M}$ has 0 tokens in S, she cannot extract MEV from Δ, hence $S \not\rightsquigarrow \Delta$ by condition (1) of Theorem 2. Instead, $S \mid \Gamma \rightsquigarrow \Delta$, as shown in Example 4. This highlights a usability limitation of $\not\rightsquigarrow$: assume that a user detects that $S \not\rightsquigarrow \Delta$, and then sends a transaction to deploy the contracts Δ. If this transaction is front-run with another transaction that deploys Γ, then the new state $S \mid \Gamma$ could violate MEV non-interference with Δ. To overcome this limitation, Definition 6 provides a notion of MEV non-interference that is robust w.r.t. the adversary's wealth and enjoys resistance to adversarial contracts.

Definition 6 (MEV non-interference against wealthy adversaries). *A contract state* Γ *is* MEV^∞ non-interfering *with* Δ*, in symbols* $\Gamma \not\rightsquigarrow^\infty \Delta$*, when*

$$\text{MEV}^\infty(\Gamma \mid \Delta, \dagger\Delta) = \text{MEV}^\infty_{\dagger\Delta}(\Gamma \mid \Delta, \dagger\Delta)$$

Lemma 4 characterizes $\not\rightsquigarrow^\infty$ in terms of $\not\rightsquigarrow$: intuitively, $\Gamma \not\rightsquigarrow^\infty \Delta$ holds whenever $W \mid \Gamma \not\rightsquigarrow \Delta$ holds for rich enough adversaries' wallets W.

Lemma 4 ($\not\rightsquigarrow^\infty$ vs. $\not\rightsquigarrow$). $\Gamma \not\rightsquigarrow^\infty \Delta$ *if and only if* $\exists W_0.\ \forall W \geq_\$ W_0.\ W \mid \Gamma \not\rightsquigarrow \Delta$

The following theorem refines Theorem 2 giving sufficient conditions for $\not\rightsquigarrow^\infty$. Note that token independence is no longer required.

Theorem 3 (Sufficient conditions for $\not\rightsquigarrow^\infty$). *Each of the following conditions implies* $\Gamma \not\rightsquigarrow^\infty \Delta$*: (1)* $\text{MEV}^\infty(\Gamma \mid \Delta, \dagger\Delta) = 0$*; (2)* Γ *and* Δ *are contract independent; (3)* Δ *is stable w.r.t. moves of* $\mathcal{M}$ *on* Γ.

[13] By contrast, zero-MEV is *not* a sufficient condition for Babel *et al.*' ε-composability: e.g., a contract may not be 0-composable in S when $\mathcal{M}$ does not have the tokens needed to extract MEV in S, but becomes able to do so after exchanging tokens between S and Δ, provided that this preserves Δ's wealth (see Example B.4 in [11]).

Table 2: Structural properties of $\not\rightarrow^\infty$. Starred properties additionally require that some contracts are sender-agnostic. Ref. lemmas/examples are in [11].

Hypothesis	Thesis	Valid	Ref.
$\Gamma \not\rightarrow^\infty \Delta$	$\Gamma \mid \Gamma' \not\rightarrow^\infty \Delta$	✓⋆	Cor. 2
	$\Gamma' \mid \Gamma \not\rightarrow^\infty \Delta$	✓	Lem. B.2
	$\Gamma \not\rightarrow^\infty \Delta \mid \Delta'$	✗	Ex. B.6
	$\Gamma \not\rightarrow^\infty \Delta' \mid \Delta$	✗	Ex. B.6
$\Gamma \mid \Gamma' \not\rightarrow^\infty \Delta$	$\Gamma \not\rightarrow^\infty \Delta$	✓	Lem. B.3
$\Gamma' \mid \Gamma \not\rightarrow^\infty \Delta$		✓	Lem. B.3
$\Gamma \not\rightarrow^\infty \Delta \mid \Delta'$		✗	Ex. B.7
$\Gamma \not\rightarrow^\infty \Delta' \mid \Delta$		✗	Ex. B.7
$\Gamma \not\rightarrow^\infty \Delta_1 \wedge \Gamma \not\rightarrow^\infty \Delta_2$	$\Gamma \not\rightarrow^\infty \Delta_1 \mid \Delta_2$	✗	Ex. B.8
$\Gamma \not\rightarrow^\infty \Delta_1 \wedge \Gamma \mid \Delta_1 \not\rightarrow^\infty \Delta_2$		✗	Ex. B.8
$\Gamma \not\rightarrow^\infty \Delta_1 \wedge \mathrm{MEV}^\infty(\Gamma \mid \Delta_1 \mid \Delta_2, \dagger\Delta_2) = 0$		✓⋆	Lem. B.4

Theorem 4 establishes a key property of $\not\rightarrow^\infty$: namely, $\Gamma \not\rightarrow^\infty \Delta$ is preserved when removing from Γ all the contracts except the dependencies of Δ. Formally. the statement filters Γ with the state stripping operator $\restriction_{\dagger\Delta}$, which preserves exactly the dependencies of Δ. This highlights the algorithmic advantage of $\not\rightarrow^\infty$ w.r.t. ε-composability, which requires to consider the whole blockchain state.

Theorem 4 (State stripping). *$\Gamma \not\rightarrow^\infty \Delta$ if and only if $\Gamma \restriction_{\dagger\Delta} \not\rightarrow^\infty \Delta$ when the contracts in $\mathit{deps}(\Delta) \cap \mathit{deps}(\dagger\Gamma \setminus \mathit{deps}(\Delta))$ are sender-agnostic.*

A direct consequence of Theorem 4, when the dependencies of Δ are sender-agnostic, is that the adversary attacking Δ gains no advantage from creating additional contracts $\Gamma_{\mathcal{M}}$ before the attack. Indeed, if $\Gamma \not\rightarrow^\infty \Delta$ holds, then it also holds when the starting state is extended with $\Gamma_{\mathcal{M}}$. Intuitively, this is because any MEV extraction exploiting $\Gamma_{\mathcal{M}}$ can also be performed by directly calling the dependencies of Δ, since they are not influenced by the caller identity.

Corollary 2 (Resistance to adversarial contracts). *If $\Gamma \not\rightarrow^\infty \Delta$, then $\Gamma \mid \Gamma_{\mathcal{M}} \not\rightarrow^\infty \Delta$ when the contracts in $\mathit{deps}(\Delta)$ are sender-agnostic.*

We summarize in Table 2 some structural properties of $\not\rightarrow^\infty$. The first block shows that it is possible to extend the LHS of $\Gamma \not\rightarrow^\infty \Delta$ with new contracts, while in general it is not possible to extend the RHS. The second block shows that it is possible to cut contracts from the LHS, but not from the RHS. The last block studies how to securely deploy a compound contract $\Delta_1 \mid \Delta_2$ in a state Γ. To have $\Gamma \not\rightarrow^\infty \Delta_1 \mid \Delta_2$ it is not enough to independently check $\Gamma \not\rightarrow^\infty \Delta_1$ and $\Gamma \not\rightarrow^\infty \Delta_2$. Surprisingly, even after checking the secure deployment of the first contract ($\Gamma \not\rightarrow^\infty \Delta_1$) and then that of the second contract in the resulting state ($\Gamma \mid \Delta_1 \not\rightarrow^\infty \Delta_2$), we cannot guarantee the MEV non-interference of $\Delta_1 \mid \Delta_2$. The last row gives a sufficient condition when the component Δ_2 has zero MEV.

DeFi compositions We now evaluate MEV non-interference of typical DeFi compositions. Each line in Table 3 shows a compound contract Δ, the context

Table 3: MEV non-interference of common DeFi compositions ([11], App. A).

Dependencies Γ	**New contracts** Δ	$\Gamma \not\rightsquigarrow^{\infty} \Delta$
AMM	AMM	✓(2)
AMM	$\texttt{Bet}_{\texttt{AMM}}$	✗
Exchange	$\texttt{Bet}_{\texttt{Exchange}}$	✓(3)
AMM1 | AMM2	$\texttt{BestSwap}_{\texttt{AMM1,AMM2}}$	✓(1)
AMM1 | AMM2	$\texttt{SwapRouter}_{\texttt{AMM1,AMM2}}$	✓(1)
AMM1 | AMM2 | $\texttt{SwapRouter1}_{\texttt{AMM1,AMM2}}$ | AMM3 | AMM4 | $\texttt{SwapRouter2}_{\texttt{AMM3,AMM4}}$	$\texttt{BestSwap}_{\texttt{SwapRouter1,SwapRouter2}}$	✓(1)
AMM1 | AMM2 | LP	$\texttt{LPArbitrage}_{\texttt{AMM1,AMM2,LP}}$	✓(1)
AMM1 | AMM2 | LP	$\texttt{FlashLoanArbitrage}_{\texttt{AMM1,AMM2,LP}}$	✓(1)

Γ, and whether $\not\rightsquigarrow^{\infty}$ holds (✓) or not (✗). Rows marked ✓ follow by Theorem 3, and indicate which case of the theorem applies. We omit the contract states in Γ and Δ: a row marked ✓ means that $\not\rightsquigarrow^{\infty}$ holds for *all* contract states, while one marked ✗ means that $\not\rightsquigarrow^{\infty}$ fails to hold for *some* state. Note that the results still hold for larger Γ, by Theorem 4. The pseudo-code of contracts is [11].

The first row shows that an AMM is always MEV non-interfering with another AMM. This holds because $\not\rightsquigarrow^{\infty}$ assumes wealthy adversaries, who always have enough tokens to attack the new AMM in Δ, without the need of exploiting the context Γ. Note that poor adversaries (like in $\not\rightsquigarrow$ and in ε-composability) could instead need to extract tokens from Γ in order to attack Δ. The second and third rows analyse the `Bet` contract. We know from Example 3 that `Bet` is not MEV non-interfering with an `AMM`, since the adversary can always win the bet by creating a price volatility in the `AMM`. This also holds for ε-composability. As expected, MEV non-interference holds when using `Exchange` as a price oracle, since the adversary cannot affect the exchange rate. Note that ε-composability does not properly capture the security of this composition: indeed, `Bet` and `Exchange` are not 0-composable when the adversary (honestly) wins the bet, since winning the bet is indistinguishable from extracting MEV. We then consider two DeFi contracts that act as wrappers of AMMs: `BestSwap` allows users to perform the most profitable swap between two AMMs, while `SwapRouter` routes a swap of $(\texttt{T}_0, \texttt{T}_2)$ across two AMMs for $(\texttt{T}_0, \texttt{T}_1)$ and $(\texttt{T}_1, \texttt{T}_2)$. Despite both contracts have AMMs in their dependencies, they are always securely composable (both $\not\rightsquigarrow^{\infty}$ and 0-composable), because they have zero balance across calls. The same holds for `BestSwapRouter`, that provides the most profitable swap between two `SwapRouter`s. The contracts `LPArbitrage` and `FlashLoanArbitrage` perform a risk-free arbitrage between two AMMs, atomically borrowing and repaying a loan from a Lending Pool (LP) (in `FlashLoanArbitrage`, with no collateral). As before, $\not\rightsquigarrow^{\infty}$ follows since the contracts do not hold a balance between calls. Instead, 0-composability does not hold when the LP fee is greater than 0. If so, borrowing increases the LP balance, possibly increasing the MEV opportunities.

```
contract C1 {
  constructor(?x:T) { require x==1; n=0 }
  f1() { require (n==0); n=1; sender!1:T }
  f2() { require (n==0); n=2 }
  f3() { return n }
}
contract C2 {
  constructor(?x:T) { require x==1 }
  g() { require (C1.f3()==2); sender!1:T }
}
```

Fig. 4: Babel *et al.*' composability does not imply MEV non-interference.

5 Discussion

We have proposed MEV non-interference, a new security notion for DeFi composition which ensures that adversaries cannot inflict economic harm on compound contracts by exploiting their dependencies. We have shown that our notion overcomes the drawbacks of ε-composability, the only other related notion in literature [7]. In particular, while ε-composability is a property of the *whole* blockchain state, MEV non-interference only needs to inspect the newly deployed contracts and their dependencies (Theorem 4). The two notions are incomparable. We already know from Example 2 that MEV non-interference does not imply ε-composability. Example 5 below shows the converse non-implication.

Example 5. Babel *et al.*' composability does not imply MEV non-interference. To show this, we craft 0-composable Γ and Δ where both contracts expose identical and *mutually exclusive* MEV: one can extract MEV from either contract, but not from both. The choice of extracting MEV from Γ or from Δ is done by calling a suitable method of Γ. Only after the choice is made the MEV is exposed, and the MEV from the other contract is permanently disabled. Since the two MEVs are mutually exclusive, we obtain $\mathrm{MEV}(\cdots \mid \Gamma) = \mathrm{MEV}(\cdots \mid \Gamma \mid \Delta)$, hence 0-composability. Non-interference fails because extracting MEV from Δ requires calling a method of Γ. More concretely, using the contracts in Figure 4, let $\Gamma = \texttt{C1}[1{:}\texttt{T}, \texttt{n} = 0]$, and let $S = \texttt{M}[0{:}\texttt{T}] \mid \Gamma$. Assume that $\$\mathbf{1}_{\texttt{T}} = 1$. Let $\Delta = \texttt{C2}[1{:}\texttt{T}]$. To study Babel *et al.*' composability, we compare the (global) MEV of S and $S \mid \Delta$. We have that $\mathrm{MEV}(S) = 1$ by the sequence $\texttt{M}{:}\texttt{C1.f1}()$, and $\mathrm{MEV}(S \mid \Delta) = 1$ by the sequence $\texttt{M}{:}\texttt{C1.f2}()\ \texttt{M}{:}\texttt{C2.g}()$ (or, alternatively, by the sequence $\texttt{M}{:}\texttt{C1.f1}()$, which provides the same MEV). Therefore, the two contracts are 0-composable. To study MEV non-interference, we have that $\mathrm{MEV}(S \mid \Delta, \{\texttt{C2}\}) = 1$ by the sequence $\texttt{M}{:}\texttt{C1.f2}()\ \texttt{M}{:}\texttt{C2.g}()$, while $\mathrm{MEV}_{\{\texttt{C2}\}}(S \mid \Delta, \{\texttt{C2}\}) = 0$, since the only callable method, i.e. $\texttt{g}()$, always fails. Therefore, $S \rightsquigarrow \Delta$, i.e. Δ is not composable with S according to our notion. $\diamond$

We have studied sufficient conditions for MEV non-interference, and rules to enable its modular verification (Theorem 3, Table 2). We have shown that these

rules allow to correctly classify the composability of common DeFi protocols (Table 3). As future work, we envision MEV non-interference as the basis of *quantitative* versions of DeFi composability, which give upper bounds to the loss of a compound contract caused by manipulation of its dependencies.

A relevant question is whether simpler notions of composability would achieve the same effect as our Definition 5. For instance, one might be tempted to regard two contracts $\mathtt{C}$ and $\mathtt{C}'$ composable whenever the (global) MEV of their composition is equal to the sum of the two individual MEVs, thus obtaining a property which could be written along the line of

$$\mathrm{MEV}(\mathtt{C}[\cdot] \mid \mathtt{C}'[\cdot]) = \mathrm{MEV}(\mathtt{C}[\cdot]) + \mathrm{MEV}(\mathtt{C}'[\cdot]) \tag{6}$$

While temptingly simple, this notion has several issues. First, it does not consider the dependencies of the two contracts, which must be part of the blockchain state. E.g., when both contracts depend on a third contract $\mathtt{D}$, we could amend (6) as:

$$\mathrm{MEV}(\mathtt{D}[\cdot] \mid \mathtt{C}[\cdot] \mid \mathtt{C}'[\cdot]) = \mathrm{MEV}(\mathtt{D}[\cdot] \mid \mathtt{C}[\cdot]) + \mathrm{MEV}(\mathtt{D}[\cdot] \mid \mathtt{C}'[\cdot]) \tag{7}$$

However, this equation is almost always false, since in the RHS the MEV extractable from $\mathtt{D}$ is counted twice. Even when $\mathtt{D}$ has no MEV, it can still act as a shared state between $\mathtt{C}$ and $\mathtt{C}'$, e.g. making it possible to extract MEV from only one of them (but not both). This, again, can be used to falsify (7). Furthermore, (7) does not mention the wallet of the adversary. Using the same wallet in each $\mathrm{MEV}(\cdot)$ would duplicate the adversary wealth in the RHS, leading to similar double-counting issues as those discussed previously for $\mathtt{D}$.

We discuss some limitations of our work. First, our blockchain model simplifies Ethereum by requiring that contract dependencies are statically known and acyclic. It looks feasible to refine our results to weaken the assumptions, so that they are only required on the contracts Δ tested for composability (see the end of Appendix B in [11] for a detailed discussion). Note that with this refinement, the rest of the blockchain state is not constrained, making our results applicable to a wider class of contracts. Another limitation is that local MEV measures the loss of a contract as the value of the tokens that adversaries can remove from it: in practice, adversaries could harm contracts also by *freezing* tokens without actually extracting them, as in the infamous Parity Wallet attack [2]. Refining local MEV to take these attacks into account could be done by adapting the notion of liquidity [10]. A further possible improvement of our results is weakening the sufficient conditions of Theorem 3: in particular, condition (3) is unnecessarily strict, since it forbids benign alterations of the state, which do not affect the loss of Δ. Standard static analysis techniques for information flow [24,15,13] could be adapted to refine this condition. Finally, to keep our model simple we assumed that the price of tokens is constant (see Section 2) and that the mempool is empty (Section 3). Relaxing these assumptions is left as future work.

Acknowledgments This work was 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. Understanding the DAO attack, http://www.coindesk.com/understanding-dao-hack-journalists/
2. A Postmortem on the Parity Multi-Sig library self-destruct (November 2017), https://goo.gl/Kw3gXi
3. Defi Pulse: What is DeFi? understanding Decentralized Finance (2019), www.defipulse.com/blog/what-is-defi
4. Furucombo website (September 2023), https://furucombo.app
5. 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
6. 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
7. 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
8. Backes, M., Pfitzmann, B.: Computational probabilistic non-interference. In: European Symposium on Research in Computer Security (ESORICS). LNCS, vol. 2502, pp. 1–23. Springer (2002). https://doi.org/10.1007/3-540-45853-0_1
9. 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
10. Bartoletti, M., Lande, S., Murgia, M., Zunino, R.: Verifying liquidity of recursive Bitcoin contracts. Log. Methods Comput. Sci. **18**(1) (2022). https://doi.org/10.46298/lmcs-18(1:22)2022
11. Bartoletti, M., Marchesin, R., Zunino, R.: DeFi composability as MEV non-interference. CoRR **abs/2309.10781** (2023). https://doi.org/10.48550/ARXIV.2309.10781, https://doi.org/10.48550/arXiv.2309.10781
12. Bartoletti, M., Zunino, R.: A theoretical basis for blockchain extractable value. CoRR **abs/2302.02154** (2023). https://doi.org/10.48550/arXiv.2302.02154
13. Bossi, A., Piazza, C., Rossi, S.: Compositional information flow security for concurrent programs. J. Comput. Secur. **15**(3), 373–416 (2007). https://doi.org/10.3233/jcs-2007-15303
14. 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
15. Dimitrova, R., Finkbeiner, B., Kovács, M., Rabe, M.N., Seidl, H.: Model checking information flow in reactive systems. In: Verification, Model Checking, and Abstract Interpretation (VMCAI). LNCS, vol. 7148, pp. 169–185. Springer (2012). https://doi.org/10.1007/978-3-642-27940-9_12
16. Giacobazzi, R., Mastroeni, I.: Abstract non-interference: parameterizing non-interference by abstract interpretation. In: ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL). pp. 186–197. ACM (2004). https://doi.org/10.1145/964001.964017

17. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy. pp. 11–20. IEEE Computer Society (1982). https://doi.org/10.1109/SP.1982.10014
18. Gudgeon, L., Pérez, D., Harz, D., Livshits, B., Gervais, A.: The decentralized financial crisis. In: Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 1–15. IEEE (2020). https://doi.org/10.1109/CVCBT50464.2020.00005
19. Kitzler, S., Victor, F., Saggese, P., Haslhofer, B.: A systematic investigation of DeFi compositions in Ethereum. In: Financial Cryptography and Data Security Workshops. LNCS, vol. 13412, pp. 272–279. Springer (2022). https://doi.org/10.1007/978-3-031-32415-4_18
20. Kitzler, S., Victor, F., Saggese, P., Haslhofer, B.: Disentangling Decentralized Finance (DeFi) compositions. ACM Trans. Web **17**(2), 10:1–10:26 (2023). https://doi.org/10.1145/3532857
21. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: ACM CCS (2016), http://eprint.iacr.org/2016/633
22. Ryan, P.Y.A., McLean, J.D., Millen, J.K., Gligor, V.D.: Non-interference: Who needs it? In: IEEE Computer Security Foundations Workshop. pp. 237–238. IEEE Computer Society (2001). https://doi.org/10.1109/CSFW.2001.930149
23. Ryan, P.Y.A., Schneider, S.A.: Process algebra and non-interference. In: IEEE Computer Security Foundations Workshop. pp. 214–227. IEEE Computer Society (1999). https://doi.org/10.1109/CSFW.1999.779775
24. Volpano, D.M., Irvine, C.E., Smith, G.: A sound type system for secure flow analysis. J. Comput. Secur. **4**(2/3), 167–188 (1996). https://doi.org/10.3233/JCS-1996-42-304
25. 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
26. Xu, J., Vavryk, N., Paruch, K., Cousaert, S.: SoK: Decentralized exchanges (DEX) with automated market maker (AMM) protocols. ACM Comput. Surv. (nov 2022). https://doi.org/10.1145/3570639