Equilibrium: Preventing Arbitrage Attacks in Optimistic Rollups

Margarita Capretto
IMDEA Software Institute
Universidad Politécnica de Madrid
Madrid, Spain
margarita.capretto@imdea.org

Martín Ceresa
Input Output
Madrid, Spain
martin.ceresa@iohk.io

Hannes Kallwies
Institute for Software Engineering and Programming Languages
University of Lübeck
Lübeck, Germany
kallwies@isp.uni-luebeck.de

César Sánchez
IMDEA Software Institute
Madrid, Spain
cesar.sanchez@imdea.org

***Abstract*—One of the most prominent applications of blockchains is decentralized finance, like Automated Market Makers (AMMs), which are exchange houses without a central authority. AMMs currently handle large amounts of tokens and are subject to vulnerabilities. One attack, called *arbitrage*, involves trading with collections of AMMs in a way that the attacker ends up with more tokens than before trading, thus gaining tokens from the AMMs. Moreover, arbitrage attackers compete by offering higher transaction fees, which affects the whole blockchain ecosystem.**

In this paper, we present *Equilibrium*, a novel solution to prevent arbitrage attacks in Layer-2 optimistic rollup blockchains (L2) where many AMMs currently execute. In our solution, L2 miners re-establish the balance by removing *large* arbitrage opportunities when computing new blocks. The re-balance is computed offchain as a solution to an optimization problem. To ensure the correct evolution of Equilibrium, we provide novel fraud-proof games that guarantee the correctness of the re-balancing process and, consequently, of the blockchain. Our preliminary empirical evidence suggests that re-balancing reduces arbitrage profits while redirecting part of them to AMMs, and is inexpensive to compute and infrequently required.

***Index Terms*—Blockchain, Decentralized Finance, Market Makers, Arbitrage, Layer-2, Optimistic Rollups, Fraud-proofs.**

I. Introduction

Blockchains [33] were introduced as decentralized ledgers enabling electronic payments without the need for trusted third parties. Blockchains later incorporated smart contracts [42], [46], i.e., stateful programs stored on the blockchain. Users can invoke contracts whose execution controls the exchange of cryptocurrency and provide complex functionality, enabling many applications such as decentralized finances (DeFi). Within DeFi, Automated Market Makers (AMMs) are one of the most popular applications. AMMs are contracts that allow trading tokens without central authorities. Users who provide tokens to AMM reserves are known as *liquidity providers*.

The Problem of Arbitrage: AMMs handle large amounts of value [16] and create opportunities for various market mechanisms and attacks, such as *arbitrages*. An arbitrage consists of performing a sequence of trades in a collection of AMMs in such a way that the attacker ends up with more tokens than they started with [3], [19], [21]. The agents executing arbitrages are called *arbitrageurs*. Some arbitrage opportunities are not significant because the fees required to execute them surpass the profit obtained. In this work, we focus on *large* onchain arbitrages.

While arbitrages have the positive effect to balance token prices across the AMMs network [2], [19], they also have a negative impact for both liquidity providers and blockchain users. First, arbitrageurs, who are rarely liquidity providers themselves, extract millions of dollars a year from AMMs [27], [28], [32], [37], [47], that is, from the liquidity providers. Second, arbitrages negatively impact the whole blockchain ecosystem [23], as arbitrageurs compete to be mined first offering higher transaction fees, which drives up costs for all users. While in traditional, non-automated markets, arbitrages are positively perceived and accepted in order to achieve consistent exchange rates, the situation is different in AMMs. Due to the simultaneous availability of all exchange rates in the AMM network and the possibility to automatically execute trades via contracts, the network of AMMs could, in principle, resolve price inefficiencies automatically, rendering arbitrages unnecessary. However, most current research on arbitrages focus on how to detect arbitrages and not how to prevent them [3], [8], [18], [19], [21], [32], [37], [44], [47].

The presence of arbitrage is not local to a contract, but it involves several AMMs. Existing formal techniques for contract reliability include static techniques [1], [4], [9], [10], [17], [34], [36], [39]–[41] and runtime verification [6], [14], [15], [22], [31], but they are not enough to prevent arbitrage attacks, as these techniques focus on the correctness of a single contract or a single execution. Arbitrages consist of sequences of valid trades, and thus, are valid by definition on each individual AMM. The combined effect of the arbitrage as a whole group of transitions is what becomes undesirable. Therefore, detecting and preventing arbitrages requires inspecting multiple AMMs and not just a single contract, which is challenging for existing techniques.

Solution Space: To address these challenges, we leverage Layer-2 Optimistic Rollups (called here simply L2). L2s per-

This work was funded in part by the DECO Project (PID2022-138072OB-I00) funded by MCIN/AEI/10.13039/501100011033 and by the ESF+, FPI PREP2022-000334 and Grant CEX2024-001471-M funded by MICIU/AEI/10.13039/501100011033. This work was supported by Input Output (iog.io).

979-8-3315-8633-1/26/$31.00 ©2026 IEEE

form most computation offchain with minimal L1 blockchain interaction—in terms of frequency and data size—to ensure correctness and trust. Optimistic Rollups are optimistic in the sense that new blocks are assumed correct unless proven incorrect. When L2 blocks are challenged, the challenger and the block proposer play a fraud-proof game (based on Refereed Delegation of Computation (RDoC) [11], [12])—governed by an L1 contract—which guarantees that honest agents always win. Both players place stakes. When the game ends, the loser stake is removed and the winner is compensated. After a predetermined time elapses, surviving blocks consolidate. L2s offer the same guarantees as the underlying blockchain, as far as at least one honest agent proposes and challenges blocks. We target L2s in this work for two reasons. First, modifying an L1 blockchain is difficult, but an L2 is more easily bootstrapped by installing and modifying the L1 contracts encoding the L2 evolution rules, including the fraud-proof games. Second, many AMMs are already running on L2s [35].

Equilibrium: To solve the arbitrage problem, we propose a modified L2 called *Equilibrium*. Transactions in Equilibrium have the same semantics as existing L2s (e.g. Arbitrum [29]), but remove significant arbitrage opportunities by balancing token prices. After computing the effect of each transaction, L2 miners re-balance the network of AMMs with an optimal arbitrage when there are large arbitrage opportunities. The tokens obtained from this arbitrage are distributed in a predetermined deterministic way between the miner and affected AMMs.

We provide sufficient conditions for AMMs under which optimal arbitrages can always be found. However, these arbitrages are not unique so when L2 miners post L2 blocks to L1, they include the arbitrage proposed. Each new block in Equilibrium has the following properties:

- all sequences of trades proposed contain only valid trades;
- the new L2 state is correct;
- large arbitrage opportunities are effectively removed, and
- profits are distributed between AMMs and the L2 miner.

To ensure that proposed L2 blocks satisfy the properties above, we introduce new fraud-proof games where a single honest agent can always win. These games are over *the correctness* of the proposed solution instead of over *traces of the algorithm* that computes the solution (as conventional L2s fraud-proof games do). This offers three advantages: efficiency (verifying optimization solutions is cheaper than generating them), clarity (since checking a solution is simpler the corresponding fraud-proofs are easier to understand) and flexibility (miners can choose the algorithm used to compute solutions). Note that Equilibrium does not change the overall architecture of L2s, only the transaction semantics and fraud-proof games used.

We present a preliminary empirical evaluation over a dataset of one month of transactions involving the popular Uniswap V2 AMMs on deployed Arbitrum, suggesting that our approach is effective, rebalancing is rarely needed and the worst-case computational overhead incurred by L2 miners when computing the optimal arbitrage is small (around 3 seconds).

Contributions: The contributions of this paper are:

(1) a new proof-of-concept L2, called Equilibrium, that removes large arbitrage opportunities eliminating the negative effect of arbitrages while still rebalancing token prices,
(2) novel fraud-proof games that are required to ensure the correctness of solutions, and
(3) a preliminary empirical evaluation quantifying the scalability, effectiveness and intrusiveness of our proposal.

II. Optimistic Rollups

L2 blockchains provide an scalable alternative to Layer-1 (L1) blockchains, like Ethereum [46]. L2s guarantee correctness while performing as much computation as possible offchain with minimal L1 interaction, in terms of the number and size of invocations. Various types of L2 exist, but our solution is based on optimistic rollups which we simply refer to as L2_{OR}.

L2_{OR} work in two phases: users inject transaction requests which are received, sequenced, packed and posted in L1 by a sequencer. Then, the effects of running the sequence of transactions are computed offchain by agents called L2_{OR} miners or State Transition Functions (STFs). STFs are independent parties in charge of computing transaction effects offchain and publishing their effect in L1 encoded as L2_{OR} blocks. As in L1s, the effect of every L2_{OR} transaction is deterministic. Hence, there is a deterministic function, $\mathsf{applyTx}$, that takes an L2_{OR} state S and a transaction t, and computes the resulting L2_{OR} state S'. L2_{OR} blocks contain a reference to the batch of transactions executed, a pointer to the previous L2_{OR} block, and the state obtained after the last transaction (encoded as the hash of an Ethereum's Merkle Patricia state trie [24]). Since L2_{OR} blocks can be potentially incorrect, there is a fixed interval of time during which competing STFs can challenge proposed blocks. Successfully challenged blocks are removed together with all the blocks depending on them. Surviving blocks consolidate and extend the chain.

Fraud-proof games are based on Refereed Delegation of Computation (RDoC) [11], [12], which are two-player turn-based games, played between a proposer and a challenger, governed by L1 contracts. First, players iteratively bisect the L2_{OR} block until the dispute is reduced to a single transaction. At that point, both players agree on the current L2_{OR} state S and the transaction t to be executed, but they disagree on the result of $\mathsf{applyTx}(S, t)$. The game continues bisecting the trace of $\mathsf{applyTx}(S, t)$, encoded in WASM [45], until the dispute is reduced to a single instruction. At this stage, the defender must provide a *one-step proof* which is verified by the L1 contract governing the system. If the proof is valid, the defender wins the game; otherwise the challenger wins. The fraud-proof mechanism satisfies that the honest participant can always win. Consequently, the existence of one honest party guarantees the correct evolution of the L2_{OR}.

III. Market Makers

In this section we model networks of automatic market makers (AMMs), we define arbitrage, establish sufficient con-

ditions under which we can find large arbitrages and show how to automatically remove large arbitrage opportunities. These definitions are based on [2], [8], [19], [21]. An important difference from previous works is that we use bounded integers in our problem and solution domain denoted as $\mathbb{Z}_b$ instead of real numbers. Restricting our problem and solution domain allows us to apply our solutions as invocations to real blockchain contracts adhering their numeric types and precision.

We refer to vectors of n bounded integers as *trades*, where each component represents the amount of tokens being traded [21]. A positive entry indicates the received tokens by a trade, while a negative entry represents the tokens provided. We use Δ to represent trades, Δ^+ for the positive entries within Δ, and Δ^- for the negative ones. We call a trade with all its components zero the *zero trade* and denote it as $\mathbf{0}$.

Definition 1 (AMM): An automated market maker is defined by: (1) a set of possible states $\mathcal{S}$, (2) a predicate $\mathsf{validTrade}$ and (3) a function $\mathsf{executeTrade}$:

$$\mathsf{validTrade} : \mathcal{S} \to \mathbb{Z}_b^n \to \mathbb{B} \qquad \mathsf{executeTrade} : \mathcal{S} \to \mathbb{Z}_b^n \to \mathcal{S}$$

where $\mathsf{validTrade}$ indicates whether a trade is valid in a given state and function $\mathsf{executeTrade}$ provides the resulting state after a trade.

Given an AMM M, we use $M.\mathsf{validTrade}$ for the valid trade predicate. We follow the assumption, common in blockchain AMMs, where each market updates its private state only when a trade is executed in that market. For simplicity, we ignore other interactions with AMMs, such as, when liquidity providers add or remove tokens. The blockchain itself is in charge of reading the AMM state before the trade, executing the requested trade if valid (and failing otherwise), and updating the state of the AMM involved.

A *network of AMMs* is a finite collection of AMMs. We consider a network formed by a fixed collection of m AMMs. A network state refers to the collective state of all AMMs in the network. A network trade, denoted $\overline{\Delta}$, is a vector of trades, one trade for each AMM in the network, $\overline{\Delta} = \langle \Delta_1, \ldots, \Delta_m \rangle$. The *network trade vector* Ψ of a network trade $\overline{\Delta}$ is the sum of the differences between received and tendered tokens over all AMMs in the network: $\Psi(\overline{\Delta}) = \sum_{i=1}^{m} \Delta_i$. The vector Ψ has size n, containing one bounded integer per token. Each entry represents the balance change for the trader. The *size* of a network trade is the number of non-zero trades. We call the network trade with size zero the *null network trade* and denote it as $\overline{0}$. Network trades are valid if and only if all individual trades are valid. Executing network trades is equivalent to executing each individual trade in its corresponding AMM, producing a state change in each AMM involved.

Definition 2 (Arbitrage [3], [21]): Given a network of AMMs, an arbitrage is a valid network trade whose network trade vector is non-negative in all components.

The null network trade $\overline{0}$ is an arbitrage because it is a valid network trade with $\mathbf{0}$ as network trade vector.

At a given point in time, the *profit* of a network trade is the utility obtained from executing it. For simplicity, we compute the utility of a network trade adding the utility obtained for each token. The utility for a given token is computed by multiplying the number of units of that token obtained by its reference market value. In symbols, we work with the following profit function: $\mathsf{profit}(\overline{\Delta}) = \sum_{i=1}^{n} v_i * (\Psi(\overline{\Delta}))_i$ where v_i is the market reference value of token i. We require that the market reference value of each token can be deterministically computed from the states of the AMMs. For each connected component in the network, we select a reference token and assign it a value of 1. For every other token t_i in the component, we identify an AMM M_i that trades t_i with a token $t_{i'}$ whose value has already been determined. Then, the value of token t_i is the maximum amount of $t_{i'}$ obtainable by trading in M_i one unit of t_i times the value of $t_{i'}$. Other profit functions can be employed, provided they are increasing and there is an efficient algorithm to find arbitrages with the highest profit.

We aim to automatically remove arbitrages with a profit that exceeds a fixed threshold α, representing the maximal residual profit exploitable by an attacker. We say that an arbitrage is *large* if its profit is higher than α and we say that the network is in *equilibrium* if there are no large arbitrages.

Constant Function Market Makers: A prevalent class of AMMs is known as *Constant Function Market Makers (CFMMs)* [21]. CFMM follow a constant formula to compute exchange rates. Constant means that the exchange rate preserves a constant relation between amount of tokens before and after the trade. Thus, the state of a CFMM can be represented by its token reserves. The following formula, parameterized by a *trading function* φ, define valid trades in a CFMM:

$$\mathsf{validTrade}(R)(\Delta) \equiv \varphi(R - \gamma * \Delta^- - \Delta^+) \geq \varphi(R)$$

Where R is the reserves of the CFMM, a vector indicating the reserves of each token, and $0 < \gamma < 1$ is the *trade fee* (typically $997/1000$). We list some examples of the CFMMs most commonly present in blockchains:

- *Constant Product Market Makers (CPMM)* such as Uniswap V2 [43] use the following trading function defined for 2 tokens x and y: $\varphi(R) = R_x * R_y$, where R_i denotes the i^{th} component of R.
- *Constant Mean Market Makers (CMMM)* originally proposed by Balancer [7], use the following function for n tokens $\varphi(R) = \prod_{i=1}^{n} R_i^{w_i}$, where $w_1, \ldots, w_n \in \mathbb{R}_+$ represents non-negative weights and $w_1 + \ldots + w_n = 1$.
- *Andre-Style Markets*, employed for stable swaps in exchanges such as Ramses [38] and Camelot [20], utilize the following trading function for 2 tokens x and y: $\varphi(R) = R_x^3 * R_y + R_x * R_y^3$.

Reestablishing the Equilibrium: The price of a given token depends not only on one AMM but on the relative prices of all AMMs that can exchange the token.

Arbitrages are possible because there may be multiple paths to trade two given tokens. After a trade, only the AMM involved updates its state (and, therefore, its prices), potentially creating imbalances relative to the rest of the network. Our goal is to reestablish the equilibrium globally in a network of AMMs after every user transaction. We focus on AMMs

that allow "composing trades", where the equilibrium can be reestablished by exploiting an arbitrage with the highest profit. AMMs are *trade composable* if, for all possible states, the composition of two valid trades is a valid trade. That is, if traders can execute two trades in a row, then they can execute them as a single combined trade. A CFMM is trade composable if its trading function ψ is a quasi-concave and strictly increasing function [2]. All CPMMs, CMMMs, and Andre-Style markets are trade composable. We say that an arbitrage is a *restoring arbitrage* if, once executed, the network is in equilibrium. When all AMMs in a network are trade composable, arbitrages with the highest profit are restoring arbitrages. Moreover, in such networks, if an arbitrage trades in an AMM more than once, there is another arbitrage that composes those trades in a single trade with the same profit. Therefore, in a network of trade composable AMMs, there is always an arbitrage with the highest profit using each AMM at most once.

Theorem 1 (Equilibrium [2], [19]): Executing an arbitrage with the highest profit reestablishes equilibrium in networks of trade composable AMMs.

Finding an Arbitrage with the Highest Profit: Following previous results, we model the problem of finding arbitrages with the highest profit as a non-linear optimization problem. We assume that for all AMMs in our network: (1) in all possible states, the set of valid trades (trading set) is closed and convex, and (2) the zero trade is always considered a valid trade. A sufficient condition for the former conditions is that φ in CFMMs is concave, which holds for all CFMMs used in practice [21]. The optimal arbitrage can then be found by solving the following optimization problem:

Definition 3 (Optimal Arbitrage Problem): Given a network of AMMs $\{M_1, \ldots, M_m\}$, the optimal arbitrage problem consists in finding an arbitrage $\overline{\Delta} = \langle \Delta_1, \ldots, \Delta_m \rangle$ that maximizes traders profit. Formally, let $\mathsf{S}_1, \ldots, \mathsf{S}_m$ be the current state of the network of AMMs, and $v_1, \ldots, v_n$ be the value of each token. The optimal arbitrage is a solution to the following optimization problem:

$$\begin{aligned} \max \quad & \mathsf{profit}(\overline{\Delta}) = \textstyle\sum_{j=1}^{n} v_j * (\sum_{i=1}^{m} \Delta_i)_j \\ \text{subject to} \quad & \forall i \in [1..m], M_i.\mathsf{validTrade}(\mathsf{S}_i, \Delta_i) \\ & (\overline{\Delta} \text{ is a valid network trade}) \\ & \textstyle\sum_{i=1}^{m} \Delta_i \geq \mathbf{0} \quad (\overline{\Delta} \text{ is an arbitrage}) \end{aligned}$$

The problem variables are the network trade $\overline{\Delta}$ defined by one trade at each market $\Delta_i \in \mathbb{Z}_b^n$. This is a convex optimization problem, since the objective function is concave and all trading sets are convex.

A relaxation of the optimal arbitrage problem where trades can have real components is introduced in [3], and can be solved using different algorithms [3], [21]. While non-linear optimization problems for bounded integers are decidable, it is in general computationally challenging to efficiently solve them. However, in certain cases, we can express the optimal arbitrage problem as a linear integer optimization problem, which can be efficiently solved using tools such as OR-Tools [25] or Gurobi [26]. Note that only the constraints enforcing that $\overline{\Delta}$ is a valid network trade may be non-linear. Consequently, the problem admits a linear integer formulation when, for example, all AMMs validity predicates are bi-linear functions, as in CPMMs. We present a preliminary empirical evaluation for CPMMs in Section VI.

IV. Semantics of Equilibrium $\mathrm{L2}_{\mathrm{OR}}$ Blockchains

In this section, we present a novel $\mathrm{L2}_{\mathrm{OR}}$ blockchain semantics that prevents large arbitrage opportunities. We call this new $\mathrm{L2}_{\mathrm{OR}}$ blockchain *Equilibrium* as it maintains the equilibrium in networks of AMMs. If after executing a transaction there is no large arbitrage, Equilibrium performs no additional action as rational arbitrageurs will also perform no action. Otherwise, STFs select a restoring arbitrage and execute it atomically as part of the post-processing of the previous transaction. By Theorem 1, an optimal arbitrage is a restoring arbitrage, so STFs can always find an arbitrage that reestablishes the equilibrium by solving the optimal arbitrage problem (Def. 3). Since the restoring arbitrage may not be unique, STFs explicitly post the arbitrage selected. In Section V we introduce novel fraud-proof games that guarantee that the equilibrium is reestablished correctly, provided there is at least one honest STF and the L1 blockchain ensures both safety and liveness.

A. Evolution of Equilibrium $\mathrm{L2}_{\mathrm{OR}}$ Blockchains

We now present the operational semantics and properties of Equilibrium. We assume that the network of m trade composable AMMs is fixed and trades n tokens, and that the optimal arbitrage problem over bounded integers can be solved efficiently.

The states of Equilibrium encode, as in any other $\mathrm{L2}_{\mathrm{OR}}$, the storage and balance of every account.

A *step* in Equilibrium is defined by the *equilibrium transition* $S \rightarrow_t S_t \Rightarrow_{\overline{\Delta}} S'$ from state S to state S' obtained by executing transaction t and then network trade $\overline{\Delta}$. The first part, $S \rightarrow_t S_t$, is a *traditional* $\mathrm{L2}_{\mathrm{OR}}$ *step* that applies t to obtain the intermediate state S_t. The second part, $S_t \Rightarrow_{\overline{\Delta}} S'$, is an *arbitrage step* which executes network trade $\overline{\Delta}$ at state S_t leading to the final state S'. An equilibrium step is *valid* if both the traditional $\mathrm{L2}_{\mathrm{OR}}$ step and the arbitrage step are valid. A traditional $\mathrm{L2}_{\mathrm{OR}}$ step $S \rightarrow_t S_t$ is valid if $S_t = \mathsf{applyTx}(S, t)$. An arbitrage step $S_t \Rightarrow_{\overline{\Delta}} S'$ is valid if it satisfies the following properties:

- **Arbitrage**: the network trade $\overline{\Delta}$ is an arbitrage at state S_t.
- **Null preference**: Either the profit of $\overline{\Delta}$ is higher than the profitable margin α or $\overline{\Delta}$ is the null network trade.
- **Post State**: There is no arbitrage with profit higher than α in S'.
- **Effect**: S' is the result of executing $\overline{\Delta}$ from S_t, i.e. $S' = \mathsf{effect}(S_t, \overline{\Delta})$.

Since $\mathsf{applyTx}$ is deterministic, we simply use $S \Rightarrow_{t,\overline{\Delta}} S'$ to represent $S \rightarrow_t \mathsf{applyTx}(S, t) \Rightarrow_{\overline{\Delta}} S'$. We force the execution

of arbitrage steps right after each transaction execution as a way to enforce the equilibrium invariant.

Function effect captures the effects of executing network trades, which includes updating the states of the AMMs involved and of contracts that manage tokens. AMM states are updated according to their executeTrade function, while token contracts keep track of the balance of each account. When a trader receives δ tokens from an AMM, the trader's balance increases by δ while the AMM's balance decreases by δ. Symmetrically, the tokens are increased by δ in the AMM and decreased by δ in the trader's balance when the trader contributes. The tokens gained during non-null arbitrage steps are distributed between the STF that computes the new state and the AMMs affected by the arbitrage. We propose that the STF receives a pre-fixed portion of the arbitrage profit to cover for the cost of the arbitrage step, and the remaining tokens are distributed among the AMMs whose net value of traded tokens is negative, proportional to their losses. Token values are computed before the arbitrage is applied, thus the sum of the losses across all AMMs matches the total profit of the executed arbitrage. We leave the analysis of other mechanisms to distribute tokens as future work. Since all trades in arbitrages are valid, and AMMs only accept trades if they have enough tokens, all AMM balances remain non-negative after executing arbitrages. Additionally, as arbitrages trade at each AMM at most once, the execution order of each trade is irrelevant (see Lemma). Based on this, Lemma 2 states that the equilibrium transition effectively reestablishes the equilibrium, which follows directly from property **Post State**.

Lemma 1 (Correct post-state): Let $S \rightarrow_t S_t \Rightarrow_{\overline{\Delta}} S'$ be a valid step. If $\overline{\Delta}$ is $\overline{0}$, then $S' = S_t$. Otherwise, S' matches S_t except for:

- all involved AMM contracts in $\overline{\Delta}$ are updated according to their corresponding execution function and trade in $\overline{\Delta}$;
- token contracts associated to each AMM are updated according to $\overline{\Delta}$ and all tokens gained in $\overline{\Delta}$ are distributed.

Moreover, at S' all accounts have non-negative balances in all tokens contracts, and the total amount of tokens in S_t is the same as in S', i.e. no tokens are burned or minted.

Lemma 2: Let $S \Rightarrow_{t,\overline{\Delta}} S'$ be a valid step. Then, S' is in equilibrium.

When the profitable threshold α is set to infinity, **Null preference** guarantees that the null network trade is selected in all valid equilibrium steps. Thus, arbitrage steps have no effect and the evolution of Equilibrium coincides exactly with traditional L2_{OR} blockchains.

Lemma 3 (Legacy): If α is set to infinity the evolution of the L2_{OR} blockchain under the equilibrium transition is the same as under the traditional L2_{OR} transition.

Following Alg. 1, STFs can always compute a next valid equilibrium step by solving the optimal arbitrage problem (Def. 3). This holds because Theorem 1 ensures that optimal arbitrages are restoring arbitrages. Likewise, this algorithm also ensures the progress of Equilibrium. Given a state S and a transaction t, Alg. 1 finds a tuple that forms a valid step. Alg. 1 first applies transaction t to state S, obtaining S_t. Then, it finds an optimal arbitrage $\overline{\Delta}$ at state S_t by solving the optimal arbitrage problem (function OptimalArbitrage). If the profit of $\overline{\Delta}$ is not higher than α, following property **Null preference**, Alg. 1 returns the null trade, that is $(\overline{0}, S_t)$. Otherwise, Alg. 1 returns the tuple $(\overline{\Delta}, \text{effect}(\overline{\Delta}, S_t))$.

Algorithm 1: Valid equilibrium step.

```
function STEP(S, t)
    S_t ← applyTx(S, t)
    Δ̄ ← OptimalArbitrage(S_t)
    if profit(Δ̄) ≤ α then return (0̄, S_t)
    else return (Δ̄, effect(S_t, Δ̄))
```

Lemma 4 (Progress): Let S be an equilibrium state and t a transaction, there is a network trade $\overline{\Delta}$ such that $S \Rightarrow_{t,\overline{\Delta}} S'$ is a valid step.

B. Equilibrium Blocks

Equilibrium blocks encapsulate a sequence of equilibrium steps. Precisely, an equilibrium block consists of:

- a reference to the previous equilibrium block,
- a sequence of transactions executed in L2_{OR} steps $t_1, \ldots, t_k$,
- a sequence of network trades executed in arbitrage steps $\overline{\Delta}_1, \ldots, \overline{\Delta}_k$,
- a sequence of resulting states $S_1, \cdots, S_k$ after each equilibrium step.

An equilibrium block is valid if and only if all steps are valid when applied in sequence, i.e. $1 \leq i \leq k$, $S_{i-1} \Rightarrow_{t_i,\overline{\Delta}_i} S_i$, and the first equilibrium state is the last state from the previous equilibrium block. From Lemma 4, it follows that STFs can always compute valid equilibrium blocks. Moreover, Lemma 2 guarantees that if an equilibrium block is valid then in all of its intermediate states $S_1, \cdots, S_k$ there are no large arbitrages.

Following traditional L2_{OR} blocks, an equilibrium block on L1 is represented by its post-state and the number of transactions executed, instead of an explicit sequence of states and transactions. To reduce space, a sequence of x consecutive null network trades is compacted with a bit to indicate null and the number x. For every non-null network trades, the STF posts the trade. STFs may try to find arbitrages with minimal size to further reduce the overhead incurred when posting equilibrium blocks in L1. This can be achieved by solving a similar optimization problem after obtaining an optimal arbitrage. In this new problem, the objective is to minimize the number of trades involved subject to a new constraint that restricts the solution space to arbitrages with maximal profit.

Equilibrium blocks consolidate following the same scheme as in traditional L2_{OR} (see Section II). As the validity of equilibrium blocks is different to traditional L2_{OR} blocks, in the next section we present new fraud-proof mechanisms to ensure that only valid equilibrium blocks consolidate.

V. Fraud-proof Games for Equilibrium

In Equilibrium, all arbitrages above some threshold α must be removed after each transaction and the gains must be divided among the AMMs and the STF that proposed the block executing the transaction. Whether an STF has properly removed all arbitrages according to these rules can be challenged by other STFs and decided by fraud-proof mechanisms.

We present here novel fraud-proof mechanisms (also called fraud-proof games or FPs) for this purpose. These FPs enable that a single honest STF can guarantee both liveness and safety of Equilibrium assuming only that L1 ensures both liveness and safety. We focus on games over the validity of proposed arbitrage steps rather than on games about traces of predetermined programs that compute such steps (like in previous FP mechanisms and RDoCs). We exploit the possibility of debunking a proposed equilibrium block by challenging properties of arbitrage steps. For example, to challenge property **Post State** (described in detail later), a challenger STF must provide an arbitrage with profit greater than α in the proposed post state. The advantages of our approach are:

- *Efficiency*: Finding a solution to the optimal arbitrage problem (Def. 3) is computationally more expensive than checking that network trades are valid solutions. Therefore, our game involves far fewer turns than the number of steps required to arbitrate execution traces, in our case, traces of an optimization algorithm.
- *Clarity*: FPs over properties are easy to understand and formalize.
- *Flexibility*: Implementing FPs over the execution of a concrete algorithm would force all STFs to use exactly this algorithm to reestablish the equilibrium. In contrast, our approach gives STFs the freedom to choose any strategy to compute an arbitrage step and, in particular, any optimization algorithm.

A. Equilibrium Fraud-proof Games

Our FPs are multiple games to dispute the validity of equilibrium blocks. These games are governed by an L1 contract referee, similar to FP games in traditional L2_{OR}, and they are all played using L1 transactions. We assume that every transaction submitted to L1 is eventually processed correctly.

FPs are finite turn based two-player games consisting of positions and moves. After a player moves, three things can happen:

1) the referee immediately determines the winner;
2) the game moves to another position within the same game;
3) the players engage into another game.

The last option is added for clarity in the explanation, but in practice all games are encoded as a single game.

We assume that the list of tokens and the network of AMMs are fixed and known by the referee. Moreover, we assume we can compute with a single L1-invocation the profit of a network trade and one component of a network trade vector associated with a given network trade, plus verify whether a trade is valid at a given AMM state.

Our games have two players: a `defender`—the STF agent that posted in L1 an equilibrium block candidate B—, and a `challenger`—an STF agent disputing the validity of B. Players take turns to play with a predetermined time per game which only decreases in their turn, similar to chess clocks (this is already employed in existing L2_{OR}). Given an equilibrium block, we follow the same game as traditional L2_{OR} to reduce it to a dispute over a single step. In our context, this corresponds to an equilibrium step $S \rightarrow_t S_t \Rightarrow_{\overline{\Delta}} S'$ that the `defender` claims to be valid. Checking the validity of $S \rightarrow_t S_t$ is delegated to the legacy L2_{OR} game `L2StepGame` which arbitrates the execution trace of $\mathsf{applyTx}(S, t)$. For the validity of the arbitrage step $S_t \Rightarrow_{\overline{\Delta}} S'$, we introduce a new game, called `arbStepGame`. The `challenger`'s first move depends on the property the `challenger` is disputing:

- **Arbitrage**: The `challenger` disputes that $\overline{\Delta}$ is an arbitrage by playing the `arbitrageGame` game described below.
- **Null preference**: The `challenger` invokes `chNull` providing P claiming that $0 < P < \alpha$ and that $\overline{\Delta}$ has profit P. The game moves to position `nullPreference` where `defender` can either defend the nullity of $\overline{\Delta}$ or challenge any of the other two claims made by the `challenger`. All cases are directly checked by contract referee deciding the winner.
- **Post State**: The `challenger` invokes `chPostState` providing $\overline{\Delta}_c$ and P_c claiming that:
  (1) $\overline{\Delta}_c$ is an arbitrage at state S',
  (2) the profit of $\overline{\Delta}_c$ is P_c, and
  (3) P_c is greater than α.
  The game moves to position `postState` where the `defender` can dispute one of those claims. For claim (1), the `defender` can initiate game `arbitrageGame` (where the roles of the defender and challenger are reversed). For claim (2), the `defender` can invoke `chProfit` $(\overline{\Delta}_c, P_c)$, where contract referee checks whether $\mathsf{profit}(\overline{\Delta}_c) = P_c$. For claim (3), the `defender` can invoke `chAlpha` (P_c), where contract referee checks whether $P_c > \alpha$. In the last two cases the referee contract immediately determines the winner.
- **Effect:** The `challenger` disputes that the network trade is properly executed, moving to game `effectGame`. Game `effectGame` is similar to `L2StepGame` but over the execution trace of function effect instead of function applyTx.

Fig. 1 illustrates game `arbStepGame`. Positions are represented as ovals and sub-games with squares. The initial position of the game is `init`, marked with a double oval. Arrows represent moves, whose destination can be positions in the same game or other games. In the latter case, the initial claim and the role of challenger and defender are provided. Arrows with no destination represent disputes that are resolved immediately by the referee contract. Orange arrows represent enabled moves for the `defender`, while red dashed arrows indicate enabled moves for the `challenger`.

a) Arbitrage Game: The game `arbitrageGame` (see Fig. 2) starts with the claim of the `defender` that $\overline{\Delta}$ is an

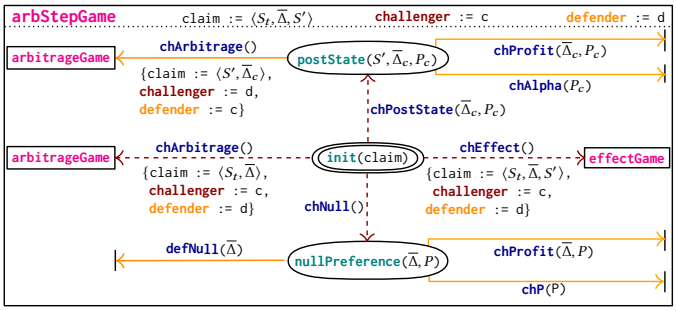

Fig. 1. Positions and moves allowed in game **arbStepGame**.

arbitrage at equilibrium state S posted as the root hash of a Patricia Merkle Tree. This entails that (1) all trades in $\overline{\Delta} = \langle \Delta_1, \ldots, \Delta_m \rangle$ are valid, and (2) all components of the network trade vector $\Psi(\overline{\Delta})$ associated with $\overline{\Delta}$ are non-negative. At the initial position the **challenger** can dispute one of the two claims:

(1) The **challenger** invokes **chTrade** providing i and S_i, claiming that S_i is the state of the i-th AMM in S and Δ_i is not a valid trade at S_i. The game moves to position **trade** where the **defender** plays.
   a) If the **defender** disputes that S_i is the state of the i-th AMM in S, the game continues as a membership proof game on a Merkle tree [13].
   b) If the **defender** disputes that Δ_i is a valid trade at S_i, then the referee checks the validity of the i-th trade and immediately decides the winner.

(2) The **challenger** provides i and invokes **chNTVector** and the referee checks whether $\Psi(\overline{\Delta})_i$ is non-negative and decides the winner.

B. The Honest Player Can Always Win

It is easy to see that every possible disagreement over an equilibrium step between the **challenger** and the **defender** corresponds to a move in one of the games described above. Therefore, the correctness of our fraud-proof games depends on simple checks done by contract referee and on games **L2StepGame** (the FP used in traditional L2_{OR}) and **membershipProofGame** (a known fraud-proof game of membership over Merkle trees). It is well-know that an honest agent can always win these games [13].

The following theorem captures that honest STFs can defend valid equilibrium blocks and enforce that invalid blocks are discarded. Combined with Lemma 4, this guarantees that one single honest STF can ensure progress and safety of Equilibrium.

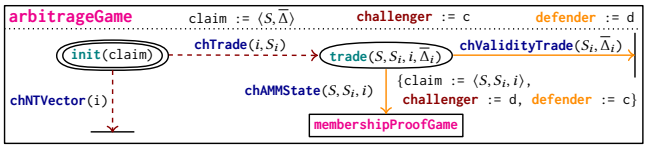

Fig. 2. Positions and moves allowed in game **arbitrageGame**.

The proof of the first part proceeds by reducing the equilibrium block to a single step, and then either playing the **L2StepGame** or defending against the necessarily false accusation in **arbStepGame** of violating one property of the arbitrage step. The proof for the second part is similar, but honest challengers must find the property violated. When the violated property is **Post State** challengers need to provide an arbitrage with profit greater than α in the proposed post state, which can be done by solving the optimal arbitrage problem (Def. 3). All other properties, require only simple arithmetic checks.

Theorem 2: Honest STFs posting Equilibrium blocks can defend them. Honest STFs can have invalid Equilibrium blocks discarded.

Hence, a single honest STF can guarantee the correct evolution of Equilibrium systems, independently of the amount of malicious STFs.

VI. Preliminary Empirical Evaluation

We conducted an empirical evaluation to assess the efficiency and effectiveness of our approach by addressing the following research questions:

- *RQ1. Scalability*: How well does Equilibrium scale on real-world data?
- *RQ2. Efficiency*: Does Equilibrium effectively reduce the profit that arbitrageurs could extract, while redirecting part of it to AMMs?
- *RQ3. Intrusiveness*: How often is the equilibrium reestablished?

Data set and Implementation: To answer these questions, we focused on Uniswap v2 contracts deployed on Arbitrum and analyzed their transactions during April 2025. The Optimal Arbitrage Problem (Def. 3) for Uniswap V2 is a bilinear optimization problem. We applied bit-blasting [30] in one variable per AMM transforming the problem into a linear optimization problem, which we solved using Gurobi [26].

Uniswap v2 in Arbitrum uses 112-bit integers for reserves but typically use 18 decimals for user representation. Since current solvers struggle with large integers, we truncated the decimals in the user representation, and eliminated AMMs that ended with empty reserves or huge reserves in at least one of the tokens. Then, we created a graph where each node represents a token and there is an edge between two nodes if an AMM exchanges those tokens. We iteratively removed AMMs for which at least one of its tokens had degree 1 in the graph, as they could never participate in an optimal arbitrage. The resulting graph consists of a single connected component with 7 tokens and 14 AMMs, which were involved in $43,563$ transactions during April 2025. After each transaction, to compute the value of a token, we used the formula from Section III with token USDC as the reference token.

Experimental setup: We simulated the evolution of an Equilibrium blockchain on a Linux machine with 16 GB of RAM and 8-core Intel i7-8565U processor running Ubuntu 24.04.3. We used the state of Arbitrum in the first block of April as initial state, and we executed all $43,563$ transactions

α	M.T.	A.T.	M.S.	A.S.	I.	P.I.
\$0	3.23s	0.10s	7	3.75	352	0.80%
\$47.8	2.97s	0.12s	8	4.22	179	0.41%
\$60.4	1.57s	0.13s	8	4.46	167	0.38%
\$75.7	1.59s	0.32s	8	4.61	135	0.30%
\$108.9	2.06s	0.16s	8	4.72	113	0.26%
\$172.4	2.41s	0.19s	8	4.87	83	0.19%
\$$\infty$	-	-	-	-	0	0.00%

α	M.P.	A.P.	S.P.	M.R.P.	A.R.P.
\$0	\$791.2	\$46.6	\$16,389.8	\$0.0	\$0.0
\$47.8	\$925.0	\$113.9	\$20,395.1	\$47.6	\$15.0
\$60.4	\$880.2	\$129.8	\$21,681.9	\$60.4	\$19.6
\$75.7	\$1,236.3	\$107.2	\$19,790.3	\$75.6	\$28.1
\$108.9	\$1,236.3	\$191.2	\$21,609.0	\$108.8	\$38.8
\$172.4	\$1,131.5	\$265.0	\$21,990.9	\$171.6	\$62.8
\$$\infty$	\$0.0	\$0.0	\$0.0	\$567.0	\$34.4

TABLE I

EMPIRICAL RESULTS FOR DIFFERENT VALUES OF α. M.T. AND A.T.: MAXIMAL AND AVERAGE TIME (RESPECTIVELY) OF EXECUTING FUNCTION OptimalArbitrage. M.S. AND A.S.: MAXIMAL AND AVERAGE (RESPECTIVELY) NUMBER OF TRADES IN NO-NULL ARBITRAGES USED IN ARBITRAGES STEPS. M.P., A.P, S.P.: THE MAXIMAL, AVERAGE, AND SUM (RESPECTIVELY) PROFIT OF NO-NULL ARBITRAGES APPLIED IN THE ARBITRAGE STEP. M.R.P. AND A.R.P.: MAXIMAL AND AVERAGE (RESPECTIVELY) RESIDUAL PROFIT. I. AND P.I.: TOTAL AND PORTION OF INTERVENTION IN THE TRACE.

involving an AMM in the network. First, we computed the trace for $\alpha = \$0$, i.e., a no-null optimal arbitrage (if it exists) is executed after each transaction. Then, to get realistic α values for our experiments, we computed the profit for the 75$^{\text{th}}$, 80$^{\text{th}}$, 85$^{\text{th}}$, 90$^{\text{th}}$, and 95$^{\text{th}}$ profit percentiles of the applied arbitrage, resulting in α of \$47.8, \$60.4 \$75.7, \$108.9 and \$172.4, respectively. For comparison purposes, we simulated $\alpha = \$\infty$, computing but never executing any arbitrage (i.e. the legacy L2_{OR} execution without re-balancing). The collected data is summarized in Table I.

Analysis: We address each research question based on the simulation results:

RQ1. Scalability: The maximum execution time was $3.23s$, with an average of $0.13s$, far below the average interval (one minute) between two transactions involving AMMs in our network. This suggests that arbitrage steps are not a significant computational overhead. Arbitrages in the arbitrage step involved at most 8 trades, with an average between 3.75 to 4.86 for non-null trades. Thus, the data that STFs must post to L1 in an equilibrium block is small.

RQ2. Efficiency: We measured two quantities: (1) the residual profit, i.e., the maximal arbitrage profit remaining after each equilibrium step (the amount that arbitrageurs could still extract); and (2) the profit of arbitrages extracted during arbitrage steps. With $\alpha = \$\infty$ (legacy execution), the maximal residual profit was \$567.0 with an average of \$34.4 (we count subsequent residual profits with same value only once, as it would disappear if exploited), with no gains for AMMs. By lowering α, residual profits decreased substantially because AMMs regain the potential arbitrages. For example, with $\alpha = \$47.8$, the maximal residual profit drops to \$47.6 with an average of \$15, that is, an 8.39% and 43.13% respectively of the legacy execution, while on average \$113.9 were distributed to the AMMs per intervention, for a total of $\$20,395$ in the whole month.

RQ3. Intrusiveness: With $\alpha = \$0$, Equilibrium intervened 352 times out of the 43,563 transactions (0.8%). As α increases, the number of interventions decreases, down to 89 times (0.19%) for $\alpha = \$172.4$. Thus, Equilibrium rarely intervenes, yet still captures a significant portion of arbitrage profits.

Conclusion: Our evaluation suggests that Equilibrium scales to real-world AMM networks (provided that the solvers scale to the numerical size of the reserves), reduces extractable arbitrage profit, and intervenes only rarely, with trade-offs depending on α.

VII. CONCLUSIONS AND FINAL DISCUSSION

AMM networks suffer from arbitrage attacks that exploit price discrepancies. Agents competing for arbitrage opportunities extract value from liquidity providers and also negatively impact the blockchain ecosystem by driving up transaction fees and network congestion.

We presented Equilibrium, a novel proof-of-concept L2_{OR} blockchain that automatically removes large arbitrages from a network of trade composable AMMs. We provided new fraud-proof games ensuring the correct evolution of Equilibrium. Unlike conventional FPs, our games verify the correctness of a proposed solution rather than execution traces. In Equilibrium, STFs compute offchain arbitrages by solving bounded integer non-linear optimization problems (Definition 3), and apply them to reestablish the equilibrium and distribute the earnings among AMMs and STFs. For the popular constant-product market makers [21] we showed empirically that our proposal is feasible, removing large arbitrage opportunities while intervening rarely and imposing little overhead. Applying this approach to other AMMs requires efficiently solving the optimization problem from Definition 3, which is an interesting future research topic. Two alternatives are (1) to adapt Equilibrium for AMM networks where the optimal arbitrage can be approximated within an additive factor, and (2) to detect external arbitrages and mimic them extracting the profit before the arbitrageur.

Finally, we assumed that the list of tokens and AMMs is fixed. However, in practice, new tokens and AMMs can be created. Extending Equilibrium to dynamic AMMs and tokens would require updating state variables in L1 contracts. Similarly, changing threshold α—that determines acceptable arbitrages—or the function to determine the value of each token can also be done by updating the L1 contracts that govern Equilibrium. Mechanisms to perform such updates are future work. A potential approach could be similar to Arbitrum DAO [5], where stakeholders vote on on-chain proposals that influence the operation and evolution of Arbitrum ecosystem.

References

[1] Ahrendt, W., Bubel, R.: Functional verification of smart contracts via strong data integrity. In: Proc. 9th Int'l Symp, on Leveraging Applications of Formal Methods, Verification and Validation: Applications (ISoLA'20)). LNCS, vol. 12478, pp. 9–24. Springer (2020). https://doi.org/10.1007/978-3-030-61467-6_2, https://doi.org/10.1007/978-3-030-61467-6_2

[2] Angeris, G., Chitra, T.: Improved price oracles: Constant function market makers. In: Proc. of the 2nd ACM Conference on Advances in Financial Technologies (AFT '20). pp. 80–91. ACM (2020). https://doi.org/10.1145/3419614.3423251, https://doi.org/10.1145/3419614.3423251

[3] Angeris, G., Evans, A., Chitra, T., Boyd, S.: Optimal routing for constant function market makers. In: Proc. of the 23rd ACM Conference on Economics and Computation (EC'22). pp. 115–128. ACM (2022). https://doi.org/10.1145/3490486.3538336, https://doi.org/10.1145/3490486.3538336

[4] Annenkov, D., Nielsen, J.B., Spitters, B.: ConCert: a smart contract certification framework in Coq. In: Proc. of the 9th ACM SIGPLAN Int'l Conf. on Certified Programs and Proofs (CPP'20). pp. 215–218. ACM (2020). https://doi.org/10.1145/3372885.3373829, http://dx.doi.org/10.1145/3372885.3373829

[5] Arbitrum Foundation: Arbitrum DAO - Governance docs, https://docs.arbitrum.foundation/concepts/arbitrum-dao

[6] Azzopardi, S., Ellul, J., Pace, G.J.: Monitoring smart contracts: ContractLarva and open challenges beyond. In: Proc. of the 18th Int'l Conf. on Runtime Verification (RV'18). LNCS, vol. 11237, pp. 113–137. Springer (2018). https://doi.org/10.1007/978-3-030-03769-7_8, https://doi.org/10.1007/978-3-030-03769-7_8

[7] Balancer: Balancer documentation. https://docs.balancer.fi/

[8] Bartoletti, M., yu Chiang, J.H., Lluch-Lafuente, A.: A theory of automated market makers in defi. In: Coordination Models and Languages. pp. 168–187. Springer International Publishing (2021)

[9] Bernardo, B., Cauderlier, R., Hu, Z., Pesin, B., Tesson, J.: Mi-Cho-Coq, a framework for certifying Tezos smart contracts. In: Proc. of the 1st Workshop on Formal Methods for Blockchains (FMBC'19). LNCS, vol. 12232, pp. 368–379. Springer (2019). https://doi.org/10.1007/978-3-030-54994-7_28, https://doi.org/10.1007/978-3-030-54994-7_28

[10] Bhargavan, K., Delignat-Lavaud, A., Fourneta, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., Béguelin, S.Z.: Formal verification of smart contracts: Short paper. In: Proc. of Workshop on Programming Languages and Analysis for Security (PLAS'16). pp. 91–96. ACM (2016). https://doi.org/10.1145/2993600.2993611, https://doi.org/10.1145/2993600.2993611

[11] Canetti, R., Riva, B., Rothblum, G.N.: Practical delegation of computation using multiple servers. In: Proc. of the 18th ACM Conf. on Computer and Communications Security (CCS'11). pp. 445–454. ACM (2011). https://doi.org/10.1145/2046707.2046759, https://doi.org/10.1145/2046707.2046759

[12] Canetti, R., Riva, B., Rothblum, G.N.: Refereed delegation of computation. Information and Computation **226**, 16–36 (2013). https://doi.org/https://doi.org/10.1016/j.ic.2013.03.003, https://www.sciencedirect.com/science/article/pii/S0890540113000217

[13] Capretto, M., Ceresa, M., Anta, A.F., Moreno-Sanchez, P., Sánchez, C.: A secure sequencer and data availability committee for rollups. In: Proc. of the 2025 ACM SIGSAC Conference on Computer and Communications Security, (CCS'25). pp. 2129–2143. ACM (2025). https://doi.org/10.1145/3719027.3765187, https://doi.org/10.1145/3719027.3765187

[14] Capretto, M., Ceresa, M., Gorostiaga, F., Macías, F., Pedregal, P., Sánchez, C.: Offchain runtime verification (for the tezos blockchain). In: Computer Security. ESORICS 2024 International Workshops - DPM, CBT, and CyberICPS. LNCS, vol. 15263, pp. 242–259. Springer (2024). https://doi.org/10.1007/978-3-031-82349-7_17, https://doi.org/10.1007/978-3-031-82349-7_17

[15] Capretto, M., Ceresa, M., Sánchez, C.: Transaction monitoring of smart contracts. In: Proc. of the 22nd Int'l Conf. on Runtime Verification (RV'22). LNCS, vol. 13498, pp. 162–180. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_9, https://doi.org/10.1007/978-3-031-17196-3_9

[16] Coingeko: Automated Market Maker (AMM) market capitalization, https://www.coingecko.com/en/categories/automated-market-maker-amm

[17] Conchon, S., Korneva, A., Zaïdi, F.: Verifying smart contracts with Cubicle. In: Proc. of the 1st Workshop on Formal Methods for Blockchains (FMBC'19). LNCS, vol. 12232, pp. 312–324. Springer (2019). https://doi.org/10.1007/978-3-030-54994-7_23, https://doi.org/10.1007/978-3-030-54994-7_23

[18] 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: 2020 IEEE Symposium on Security and Privacy (SP'20). pp. 910–927 (2020). https://doi.org/10.1109/SP40000.2020.00040

[19] Danos, V., Khalloufi, H.E., Prat, J.: Global order routing on exchange networks. In: Financial Cryptography and Data Security. FC 2021 Int'l Workshops. pp. 207–226. Springer Berlin Heidelberg (2021)

[20] DEX, C.: Camelot documentation. https://docs.camelot.exchange/

[21] Diamandis, T., Resnick, M., Chitra, T., Angeris, G.: An efficient algorithm for optimal routing through constant function market makers. In: Financial Cryptography and Data Security (FC'24). pp. 128–145. Springer Nature Switzerland (2024)

[22] Ellul, J., Pace, G.J.: Runtime verification of Ethereum smart contracts. In: Proc. of the 14th European Dependable Computing Conf. (EDCC'18). pp. 158–163. IEEE Computer Society (2018). https://doi.org/10.1109/EDCC.2018.00036, https://doi.org/10.1109/EDCC.2018.00036

[23] Ethereum Foundation: Maximal extractable value (MEV). https://ethereum.org/en/developers/docs/mev/

[24] Ethereum Foundation: Merkle Patricia Trie. https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/

[25] Google for Developers: Google OR-Tools, https://developers.google.com/optimization

[26] Gurobi Optimization, LLC: Gurobi, https://www.gurobi.com/

[27] halo3mic: Arbitrum arbitrage full. https://dune.com/halo3mic/arbitrage-2

[28] halo3mic: Arbitrum atomic arbitrage. https://dune.com/halo3mic-plus/arbitrum-arbitrage

[29] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: Proc. of the 27th USENIX Conf. on Security Symposium (USENIX Security 18). pp. 1353–1370. USENIX Association (2018), https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner

[30] Kroening, D., Strichman, O.: Bit Vectors, pp. 135–156. Springer Berlin Heidelberg (2016). https://doi.org/10.1007/978-3-662-50497-0_6, https://doi.org/10.1007/978-3-662-50497-0_6

[31] Li, A., Choi, J.A., Long, F.: Securing smart contract with runtime validation. In: Proc. of 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'20). pp. 438–453. ACM (2020). https://doi.org/10.1145/3385412.3385982, https://doi.org/10.1145/3385412.3385982

[32] McLaughlin, R., Kruegel, C., Vigna, G.: A large scale study of the ethereum arbitrage ecosystem. In: 32nd USENIX Security Symposium (USENIX Security'23). pp. 3295–3312. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/mclaughlin

[33] Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2009)

[34] Nehaï, Z., Bobot, F.: Deductive proof of industrial smart contracts using Why3. In: Proc. of the 1st Workshop on Formal Methods for Blockchains (FMBC'19). LNCS, vol. 12232, pp. 299–311. Springer (2019). https://doi.org/10.1007/978-3-030-54994-7_22, https://doi.org/10.1007/978-3-030-54994-7_22

[35] Offchain Labs: DEX projects in Arbitrum. https://portal.arbitrum.io/projects?subcategories=dex

[36] Permenev, A., Dimitrov, D., Tsankov, P., Drachsler-Cohen, D., Vechev, M.: VerX: Safety verification of smart contracts. In: Proc. of the 41st IEEE Symp. on Security and Privacy (S&P'20). pp. 1661–1677. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00024, https://doi.org/10.1109/SP40000.2020.00024

[37] Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? 2022 IEEE Symposium on Security and Privacy (SP'22) pp. 198–214 (2021)

[38] Ramses: Ramses documentation. https://docs.ramses.exchange/

[39] Schiffl, J., Ahrendt, W., Beckert, B., Bubel, R.: Formal analysis of smart contracts: Applying the KeY system. In: Deductive Soft-

ware Verification: Future Perspectives - Reflections on the Occasion of 20 Years of KeY, LNCS, vol. 12345, pp. 204–218. Springer (2020). https://doi.org/10.1007/978-3-030-64354-6_8, https://doi.org/10.1007/978-3-030-64354-6_8

[40] Sergey, I., Kumar, A., Hobor, A.: Scilla: a Smart Contract Intermediate-Level LAnguage. https://arxiv.org/abs/1801.00687 (2018)

[41] Stephens, J., Ferles, K., Mariano, B., Lahiri, S., Dillig, I.: SmartPulse: Automated checking of temporal properties in smart contracts. In: Proc. of the 42nd IEEE Symp. on Security and Privacy (S&P'21). pp. 555–571. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00085

[42] Szabo, N.: Smart contracts: Building blocks for digital markets. Extropy **16** (1996)

[43] Uniswap: Uniswap v2 documentation. https://docs.uniswap.org/contracts/v2/overview

[44] Wang, Y., Chen, Y., Wu, H., Zhou, L., Deng, S., Wattenhofer, R.: Cyclic arbitrage in decentralized exchanges. In: Companion Proc. of the Web Conference 2022 (WWW'22). pp. 12–19. ACM (2022). https://doi.org/10.1145/3487553.3524201, https://doi.org/10.1145/3487553.3524201

[45] WebAssembly Working Group: WebAssembly, https://webassembly.org/

[46] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper (2014)

[47] Zhou, L., Qin, K., Cully, A., Livshits, B., Gervais, A.: On the just-in-time discovery of profit-generating transactions in defi protocols. In: 42nd IEEE Symposium on Security and Privacy (SP'21). pp. 919–936. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00113, https://doi.org/10.1109/SP40001.2021.00113