Netting-based Liquidity-saving Automated Market Makers

Margherita Renieri[a,*], Letterio Galletta[a], Alberto Lluch Lafuente[b], Aleksander Junge[b], James Hsin-yu Chiang[c]

[a]*IMT School for Advanced Studies Lucca, Lucca, Italy*
[b]*DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark*
[c]*Aarhus University, Aarhus, Denmark*

Abstract

Automated Market Makers (AMMs) are one of the most used Decentralized Finance services enabling users to exchange crypto-assets directly without intermediaries. However, current protocols impose significant constraints on the liquidity levels required for transactions. In this paper, we propose a liquidity-saving mechanism designed to minimize the liquidity required by AMM services. Our mechanism delays the transactions violating the liquidity constraints in a queue, and, when certain conditions are met, it selects from the queue a feasible transaction sequence that fulfills the constraints and executes them atomically on the blockchain. We provide an operational semantics of such a mechanism that precisely characterizes the interactions between users and AMMs and the conditions when the liquidity-saving mechanism is triggered. Moreover, we show that our mechanism allows for novel liquidity saving behavior for multi-party exchange, multi-AMM arbitrage, and enhances user intent compared to traditional AMMs. Finally, to validate our approach, we develop a simulator and experiment with various application scenarios, yielding insights into the practical implications of our mechanism.

Keywords: Formal methods, Operational semantics, Liquidity-Saving Mechanism, Netting, AMMs

1. Introduction

In Decentralized Finance (DeFi) systems, Automated Market Makers (AMMs) are decentralized exchanges (DEXs) that provide users with liquidity using different reserves, enabling them to trade different crypto-assets, i.e., tokens. Users participating in these token exchanges pay fixed trading and gas fees charged by the blockchain network. Similarly to traditional financial systems, users need to provide an upfront amount of crypto-assets as collateral to operate with such decentralized financial services. Liquidity is essential for AMMs to work properly, otherwise, it is possible to incur scenarios that hinder their economic performance and limit user involvement.

AMMs rely on users who act as liquidity providers by depositing a certain amount of tokens into AMM smart contracts that can be used for trades. AMM protocols implement *liquidity-saving mechanisms* (LSM) that aim at reducing liquidity costs by minimizing the required liquidity for transaction settlement. DeFi protocols have introduced various approaches to achieve this goal, e.g., flash loans [1, 2] and flash swaps [3]. These mechanisms allow users to temporarily loan tokens with no collateral but require them to pay back their debt within a single transaction. One of their limitations is that they do not easily permit scenarios with more users involved or trades requiring more than one transaction to be completed.

In this paper, we propose the design of an LSM for AMMs that offers the advantages of flash loans and flash swaps but overcomes their limitations. In particular, our mechanism enables users to operate on the AMM without upfront balance by allowing temporary overdrafts that will be eventually covered, and by aggregating such actions over a specific time frame. Our aggregation mechanism is inspired by *netting* [4, 5, 6], an LSM used in traditional finance: netting creates payment instruction queues among banks, offsets the value of multiple payments, and finds the most convenient payment combination to eliminate stunting periods.

We provide here a formalization of our mechanism in terms of a labeled transition system (LTS) that precisely characterizes the interactions between users and AMMs, the conditions when the netting mechanism is triggered, and allows us to prove some formal properties enjoyed by our mechanism.

Intuitively, our mechanism works as follows. First, we equip AMMs with a finite queue to store trading transactions proposed by users. When a user performs a transaction that would result in a balance overdraft, the transaction is stored in the queue. Otherwise, if the transaction has the effect of setting all balances of users having pending transactions in the queue to a positive value, the transaction queue is executed atomically, resulting in a liquidity-saving sequence of actions. If the queue reaches its maximum capacity where some users' balances are still negative, a netting algorithm selects a subset of the stored transactions that meet the liquidity constraints for each user, if any, and atomically performs them. In such a way, users capitalize on market inefficiencies, facilitating the trading of assets more efficiently and cost-effectively. We formalize those sce-

*Corresponding author.
Email addresses: margherita.renieri@imtlucca.it (Margherita Renieri), letterio.galletta@imtlucca.it (Letterio Galletta), albl@dtu.dk (Alberto Lluch Lafuente), s223125@student.dtu.dk (Aleksander Junge), jachiang@cs.au.dk (James Hsin-yu Chiang)

narios where our netting-based AMMs ensure users fulfill their intents in terms of token achievements even when they do not have enough liquidity, whereas traditional AMMs do not allow them to operate on the market. In particular, we prove that our mechanism allows a user to successfully increase her reserve of a given token even if she initially has no liquidity, and that it enables multi-party trading operations that are not executed with standard AMMs due to liquidity constraints.

In summary, our main contributions are:

- We introduce an LSM that exploits a netting algorithm to settle transactions and allows us to implement liquidity-optimized AMM. We show that our mechanism allows users to perform multi-party exchanges and multi-AMM arbitrage without the upfront liquidity required by traditional AMMs.
- We formalize our mechanism through a two-level operational semantics. The first level addresses the interactions between users and AMMs. This semantics is derived from the one of Bartoletti *et al.* [7] by removing the liquidity constraints on users' balances when performing swap actions. The second level semantics formalizes our liquidity-saving mechanism and describes when transactions leading to balance overdrafts are enqueued and when the netting algorithm is executed to select the transactions to be settled.
- We formally prove that in some scenarios our LSM achieves users' intents and improves over standard AMMs.
- We provide a polynomial netting algorithm that can run on the blockchain as part of a smart contract to select a subsequence of actions that can be performed without violating any liquidity constraints. We also prove a property of how this algorithm deals with dependencies among actions.
- We implement a Haskell simulator that is open source and available online [8] and allows us to simulate our mechanism and experiment with several application scenarios.

In the rest of the paper we proceed as follows. We provide a high-level overview of our approach in Section 2. Section 3 presents the operational semantics of our LSM mechanism, and proves its formal properties about its profitability against standard AMMs. In Section 4, we illustrate our netting algorithm and prove its properties. Section 5 describes our simulator and some application scenarios. In Section 6, we provide a brief overview of related literature and explore an alternative design that enables offline netting. Additionally, we discuss how emerging trends, challenges, and opportunities within the blockchain ecosystem could influence the practical implementation and scalability of our solution. Section 7 concludes and discusses some future work.

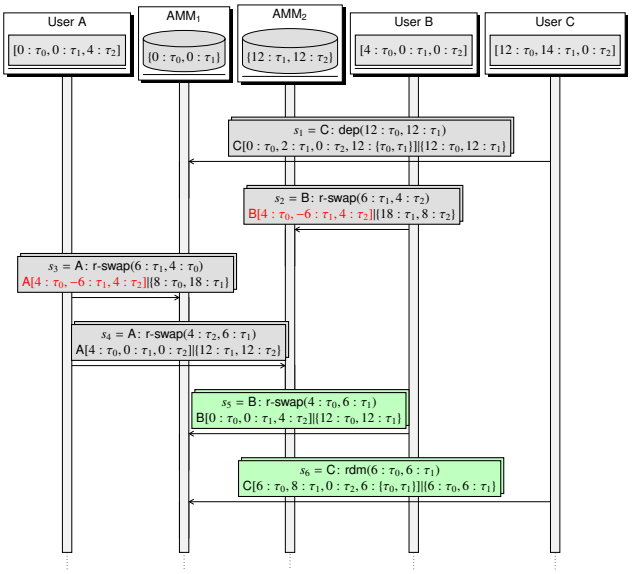

Figure 1: Interaction between three Users and two AMMs.

Comparison with previous work. A preliminary version of this paper was presented at DLT Workshop 2024 [9]. In our previous work, we presented a single-level semantics for our mechanism that only considers swap actions. In this paper, we extend the previous semantics by introducing other possible actions available to users and AMMs for depositing and redeeming specific token amounts (Section 3). This improvement allows us to capture various dynamics that our netting mechanism enables and to model more realistic scenarios. Moreover, in this paper, we characterize and prove that in some specific scenarios, our LSM allows users to achieve their intents in terms of token liquidity compared to standard AMMs. Additionally, we extend the previous work by implementing a simulator of our mechanism that allows us to experiment with multiple application scenarios, providing valuable insights into the practical implications of our mechanism (Section 5).

In this paper, we also prove that our netting algorithm is quadratic in the size of the queue, and other results on its behavior when it removes dependent actions (Section 4).

Additionally, we discuss here an alternative approach for implementing the netting mechanism offline, emphasizing the importance of incorporating an incentive mechanism to mitigate the high gas fees and costs associated with the netting procedure. Finally, we analyze how the evolving landscape of the blockchain ecosystem, including new trends, challenges, and opportunities, may affect the feasibility and scalability of implementing our solution in real-world applications (Section 6).

2. Overview of the approach

In current DeFi protocols, such as Uniswap [3], a user interacts with the AMM reserve through single actions. She can deposit, redeem, or swap crypto-assets in change of others in the AMM. However, she can perform the transaction only if she has the needed up-front capital to cover it. For instance, consider the example of Figure 1 where there are three users

A, B, and C that interact with two AMMs performing a sequence of actions $s_1 s_2 s_3 s_4 s_5 s_6$. In detail, user A owns 4 tokens of type τ_2, no tokens of types τ_0 and τ_1 (denoted with the notation $\mathsf{A}[0 : \tau_0, 0 : \tau_1, 4 : \tau_2]$ at the top of the figure), user B has 4 tokens of type τ_0, no tokens of types τ_1 and τ_2; whereas user C owns 12 tokens of type τ_0 and 14 tokens of type τ_1. The intent of users A and B is to increase their amount of token type τ_0 and τ_2, respectively. For this reason, they would like to exchange a certain amount of tokens of type τ_1 with the available AMMs with 1-to-1 rate to achieve their goals, even without having enough tokens in their wallet. Initially, user C with action s_1 creates the AMM_1 depositing 12 tokens of type τ_0 and 12 tokens of type τ_1 becoming the liquidity provider of this new AMM reserve. For this deposit, C receives equal amounts of minted tokens written as $\{\tau_0, \tau_1\}$ that work as a receipt of her deposit and that can be used afterward by C to redeem her investment. After this deposit, the AMMs have 12 tokens type of τ_0, τ_1 and τ_1, τ_2, respectively (denoted with $\{12 : \tau_0, 12 : \tau_1\}$ and $\{12 : \tau_1, 12 : \tau_2\}$ in the figure). In standard protocols like Uniswap, transactions s_2 and s_3 are rejected because users' balances cannot cover their fulfillment: B cannot swap 6 tokens of type τ_1 for 4 tokens of type τ_2, in symbols $s_2 = \mathsf{B}\colon \mathsf{r\text{-}swap}(6 : \tau_1, 4 : \tau_2)$, and A cannot swap 6 token of type τ_1 for 4 token of type τ_0, in symbols $s_3 = \mathsf{A}\colon \mathsf{r\text{-}swap}(6 : \tau_1, 4 : \tau_0)$. The same also holds for actions s_4 and s_5 that users A and B cannot afford.

Our LSM overcomes the previous scenario: the underlying idea is to accept a momentary deficit in users' balances as long as they are covered by subsequent transactions, for instance, a swap in a different direction made by the same user or an update to the token reserve. More precisely, transactions violating liquidity constraints on user wallets are delayed and stored in a pending queue. If a subsequent transaction covers the deficit of users, all the transactions in the queue are performed atomically. Otherwise, when the queue reaches a certain length, our LSM executes a netting procedure to discard from the queue the transactions that make user balances negative, and that cannot be covered. Thus, the protocol reaches a state where no liquidity constraints are violated, and transactions can be safely performed.

Back to the example of Figure 1, our LSM allows the sequence of the actions $s_1 s_2 s_3 s_4 s_5 s_6$ (we report how each action affects the user balances and AMM reserves in the figure) to be executed since s_5 yields a state where all balances are positive, so the queue is emptied. It is not relevant if there are intermediate states where user balances are temporarily negative (colored in the figure) because they will be eventually balanced. Finally, user C redeems $6 : \{\tau_0\}$ and $6 : \{\tau_1\}$ with transaction s_6. In this way, our LSM permits a multi-party trade among A, B, and C, even when they individually do not have sufficient funds. In contrast, traditional protocols reject all four swap actions due to the lack of liquidity.

3. The formal model of the liquidity-saving mechanism

This section formalizes our LSM, and proves some properties it enjoys. More precisely, we formalize user interaction and AMMs as a labeled transition system (LTS). Our LTS builds on the work of Bartoletti *et al.* [7], from which we adopt both the structure and the notation. Intuitively, LTS states represent the system configurations where we store the token supplies owned by each user and the reserves of each AMM. Note that in our model we assume that a user can submit a transaction to the AMM whenever she wants. Transitions represent transactions performed by users, which may result in an update of token supplies or AMM reserves. For the time being, in our model we assume that all the actions require no fees and that there are no transaction fees.

3.1. Basic definitions

We assume a set $\mathbb{T} = \mathbb{T}_0 \uplus \mathbb{T}_m$ of *tokens* (ranged over by τ, τ') defined as the disjoint union of the set of *atomic token types* $\mathbb{T}_0$ and *minted token types* $\mathbb{T}_m$. The first set includes native cryptocurrencies and application-specific tokens, whereas the second set includes tokens denoting claims that a user is acting as a liquidity provider. Minted tokens are represented by unordered pairs of distinct atomic tokens: if τ_0 and τ_1 are different atomic token types, $\{\tau_0, \tau_1\}$ is the corresponding minted token type meaning that the user has locked in an AMM a certain amount of tokens τ_0 and τ_1. In general, we denote with r, r' real numbers ($\mathbb{R}$), and, following the standard typing notation, we write $r\colon \tau$ to denote r units of (atomic or minted) tokens of type $\tau \in \mathbb{T}$.

We also assume a set of *users* $\mathbb{A}$ ranged over by A, A'. The *wallet* of a user A is represented as $\mathsf{A}[\sigma_\mathsf{A}]$, where $\sigma_\mathsf{A} \in \mathbb{T} \mapsto \mathbb{R}$ is a map storing A token supplies: $\sigma_\mathsf{A}(\tau)$ is the amount of token τ owned by A; when $\sigma_\mathsf{A}(\tau) = 0$ A owns no tokens of type τ. Given a wallet σ, a positive real number v, a token type τ, and an operation $\circ \in \{+, -\}$, we define operation $\sigma \circ v : \tau$ to increase or decrease the amount of token type τ as follows:

$$\sigma \circ v : \tau = \begin{cases} \sigma\{\tau \mapsto \sigma(\tau) \circ v\} & \text{if} \quad \tau \in \operatorname{dom} \sigma \text{ and } \sigma(\tau) \circ v \in \mathbb{R} \\ \sigma\{\tau \mapsto v\} & \text{if} \quad \tau \notin \operatorname{dom} \sigma \text{ and } \circ = + \end{cases}$$

where $\sigma\{\tau \mapsto v\}$ means that we update the entry of the map σ for the token type τ to the value v leaving the other entries unchanged.

Following Bartoletti *et al.* [7], we model an AMM as an unordered pair $r_0 : \tau_0, r_1 : \tau_1$, where $r_0 : \tau_0$ and $r_1 : \tau_1$ represent the reserves of tokens $\tau_0, \tau_1 \in \mathbb{T}$, with $\tau_0 \neq \tau_1$.

The *states* Γ, Γ' of the protocol are finite non-empty compositions of wallets and AMMs. Formally, they are defined as follows:

$$\Gamma = \mathsf{A}_1[\sigma_{\mathsf{A}_1}] \mid \cdots \mid \mathsf{A}_n[\sigma_{\mathsf{A}_n}] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \cdots \mid \{r_w : \tau_w, r_k : \tau_k\} \tag{1}$$

We assume that the operator $\mid$ is commutative and associative, and we use the notation $C \mid \Gamma$ when we want to highlight some components C (wallets and/or AMMs) of the state. Moreover, for simplicity, we assume the following conditions hold: for all $i \neq j$ (*i*) each user has a single wallet ($\mathsf{A}_i \neq \mathsf{A}_j$); (*ii*) distinct AMMs cannot hold exactly the same token types $\{r_w : \tau_w, r_{w'} : \tau_{w'}\} \neq \{r_k : \tau_k, r_{k'} : \tau_{k'}\}$.

We denote with $\Gamma_{\mathsf{A}}(\tau)$ the amount of token types τ owned by user A in the state Γ, i.e., $\sigma_{\mathsf{A}}(\tau)$ where the σ_{A} is the wallet of A in Γ. We define the *supply* of a token type τ in a state Γ, written $S_\Gamma\tau$, as the sum of the balances of τ in all the wallets and the AMMs in Γ. Formally, given a state Γ and a token type τ, we define $S_\Gamma\tau$ inductively on the structure of Γ as follows:

$$S_{\mathsf{A}[\sigma]}\tau = \sigma_{\mathsf{A}}(\tau) \quad S_{\{r_0:\tau_0,r_1:\tau_1\}}\tau = \begin{cases} r_i & \text{if } \tau = \tau_i \\ 0 & \text{otherwise} \end{cases}$$

$$S_{\Gamma|\Gamma'}\tau = S_\Gamma\tau + S_{\Gamma'}\tau.$$

Given a token type τ, we denote its price in the state Γ with $P_\Gamma\tau \in \mathbb{R}_0^+$. For the atomic tokens, we assume that their price is given by an external oracle. The price $P_\Gamma(\{\tau_0,\tau_1\})$ of a minted token $\{\tau_0,\tau_1\}$ depends both on the supply of $\{\tau_0,\tau_1\}$ in the users' wallets and on the reserves of τ_0 and τ_1 in the corresponding AMM. More precisely, the price of a minted token $\{\tau_0,\tau_1\}$ is the ratio between the value of the corresponding AMM and its total supply in the state Γ. The value of an AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$ is obtained by summing the price of its reserves: $r_0 \cdot P_{\tau_0} + r_1 \cdot P_{\tau_1}$. Therefore, we define the price of a minted token $\{\tau_0,\tau_1\}$ as follows:

$$P_\Gamma\{\tau_0,\tau_1\} = \frac{r_0 \cdot P_{\tau_0} + r_1 \cdot P_{\tau_1}}{S_\Gamma\{\tau_0,\tau_1\}} \quad \text{if} \quad \{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma.$$

3.2. LSM semantics

The semantics of our LSM is defined through a two-level transition system, which provides a structured and modular approach to handling the complexities of DeFi scenarios. First, we introduce the first level, which focuses on individual actions, including deposit, redeem, and relaxed swaps. Then, we illustrate the second level that implements our LSM, coordinating the interactions between users and AMMs and managing the pending queue of actions.

3.2.1. First Level Semantics

Here, we concentrate on the semantics of single actions modeling the interaction between users and AMMs as transitions between states. A transition $\Gamma \xrightarrow{s} \Gamma'$ represents the evolution of the state Γ into Γ' upon the execution of the transaction s by some user. The possible transactions that represent the labels of our LTS are:

- $\mathsf{A}\colon \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$ meaning that user A deposits v_0 units of token τ_0 and v_1 units of token of τ_1 in the AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$, receiving in return a certain amount of the minted token $\{\tau_0,\tau_1\}$; note that the AMM is created if it does not already exists;
- $\mathsf{A}\colon \mathsf{r\text{-}swap}(v_0 : \tau_0, v_1^* : \tau_1)$ meaning that user A transfers v_0 units of token τ_0 to AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$, possibly without having any up-front collateral, saying that she wants to receive at least v_1^* units of token τ_1 in return;
- $\mathsf{A}\colon \mathsf{rdm}(v : \tau)$ meaning a user A redeems v units of minted token $\tau = \{\tau_0,\tau_1\}$ from the AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$, receiving in return units of the atomic tokens τ_0 and τ_1.

Below, we present the semantic rules governing the state transitions for these actions.

The semantics of the deposit action $\mathsf{A}\colon \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$ is given by two rules depending on whether the action will result in the creation of a new AMM or in the increase of the liquidity of an already-existent AMM. In the first case, the following rule applies:

[Deposit0]
$$\frac{\sigma(\tau_i) \geq v_i > 0 \ (i \in \{0,1\}) \quad S_\Gamma\{\tau_0,\tau_1\} = 0}{\mathsf{A}[\sigma]|\,\Gamma \xrightarrow{\mathsf{A}\colon \mathsf{dep}(v_0:\tau_0,v_1:\tau_1)} \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v_0 : \{\tau_0,\tau_1\}]|\,\{v_0 : \tau_0, v_1 : \tau_1\}|\,\Gamma}$$

The first premise requires user A to have the necessary amount of tokens in her wallet. The second premise $S_\Gamma\{\tau_0,\tau_1\} = 0$ implies (*i*) that τ_0 and τ_1 are distinct atomic tokens, since otherwise $\{\tau_0,\tau_1\}$ would not be a minted token; and (*ii*) that in Γ there is no AMM for the token pair τ_0,τ_1. In return, A receives v_0 units of the minted new token type $\{\tau_0,\tau_1\}$ which user A can use to redeem her investment. Note that, as done in [7], we do not specify the exact number of received units because it is not critical, so any v_0 would be equally valid.

When the user provides liquidity to an existent AMM, the following rule applies:

[Deposit1]
$$\frac{\sigma(\tau_i) \geq v_i > 0 \ (i \in \{0,1\}) \quad r_0 v_1 = r_1 v_0 \quad v = \dfrac{v_0}{r_0} \cdot S_\Gamma\{\tau_0,\tau_1\}}{\Gamma = \mathsf{A}[\sigma]|\,\{r_0 : \tau_0, r_1 : \tau_1\}|\,\Gamma' \xrightarrow{\mathsf{A}\colon \mathsf{dep}(v_0:\tau_0,v_1:\tau_1)} \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v : \{\tau_0,\tau_1\}]|\,\{r_0 + v_0 : \tau_0, r_1 + v_1 : \tau_1\}|\,\Gamma'}$$

The first premise requires user A to have enough funds to deposit; the second one requires that the ratio of tokens in the reserve of the AMM is maintained; as done in [7], the last premise says that user A receives the amount of the minted token v that is proportional to the fraction of the provided amount of token τ_0 respect to the quantity in the AMM and total supply $S_\Gamma\{\tau_0,\tau_1\}$ of the minted token in the state Γ After the action, we reach a new state where the balances in the wallet of A and the reserves of the AMM are updated.

The semantics of a swap action $\mathsf{A}\colon \mathsf{r\text{-}swap}(v_0 : \tau_0, v_1^* : \tau_1)$ allows user A to exchange v_0 amount of token τ_0 from her wallet to the AMM and obtain back at least v_1^* amount of token τ_1. Formally, the behavior of this action is given by the following rule:

[RelaxedSwap]
$$\frac{v_1 = \dfrac{r_1 \cdot v_0}{r_0 + v_0} \quad 0 \leq v_1^* \leq v_1}{\mathsf{A}[\sigma]|\,\{r_0 : \tau_0, r_1 : \tau_1\}|\,\Gamma \xrightarrow{\mathsf{A}\colon \mathsf{r\text{-}swap}(v_0:\tau_0,v_1^*:\tau_1)} \mathsf{A}[\sigma - v_0 : \tau_0 + v_1 : \tau_1]|\,\{r_0 + v_0 : \tau_0, r_1 - v_1 : \tau_1\}|\,\Gamma}$$

The premise of the rule requires that the actual amount v_1 of received units of τ_1, calculated based on the exchange rate in the current state, must satisfy the *constant-product invariant*, which is widely implemented in protocols such as Uniswap [10], SushiSwap [11], and Curve [12]:

$$r_0 \cdot r_1 = (r_0 + v_0) \cdot (r_1 - v_1)$$

Moreover, compliance with this rule ensures the preservation of the reserve ratio in the AMM. If the second ($v_1 = \frac{r_1 \cdot v_0}{r_0 + v_0}$) and third premises ($0 \leq v_1^* \leq v_1$) are not satisfied, the action cannot be performed, as they are necessary conditions for respecting the invariant.

This action results in the update of A's wallet and the AMM's reserve. Note that the rule does not impose any liquidity constraints on the wallet of user A, so it can be triggered also when A owns less than v_0 units of tokens τ_0. In this case, the resulting wallet will be negative. Negative wallets are managed by the second level of our semantics.

Users can redeem units of a minted token $\{\tau_0, \tau_1\}$ for units of the atomic tokens τ_0 and τ_1. Each unit of $\{\tau_0, \tau_1\}$ can be redeemed for equal fractions of τ_0 and τ_1 remaining in the AMM. The semantics of the redeem action is given by the rule below:

[Redeem]

$$\frac{\sigma\{\tau_0, \tau_1\} \geq v > 0 \quad v_0 = v \cdot \frac{r_0}{S_\Gamma\{\tau_0, \tau_1\}} \quad v_1 = v \cdot \frac{r_1}{S_\Gamma\{\tau_0, \tau_1\}}}{\Gamma = \mathsf{A}[\sigma] \,|\, \{r_0 : \tau_0, r_1 : \tau_1\} \,|\, \Gamma' \xrightarrow{\mathsf{A}:\, \mathsf{rdm}(v:\tau_0,\tau_1)} \mathsf{A}[\sigma + v_0 : \tau_0 + v_1 : \tau_1 - v : \{\tau_0, \tau_1\}] \,|\, \{r_0 - v_0 : \tau_0, r_1 - v_1 : \tau_1\} \,|\, \Gamma'}$$

The first premise requires user A to have the necessary amount of minted tokens in her wallet, while the other ones specify the amount of tokens to redeem from the AMM: these amounts are proportional to the quantity available in the reserve with respect to the total supply of the minted token. The resulting state is obtained by updating the wallet of A and the AMM reserves.

Finally, we introduce some notation. We write $\Gamma \xrightarrow{\lambda} \Gamma'$ to denote the sequence of transactions $\lambda = s_1 \cdots s_n$ with intermediate states of the form $\Gamma_1, \ldots, \Gamma_{n-1}$ such that $\Gamma \xrightarrow{s_1} \Gamma_1 \xrightarrow{s_2} \cdots \xrightarrow{s_{n-1}} \Gamma_{n-1} \xrightarrow{s_n} \Gamma'$. We say that a state Γ is *reachable* if there exist some initial states Γ_0, which only contain wallets with atomic tokens and some λ such that $\Gamma_0 \xrightarrow{\lambda} \Gamma$. Hereafter, all the states mentioned in our results are implicitly assumed to be reachable. Moreover, we assume that when λ is the empty sequence ϵ, the execution results in an unchanged state, i.e., $\Gamma \xrightarrow{\epsilon} \Gamma$. A *symbolic state* Γ is a state in which the amount of tokens in wallets and AMM reserves are described by arithmetic formulas over a set of variables and numbers. A *symbolic transaction* is a transaction between symbolic states where the amount of tokens in the actions are represented by symbolic formulas rather than concrete values. We call a sequence ρ of such symbolic transactions a *pattern*.

Below, we report two properties of our semantics that are the counterparts of the properties of Bartoletti *et al.* [7]. The first theorem states that our semantics is deterministic, and the second one says that the AMM reserves never become negative. Their proofs follow the same lines as those of Bartoletti *et al.* [7].

Theorem 1 (Deterministic semantics [7])**.** *If* $\Gamma \xrightarrow{s} \Gamma'$ *and* $\Gamma \xrightarrow{s} \Gamma''$*, then* $\Gamma' = \Gamma''$.

Theorem 2 (Non deplention of AMM reserves [7])**.** *For all states* Γ*, if* $\{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma$ *then:*

1. $r_i > 0$*, for* $i \in \{0, 1\}$*;*
2. $S_\Gamma\{\tau_0, \tau_1\} > 0$.

3.2.2. Second Level Semantics

The second level of the transition system implements the overall mechanism of our protocol by coordinating interactions between individual actions, ensuring smooth operation and adherence to specified semantics, and upholding broader system objectives and constraints. The LTS states are called *configurations*, each defined as a 3-tuple $\langle \Gamma, \Gamma', \Lambda_\ell \rangle$, where Γ is a state of the form (1), Γ' represents the last simulated state (as discussed below), and Λ_ℓ is a transaction queue with maximum size ℓ. In cases where the size ℓ is not central to the discussion, we will use the abbreviated form Λ to simplify notation and avoid unnecessary complexity in the presentation. This queue stores the actions that result in overdrafts on users' balances that cannot be immediately executed. Formally, it is a term obtained by the following grammar:

$$\Lambda := \emptyset \mid \Lambda \circ s$$

where $\emptyset$ denotes an empty queue with no pending actions, and $\Lambda \circ s$ a queue where Λ is the head and the action s is the tail of the queue. In the semantic rules below, we use $\Lambda \circ s$ to represent when an action s is added to the queue Λ. Moreover, we write $|\Lambda|$ to indicate the size of the queue Λ. For instance, $\emptyset$ is an empty queue with no pending action; while the term $\emptyset \circ s_1$ denotes a queue with a single action s_1; and the term $\emptyset \circ s_1 \circ s_2$ denotes a queue with two actions: the head is s_1, and the tail is s_2. Given a state Γ we define $\langle \Gamma, \Gamma, \emptyset \rangle$ as an *initial configuration*.

The transitions of our LTS from a configuration to another one have the form $\langle \Gamma, \Gamma', \Lambda \rangle \xrightarrow{s} \langle \Gamma'', \Gamma''', \Lambda' \rangle$ and are triggered when a user submits a transaction s, and are defined by the inference rules below.

It is convenient to introduce some notation for the semantic rules. We say that a state Γ is `green` when the token supply of each user is non-negative, formally:

$$\forall\, \mathsf{A} \in \mathbb{A} \cap \Gamma, \tau \in \mathbb{T} \,.\, \sigma_{\mathsf{A}}(\tau) \geq 0$$

where we denote with $\mathbb{A} \cap \Gamma$ the set of users occurring in the configuration Γ. While we say it is `red` otherwise, namely, when

$$\exists\, \mathsf{A} \in \mathbb{A} \cap \Gamma, \tau \in \mathbb{T} \,.\, \sigma_{\mathsf{A}}(\tau) < 0$$

When a transition is triggered by an action s in a configuration $\langle \Gamma, \Gamma', \Lambda \rangle$, we have three possible scenarios.

The first scenario arises when the execution of s in Γ' (the last simulated state) using our first level semantics produces a state Γ'' that is `green`. In this case, we apply the following rule:

[Cover]

$$\frac{\Gamma' \xrightarrow{s} \Gamma'' \qquad \Gamma'' \texttt{green}}{\langle \Gamma, \Gamma', \Lambda \rangle \xrightarrow{s} \langle \Gamma'', \Gamma'', \emptyset \rangle}$$

The [Cover] rule performs all the pending actions in Λ (if any), and the resulting configuration consists of the `green` state, Γ'' (in the first two components), and the emptied queue. Note that our LSM aims to maximize the number of settled actions performed by users. Moreover, our first level semantics imposes no constraints on user balances, thus, some of them could be negative in the resulting state Γ'. Differently, we maintain the checks

to impose non-negativity constraints on the AMM reserve (see [RelaxedSwap] rule).

The second scenario happens when the simulation through the first level transition system of s in Γ' reaches a red state Γ'', and the length of Λ is less than the queue capacity ℓ (a parameter of our mechanism). In this case, we apply the following rule:

[Overdraft]
$$\frac{\Gamma' \xrightarrow{s} \Gamma'' \qquad \Gamma''\texttt{red} \qquad |\Lambda_\ell| < \ell \qquad \Lambda'_\ell = \Lambda_\ell \circ s}{\langle \Gamma, \Gamma', \Lambda_\ell \rangle \xrightarrow{s} \langle \Gamma, \Gamma'', \Lambda'_\ell \rangle}$$

The [Overdraft] rule enqueues s in Λ, thus, the resulting configuration consists of the current state Γ, the red state Γ'', and Λ' that is Λ extended with the incoming action s.

The last scenario occurs when the simulation through the first level transition system of s leads to a red state Γ'', and the length of Λ equals ℓ. In this case, we use the netting procedure (denoted by the function *net* in the following rule) to perform the *settlement* from the state Γ and the actions of Λ plus s.

The netting procedure identifies and returns a (possibly empty) sequence of feasible actions Λ'. Such sequence is a subset of the input sequence Λ. The obtained subsequence Λ' is executed to obtain the resulting state Γ^*. All the other actions of Λ that are not selected by the procedure are discarded.

The resulting configuration is a 3-tuple with the state Γ^* (for the first two components) and the empty queue for the last one (the queued actions were carried out or discarded). Formally, we apply the following rule:

[Netting]
$$\frac{\Gamma' \xrightarrow{s} \Gamma'' \quad \Gamma''\texttt{red} \quad |\Lambda_\ell| = \ell \quad \Lambda'_\ell = net(\Gamma, \Lambda_\ell \circ s) \quad \Gamma \xrightarrow{\Lambda'_\ell} \Gamma^*}{\langle \Gamma, \Gamma', \Lambda_\ell \rangle \xrightarrow{s} \langle \Gamma^*, \Gamma^*, \emptyset \rangle}$$

It is worth remarking that the mechanism provided in this section is agnostic with respect to the netting procedure that we invoke in a black-box manner. For instance, a simple procedure could be to discard all enqueued transactions. The next section introduces a more meaningful procedure. Also, we remark that when we apply the above rule, the queue Λ contains a prefix of actions leading to a red state that is not balanced by other actions in the queue. Otherwise, the [Cover] rule would be applicable. The rule invokes the netting procedure to execute a subset of actions that allows the AMM to progress.

Finally, note that our configurations and semantic rules could be written without Γ' (the 2nd component of the configuration recording the simulated/speculative final state) and replacing the premise $\Gamma' \xrightarrow{s} \Gamma''$ with $\Gamma \xrightarrow{\Lambda \circ s} \Gamma''$. This approach requires recomputing the final state every time a new action is performed. These re-computations are not efficient in an implementation where it is more convenient to store the intermediate state. We decided to follow this approach to make our formalization coherent with a possible implementation.

Theorem 3 (Deterministic semantics)**.** *If* $\langle \Gamma, \Gamma', \Lambda \rangle \xrightarrow{s} \langle \Gamma'', \Gamma''', \Lambda' \rangle$ *and* $\langle \Gamma, \Gamma', \Lambda \rangle \xrightarrow{s} \langle \Gamma_2, \Gamma_3, \Lambda'' \rangle$*, then* $\langle \Gamma'', \Gamma''', \Lambda' \rangle = \langle \Gamma_2, \Gamma_3, \Lambda'' \rangle$.

Proof. Assume by contradiction that $\langle \Gamma, \Gamma', \Lambda \rangle \xrightarrow{s} \langle \Gamma'', \Gamma''', \Lambda' \rangle$ and $\langle \Gamma, \Gamma', \Lambda \rangle \xrightarrow{s} \langle \Gamma_2, \Gamma'_3, \Lambda'' \rangle$, but $\langle \Gamma'', \Gamma''', \Lambda' \rangle \neq \langle \Gamma_2, \Gamma_3, \Lambda'' \rangle$. The proof proceeds by cases on the last semantic rule applied. Let's consider the case for the [Cover] rule. By the premise of the rule we have that $\Gamma' \xrightarrow{s} \Gamma''$ and Γ'' is green, and the resulting configuration is $\langle \Gamma'', \Gamma'', \emptyset \rangle$. Moreover, by Lemma 1 we have that $\Gamma'' = \Gamma_2$, thus, $\langle \Gamma'', \Gamma'', \emptyset \rangle = \langle \Gamma_2, \Gamma_2, \emptyset \rangle$, and we have a contradiction. The cases for [Overdraft] and [Netting] rules are similar. In the case for [Netting], we also use the assumption that the algorithm *net* is deterministic. □

3.3. Formal Properties

Our mechanism enables liquidity-saving behavior that is not possible with ordinary AMMs. An example is the simultaneous change of tokens through multiple AMMs that we presented in Section 2. In this subsection, we present some theorems that characterize those scenarios where our LSM allows users to achieve some gains compared to the way used by current DeFi protocols. In Section 5, we will present concrete examples and specific instances corresponding to these scenarios. Those examples will clarify and instantiate our theoretical results, providing the reader with a deeper understanding of their practical implications.

In the theorems below we use the notion of patterns introduced above and refer to the execution process of the standard mechanism using the following transitions:

$$\Gamma \xrightarrow{s}\!\!\!\!\rightarrow \Gamma'$$

The semantics of the standard mechanism can be easily obtained from the semantics we presented in Section 3.2.1 by replacing the [RelaxedSwap] with two rules.[1] Note that in both of the following rules, the second and third premises, namely, $v_1 = \frac{r_1 \cdot v_0}{r_0 + v_0}$ and $0 \leq v_1^* \leq v_1$, must hold in accordance with the constant-product invariant introduced earlier in Section 3.2.1. If these conditions are not satisfied, the action cannot be performed.

[Swap_1]
$$\frac{\sigma\tau_0 \geq \mathbf{v_0} \quad v_1 = \dfrac{r_1 \cdot v_0}{r_0 + v_0} \quad 0 \leq v_1^* \leq v_1}{\mathsf{A}[\sigma]|\{r_0 : \tau_0, r_1 : \tau_1\}|\Gamma \xrightarrow{\mathsf{A}:\ \mathsf{swap}(v_0:\tau_0, v_1^*:\tau_1)} \mathsf{A}[\sigma - v_0 : \tau_0 + v_1 : \tau_1]|\{r_0 + v_0 : \tau_0, r_1 - v_1 : \tau_1\}|\Gamma}$$

Differently from [RelaxedSwap] rule, the first premise checks that the wallet has enough funds, and if this is the case, update the wallet and AMM reserves accordingly. The second rule below deals with the case where the liquidity constraints are not met:

[Swap_2]
$$\frac{\sigma\tau_0 < \mathbf{v_0} \quad v_1 = \dfrac{r_1 \cdot v_0}{r_0 + v_0} \quad 0 \leq v_1^* \leq v_1}{\mathsf{A}[\sigma]|\{r_0 : \tau_0, r_1 : \tau_1\}|\Gamma \xrightarrow{\mathsf{A}:\ \mathsf{swap}(v_0:\tau_0, v_1^*:\tau_1)} \mathsf{A}[\sigma]|\{r_0 : \tau_0, r_1 : \tau_1\}|\Gamma}$$

[1] For further details on the semantics of the swap transaction, we refer the interested reader to [7, 13].

the first premise ($\sigma\tau_0 < v_0$) checks that the amount of token type τ_0 is less than the exchanged ones, so the action is ignored and the resulting state is left unchanged.

Note that, in the theorems below, we consider the sequence of symbolic actions ρ (call it patterns) whose size is less than the queue max size ℓ, namely, $|\rho| < \ell$. Moreover, recall that we ignore the amount of transaction fees.

The following theorem characterizes those scenarios where our mechanism allows a user to successfully increase her reserve of a token type τ_0 achieving her intent, even if she initially has no tokens in her wallet. This is in contrast to the standard mechanism that, in the same scenarios, neither enables any action nor allows the user to achieve any intent.

Theorem 4 (Single user token increment)**.** *Consider a state*

$$\Gamma = \mathsf{A}[0:\tau_0, 0:\tau_1, z:\tau_2] \,|\, \{m:\tau_0, n:\tau_1\} \,|\, \{u:\tau_2, v:\tau_1\} \,|\, \widehat{\Gamma}$$

where the agent A *aims to increase the value of the token type τ_0 in her wallet; and where the two* AMM*s have the token type τ_1 in common.*

Assume that A *performs the sequence of actions $\rho = s_1\rho' s_2$ of the form:*

- $s_1 = \mathsf{A}\colon \mathsf{r\text{-}swap}(a:\tau_1, b:\tau_0)$ *such that* $a, b > 0$*;*
- ρ' *is a sequence of actions performed by other users, that do not cause overdraft, and that do not impact the reserves of the* AMM $\{u:\tau_2, v:\tau_1\}$*;*
- $s_2 = \mathsf{A}\colon \mathsf{r\text{-}swap}(c:\tau_2, d:\tau_1)$ *such that* $c, d > 0$*,* $c < z$*, and* $a \leq d$*.*

If $\langle \Gamma, \Gamma, \emptyset\rangle \xrightarrow{\rho} \langle \Gamma', \Gamma', \emptyset\rangle$ *then* $\Gamma'_{\mathsf{A}}(\tau_0) > \Gamma_{\mathsf{A}}(\tau_0)$*; while if* $\Gamma \overset{\rho}{\twoheadrightarrow} \Gamma^s$ *then* $\Gamma^s_{\mathsf{A}}(\tau_0) = \Gamma_{\mathsf{A}}(\tau_0)$*.*

Proof. By using our operational semantics we have the following sequence of transition from the initial state Γ:

$$\langle \Gamma, \Gamma, \emptyset\rangle \xrightarrow{s_1} \langle \Gamma, \Gamma''', \emptyset \circ s_1\rangle \quad \text{by [overdraft] rule} \tag{2}$$

$$\xrightarrow{\rho'} \langle \Gamma, \Gamma'', \emptyset \circ s_1 \circ \rho'\rangle \quad \text{by a repetition of [overdraft] rule} \tag{3}$$

$$\xrightarrow{s_2} \langle \Gamma', \Gamma', \emptyset\rangle \quad \text{by [cover] rule} \tag{4}$$

where

$$\Gamma''' = \mathsf{A}[b:\tau_0, -a:\tau_1, z:\tau_2] \,|\, \{m-b:\tau_0, n+a:\tau_1\} \,|\, \{u:\tau_2, v:\tau_1\} \,|\, \widehat{\Gamma}$$

$$\Gamma'' = \mathsf{A}[b:\tau_0, -a:\tau_1, z:\tau_2] \,|\, \{m':\tau_0, n':\tau_1\} \,|\, \{u:\tau_2, v:\tau_1\} \,|\, \widehat{\Gamma'}$$

$$\Gamma' = \mathsf{A}[b:\tau_0, -a+d:\tau_1, z-c:\tau_2] \,|\, \{m':\tau_0, n':\tau_1\} \,|\, \{u+c:\tau_2, v-d:\tau_1\} \,|\, \widehat{\Gamma'}$$

In the first transition (eq. 2), the execution of s_1 results in a `red` state so we apply the [overdraft] rule that enqueues s_1 in the pending action queue. Following this, the system evolves, taking into account the other actions ρ' that do not settle A's overdraft (so the negative supply of the user A in Γ'' of the token type τ_1 is the same as the previous configuration, so $\Gamma'_{\mathsf{A}}(\tau_1) = \Gamma''_{\mathsf{A}}(\tau_1)$), and the action in ρ' are inserted in the queue (eq. 3). Finally, A performs the action s_2 that covers the negative balance in her wallet, and the configuration of the system becomes `green`, carrying out all the actions in the queue (eq. 4). Since $b > 0$, we have that $b = \Gamma'_{\mathsf{A}}(\tau_0) > \Gamma_{\mathsf{A}}(\tau_0) = 0$, so A achieves her goal.

Whereas, when we use the standard mechanism from Γ we have that $\Gamma \xrightarrow{s_1} \Gamma$ because s_1 is discarded due to the violation of liquidity constraints; then, $\Gamma \xrightarrow{\rho'} \Gamma^s$ and $\Gamma^s \xrightarrow{s_2} \Gamma^{s'}$ because ρ' is performed by other users than A and does not affect the supply of the AMM $\{u:\tau_2, v:\tau_1\}$ and s_2 is executed affecting only τ_1 and τ_2, not τ_0 in the wallet of A. Therefore, $\Gamma^{s'}_{\mathsf{A}}(\tau_0) = \Gamma_{\mathsf{A}}(\tau_0)$: in this case user A does not achieve her goal. □

The following corollary of Theorem 4 illustrates how our mechanism enables a single user, who initially has no tokens in her wallet, to successfully increase her holdings. In contrast, the standard mechanism neither facilitates any action nor allows the achievement of any intent.

Corollary 4.1. *Consider a state*

$$\Gamma = \mathsf{A}[0:\tau_0, 0:\tau_1, 0:\tau_2] \,|\, \{m:\tau_0, n:\tau_1\} \,|\, \{u:\tau_2, v:\tau_1\} \,|\, \{p:\tau_2, q:\tau_0\}$$

where the agent A *aims to maximize the value in her wallet. Assume that she performs the sequence of actions $\rho = s_1 s_2 s_3$ of the form (where max queue size $\ell = 2$):*

- $s_1 = \mathsf{A}\colon \mathsf{r\text{-}swap}(a:\tau_0, b:\tau_1)$ *such that* $a, b > 0$
- $s_2 = \mathsf{A}\colon \mathsf{r\text{-}swap}(e:\tau_1, f:\tau_2)$ *such that* $e, f > 0$
- $s_3 = \mathsf{A}\colon \mathsf{r\text{-}swap}(i:\tau_2, l:\tau_0)$ *such that* $i, l > 0$

and $l > a$ *for token type* τ_0*,* $b > e$ *for* τ_1*, and* $f > i$ *for* τ_2*.*
If $\langle \Gamma, \Gamma, \emptyset\rangle \xrightarrow{\rho} \langle \Gamma', \Gamma', \emptyset\rangle$ *then* $\Gamma'_{\mathsf{A}}(\tau_0) > \Gamma_{\mathsf{A}}(\tau_0)$*,* $\Gamma'_{\mathsf{A}}(\tau_1) > \Gamma_{\mathsf{A}}(\tau_1)$*, and* $\Gamma'_{\mathsf{A}}(\tau_2) > \Gamma_{\mathsf{A}}(\tau_2)$*. Whereas if* $\Gamma \overset{\rho}{\twoheadrightarrow} \Gamma^s$ *than* $\Gamma^s_{\mathsf{A}}(\tau_0) = \Gamma_{\mathsf{A}}(\tau_0)$*,* $\Gamma^s_{\mathsf{A}}(\tau_1) = \Gamma_{\mathsf{A}}(\tau_1)$*, and* $\Gamma^s_{\mathsf{A}}(\tau_2) = \Gamma_{\mathsf{A}}(\tau_2)$*.*

Proof. The proof directly follows by Theorem 4. Note that in this case, ρ is the action s_2 performed by the user A of the form $\mathsf{A}\colon \mathsf{r\text{-}swap}(e:\tau_1, f:\tau_2)$.

□

Finally, the following theorem characterizes those scenarios where our LSM mechanism allows the simultaneous exchanges of tokens between two users with different intents. In the lemma, we assume that the first user aims to increase her reserve of token type τ_0, while the second user wants to increase her reserve of token type τ_2. The lemma ensures that by using our mechanism, both users can successfully achieve their respective goals. In contrast, the lemma states that the standard mechanism does not permit the users to fulfill their intents.

Theorem 5 (Simultaneous exchange)**.** *Consider a state*

$$\Gamma = \mathsf{A}[0:\tau_0, 0:\tau_1, z:\tau_2] \mid \mathsf{B}[u:\tau_0, 0:\tau_1, 0:\tau_2] \mid$$

$$\{m:\tau_0, n:\tau_1\} \mid \{p:\tau_1, r:\tau_2\}$$

where agent A *aims to increase the amount of token* τ_0 *in her wallet and* B *wants to increase the amount of token* τ_2*.*

Assume they perform the sequence of actions $\rho = s_1 s_2 s_3 s_4$ *of the form (where max queue size* $\ell = 3$*):*

- $s_1 = \mathsf{A}$: r-swap$(a:\tau_1, b:\tau_0)$ *such that* $a, b > 0$
- $s_2 = \mathsf{B}$: r-swap$(c:\tau_1, d:\tau_2)$ *such that* $c, d > 0$
- $s_3 = \mathsf{A}$: r-swap$(e:\tau_2, f:\tau_1)$ *such that* $e, f > 0$*,* $z \geq e$ *and* $f \geq a$
- $s_4 = \mathsf{B}$: r-swap$(g:\tau_0, h:\tau_1)$ *such that* $g, h > 0$*,* $u \geq g$ *and* $h \geq c$

If $\langle \Gamma, \Gamma, \emptyset\rangle \xrightarrow{\rho} \langle \Gamma', \Gamma', \emptyset\rangle$ *then* $\Gamma'_{\mathsf{A}}(\tau_0) > \Gamma_{\mathsf{A}}(\tau_0)$ *and* $\Gamma'_{\mathsf{B}}(\tau_2) > \Gamma_{\mathsf{B}}(\tau_2)$*; while if* $\Gamma \overset{\rho}{\twoheadrightarrow} \Gamma^s$ *then* $\Gamma^s_{\mathsf{A}}(\tau_0) = \Gamma_{\mathsf{A}}(\tau_0)$ *and* $\Gamma^s_{\mathsf{B}}(\tau_2) = \Gamma_{\mathsf{B}}(\tau_2)$*.*

Proof. By using our operational semantics we have the following sequence of transition from the initial state Γ:

$$\langle \Gamma, \Gamma, \emptyset\rangle \xrightarrow{s_1} \langle \Gamma, \Gamma'''', \emptyset \circ s_1\rangle \quad \text{by [overdraft] rule} \tag{5}$$

$$\xrightarrow{s_2} \langle \Gamma, \Gamma''', \emptyset \circ s_1 \circ s_2\rangle \quad \text{by [overdraft] rule} \tag{6}$$

$$\xrightarrow{s_3} \langle \Gamma, \Gamma'', \emptyset \circ s_1 \circ s_2 \circ s_3\rangle \quad \text{by [overdraft] rule} \tag{7}$$

$$\xrightarrow{s_4} \langle \Gamma', \Gamma', \emptyset\rangle \quad \text{by [cover] rule} \tag{8}$$

where

$$\Gamma'''' = \mathsf{A}[b:\tau_0, -a:\tau_1, z:\tau_2] \mid \mathsf{B}[u:\tau_0, 0:\tau_1, 0:\tau_2] \mid$$
$$\{m-b:\tau_0, n+a:\tau_1\} \mid \{p:\tau_1, r:\tau_2\}$$
$$\Gamma''' = \mathsf{A}[b:\tau_0, -a:\tau_1, z:\tau_2] \mid \mathsf{B}[u:\tau_0, -c:\tau_1, d:\tau_2] \mid$$
$$\{m-b:\tau_0, n+a:\tau_1\} \mid \{p+c:\tau_1, r-d:\tau_2\}$$
$$\Gamma'' = \mathsf{A}[b:\tau_0, -a+f:\tau_1, z-e:\tau_2] \mid \mathsf{B}[u:\tau_0, -c:\tau_1, d:\tau_2] \mid$$
$$\{m-b:\tau_0, n+a:\tau_1\} \mid \{u+c+e:\tau_2, v-d-f:\tau_1\}$$
$$\Gamma' = \mathsf{A}[b:\tau_0, -a+f:\tau_1, z-e:\tau_2] \mid$$
$$\mathsf{B}[u-g:\tau_0, -c+h:\tau_1, d:\tau_2] \mid \{m+b+g:\tau_0, n-a-h:\tau_1\} \mid$$
$$\{u+c:\tau_2, v-d:\tau_1\}$$

In the first transition (eq. 5), the execution of s_1 results in a `red` state, so we apply the [overdraft] rule, which enqueues s_1 in the queue. Similarly, the execution of s_2 leads to a `red` state Γ''', and thus we again apply the [overdraft] rule, enqueuing s_2 (eq. 6). Following this, the system performs the action s_3, which makes A's wallet balance positive without changing the overall state, which remains `red`. Whereas, the negative balance of user B in Γ'' for token type τ_1 remains unchanged, i.e., $\Gamma''''_{\mathsf{B}}(\tau_1) = \Gamma''_{\mathsf{B}}(\tau_1)$. So, the action s_3 is also inserted into the queue (eq. 7). Finally, B performs the action s_4 that covers her negative balance, resulting in a `green` state and executing all actions in the queue (eq. 8). Since $b > 0$, we have $b = \Gamma'_{\mathsf{A}}(\tau_0) > \Gamma_{\mathsf{A}}(\tau_0) = 0$, and since $d > 0$, we have $d = \Gamma'_{\mathsf{B}}(\tau_0) > \Gamma_{\mathsf{B}}(\tau_0) = 0$, thus A and B achieve their goals.

In contrast, using the standard mechanism from Γ, we have $\Gamma \overset{s_1}{\twoheadrightarrow} \Gamma$ because s_1 is discarded due to liquidity constraint violations. Similarly, $\Gamma \overset{s_2}{\twoheadrightarrow} \Gamma$ because s_2 is discarded for the same reason. Then, $\Gamma \overset{s_3}{\twoheadrightarrow} \Gamma^s$ and $\Gamma^s \overset{s_4}{\twoheadrightarrow} \Gamma^{s'}$ because s_3 and s_4 can be executed by the standard mechanism but do not help the users achieve their goals (s_3 increases $\Gamma^s_{\mathsf{A}}(\tau_1)$ while s_4 increases $\Gamma^{s'}_{\mathsf{B}}(\tau_1)$, respectively). Therefore, $\Gamma^s_{\mathsf{A}}(\tau_0) = \Gamma_{\mathsf{A}}(\tau_0)$ and $\Gamma^{s'}_{\mathsf{B}}(\tau_2) = \Gamma_{\mathsf{B}}(\tau_2)$, meaning users A and B do not achieve their goals. □

4. The netting procedure

Here, we provide an algorithm for the netting procedure *net* invoked in the [Netting] rule of Section 3, and then we study some of its properties.

4.1. The algorithm

Our netting problem is defined as follows: given a sequence of transaction Λ and a starting state Γ, we aim to find a subsequence of transaction of Λ, call it Λ', that maximizes a given objective function f such that the execution of the transactions in Λ' from Γ lead to a `green` state Γ^*. For example, the objective function f may maximize the size of Λ', the number of involved users, or the amount of given token types, or holdings in wallets. Recall also in a `green` state the *liquidity constraints* are satisfied, namely, there is no overdraft in users' wallets. Formally, we define the following optimization problem:

$$\max\ f(\Lambda', \Gamma^*) \tag{9}$$

subject to

$$\Gamma \xrightarrow{\Lambda'} \Gamma^* \tag{10}$$

$$\Gamma^* \texttt{ green} \tag{11}$$

Hereafter, we consider that the objective function f aims to maximize the size of Λ', namely, $f(\Lambda', \_) = |\Lambda'|$. Since we need an algorithm that can run on a smart contract (thus incurring affordable gas expenses), we adopt a heuristic approach that trades optimality for efficiency that avoids enumerations and attempts to maximize the number of transactions performed. To this purpose, we propose *Algorithm 1* that runs in polynomial time (quadratic in the size of the queue, at worst – see below).

Intuitively, *Algorithm 1* executes the transactions using our first level semantics of Section 3 that ignores the liquidity constraints. It starts by initializing Λ_r with Λ, the initial queue of pending actions, and Γ^0 with Γ, the initial state. It then starts a loop (line 2), where it simulates the execution of the actions until either the final state Γ^l is not `red` or the queue Λ_r becomes empty, namely, until there are still actions to be processed and overdrafts in the system. During each iteration, we select from Λ_r the first action s_i that makes the balance of some user A

Algorithm 1 Heuristic netting procedure implementation

Input: $\Lambda = [s_1, s_2, \ldots, s_l]$, Γ
Output: Λ_r
Initialization: $\Lambda_r \leftarrow \Lambda$, $\Gamma^0 \leftarrow \Gamma$

1: $\Gamma^0 \xrightarrow{\Lambda_r} \Gamma^l$ ▷ *Starting execution*
2: **while** Γ^l `red` $\wedge\ \Lambda_r \neq \emptyset$ **do** ▷ *Final state has overdraft*
3: select *min* i s.t. $s_i \in \Lambda_r$ where $\sigma_{\mathsf{A}}(\tau) < 0$ ▷ *Select action that overdraft*
4: $\Lambda_r \leftarrow \Lambda_r - s_i$ ▷ *Update actions queue*
5: $\Gamma^i \leftarrow \Gamma^{i-1}$ ▷ *Update last* `green` *state*
6: $\Gamma^i \xrightarrow{s_{i+1}} \Gamma^{i+1} \xrightarrow{s_{i+2}} \Gamma^{i+2} \ldots \xrightarrow{s_f} \Gamma^f$ ▷ *Run all remaining actions in the queue discarding all the stuck actions*
7: $\Gamma^l \leftarrow \Gamma^f$ ▷ *Update the variable* Γ^l
8: **end while**
9: **return** Λ_r

negative, i.e., the execution of s_i leads Γ^l to `red` state (line 3). Then, the algorithm removes s_i from Λ_r (line 4) and updates the resulting state Γ^i (line 5) by reverting s_i. We achieve that by simply considering the previous state Γ^{i-1} that is the last `green` state. After we recover to the last `green` state, the execution is run again but from Γ^i using the actions following s_i until all balances are non-negative. Moreover, all the actions that cannot be executed due to the exchange-rate or liquidity constraints being unmet are discarded. As a result of this last execution and filtering, we obtain the `green` state Γ^f (line 6). Note that, the actions were performed from scratch, so the exchanged values may change according to modifications in the rate. These steps are iterated until we obtain a `green` state or an empty Λ_r. When this happens, the algorithm returns Λ_r. Our algorithm operates on a transaction queue that represents submissions within a specific time frame. As such, its decisions are inherently local to that window. This means that, although effective within the current snapshot, the algorithm might discard a transaction that appears to cause an overdraft now but could be covered by future incoming funds, thus missing a potentially more convenient trade over a longer time horizon.

Note that our algorithm does not strictly depend on the policy to select the action to be removed. Indeed, one could easily accommodate new policies by modifying line 3. For example, we can replace it as follows. We first identify the user A^* and the token type τ^* with the largest overdraft in the final state Γ^l, namely:

$$\tau^*, \mathsf{A}^* \leftarrow \arg\min_{\tau,\mathsf{A}} \Gamma^l_{\mathsf{A}}(\tau)$$

Then, we remove from the queue Λ_r the action that causes the largest overdraft of the token type τ^* in user A^*'s wallet, namely:

$$i \leftarrow \arg\min_i\ \Gamma^i_{\mathsf{A}^*}(\tau^*)\ s.t.\ s_i \in \Lambda_r \tag{12}$$

where $s_i = \mathsf{A}^* : \mathsf{r\text{-}swap}(\_ : \tau^*, \ldots)$. In Section 5, Example 6 provides a concrete use case of the policy described above.

4.2. Formal properties of the netting procedure

Here, we study some formal properties of our netting algorithm. First, we prove that our algorithm terminates and its complexity is quadratic on the length of the input queue in the worst case scenario.

Theorem 6 (Termination and Complexity)**.** *Algorithm 1 always terminates, and its complexity is $O(|\Lambda|^2)$ where Λ is the input queue.*

Proof. Remember from our assumption that the length of Λ is at most l. We prove termination by showing that the number of iterations of the loop is finite. In each iteration of the loop, at least one action s_i causing an overdraft is removed from Λ_r. Since Λ_r initially contains l actions, in the worst-case scenario, each iteration removes at least one action, so the length of Λ decreases at each iteration. Therefore, after at most l iterations, the queue Λ_r will be empty, and the algorithm will terminate.

We now consider its complexity. We know that each iteration involves finding inside the queue Λ the first action s_i causing an overdraft. This search takes $O(|\Lambda|)$ time. Removing the action s_i and re-executing the remaining actions $s_{i+1} \ldots s_{|\Lambda|}$ takes $O(|\Lambda|)$ in the worst-case. Since there can be at most $O(|\Lambda|)$ iterations, the total complexity is $O(|\Lambda|^2)$, namely, the algorithm is quadratic in the length of the queue Λ. □

The following theorem guarantees that when the algorithm removes a swap action from the queue that a deposit action in the queue depends on, this last action is also removed in turn.

Theorem 7 (Dependent actions)**.** *Consider a state*

$$\Gamma = \mathsf{A}[0 : \tau_0, 0 : \tau_1, z : \tau_2] \mid \{m : \tau_0, n : \tau_1\} \mid \{u : \tau_1, v : \tau_2\}$$

Assume the user A *performs the sequence of actions $\Lambda = p \circ s_1 \circ s_2 \circ p'$ where $|\Lambda| = \ell$. The sequence of actions, p does not concern τ_0 and τ_1, s_1 and s_2 have the following form:*

- $s_1 = \mathsf{A}\colon \mathsf{r\text{-}swap}(a : \tau_0, b : \tau_1)$ *such that $a, b > 0$ and $S_{\Gamma^1}\tau_0 < 0$;*
- $s_2 = \mathsf{A}\colon \mathsf{dep}(b : \tau_1, c : \tau_2)$ *such that $c > 0$ and $S_{\Gamma^2}\tau_1 = b$.*

and p' is a sequence of actions that may depend on s_2. If $\Gamma \xrightarrow{\Lambda} \Gamma'$ with Γ' `red` *and if $\Gamma \xrightarrow{p} \Gamma^1$ and $\Gamma \xrightarrow{p \circ s_1} \Gamma^2$ with $\Gamma^2_{\mathsf{A}}(\tau_0) < 0$ then $net(\Gamma, \Lambda) = \Lambda'$ and $s_1, s_2 \notin \Lambda'$.*

Proof. The hypothesis $\Gamma \xrightarrow{p} \Gamma^1$ and $\Gamma \xrightarrow{p \circ s_1} \Gamma^2$ with $\Gamma^2_{\mathsf{A}}(\tau_0) < 0$ means that s_1 does not cover any action in p but causes an overdraft concerning τ_0 for user A. This means that at a certain point during the execution of Algorithm 1, the action s_1 will be identified as the action causing the overdraft and will be removed from the queue. When Algorithm 1 reaches line 6, it identifies s_2 as a deposit that cannot be performed by user A because of lacking of funds ($\Gamma^2_{\mathsf{A}}(\tau_1) < b$), and removes s_2 as well. Therefore, the queue returned at the end of the execution of Algorithm 1 contains neither s_1 nor s_2.

□

Similarly, we can prove that, if in p' there is a redeem action of the form $\mathsf{A}\colon \mathsf{rdm}(b : \tau_1, c : \tau_2)$ (that is triggered by the execution of s_1), the Algorithm 1 discards this action too.

5. Implementation

In this section, we first discuss the implementation of our simulator and then present various application scenarios that highlight the advantages of our netting-based approach compared to the approach method used by standard AMM protocols.

5.1. Protocol simulator

We have developed a simulator written in Haskell that implements the semantics of our LSM mechanism and our netting algorithm outlined in Sections 3 and 4. The simulator includes support for the liquidity provider transactions of deposit and redeem, as well as the r-swap transaction used to exchange tokens on the AMM. Internally, it uses the two-level semantic structure described earlier, applying either the [Cover], [Overdraft], or [Netting] rule after each transaction. It also implements Algorithm 1 with the policy to select the action to be removed described in the paper and with the variant introduced by equation (12). The simulator can be used to check the execution traces of the examples presented in the next section. It is open source and available online [8].

5.2. Application scenarios

In this section, we present several application examples related to application scenarios that we formally characterized in Sections 3.3 and 4.2. We ran all the following examples through our simulator that validated them.

The first example illustrates the scenario considered in Theorem 5.

Example 1 (Negative balance covered)**.** *Consider the state*

$$\Gamma = \mathsf{A}[0:\tau_0, 0:\tau_1, 0:\tau_2] \mid \{8:\tau_0, 18:\tau_1\} \mid \{8:\tau_1, 18:\tau_2\} \mid \{8:\tau_2, 18:\tau_0\}$$

Assuming a 1-to-1 exchange rate, three arbitrage opportunities arise but are only available to users with sufficient funds. Suppose agent A *wants to perform the following actions* $\rho = s_1 s_2 s_3$ *increasing her wallet (where max queue size* $\ell = 2$*):*

- $s_1 = \mathsf{A}\text{: r-swap}(4:\tau_0, 6:\tau_1)$
- $s_2 = \mathsf{A}\text{: r-swap}(4:\tau_1, 6:\tau_2)$
- $s_3 = \mathsf{A}\text{: r-swap}(4:\tau_2, 6:\tau_0)$

Using the standard mechanism, all the actions are discarded because A *does not have enough balance, so the system will no evolve, namely* $\Gamma \overset{\rho}{\nrightarrow} \Gamma$*. Since* $\Gamma_{\mathsf{A}}(\tau_0), \Gamma_{\mathsf{A}}(\tau_1)$*, and* $\Gamma_{\mathsf{A}}(\tau_2)$ *does not increase, agent* A *does not achieve her goal. Figure 2 shows a flow diagram of the actions performed adopting our mechanism. By our operational semantics, we have the following sequence of transitions:*

$$\langle \Gamma, \Gamma, \emptyset \rangle \xrightarrow{s_1} \langle \Gamma, \Gamma''', \emptyset \circ s_1 \rangle \quad \textit{by } [\text{overdraft}] \textit{ rule} \tag{13}$$

$$\xrightarrow{s_2} \langle \Gamma, \Gamma'', \emptyset \circ s_1 \circ s_2 \rangle \quad \textit{by } [\text{overdraft}] \textit{ rule} \tag{14}$$

$$\xrightarrow{s_3} \langle \Gamma', \Gamma', \emptyset \rangle \quad \textit{by } [\text{cover}] \textit{ rule} \tag{15}$$

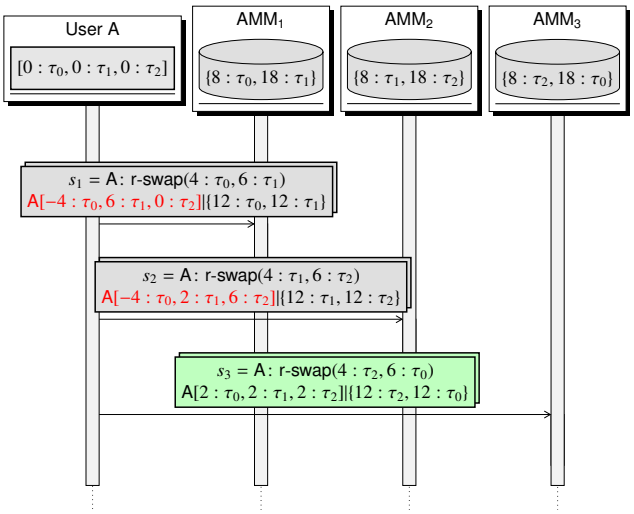

Figure 2: Flow diagram of the Example 1 among User A and three AMMs.

where

$$\Gamma''' = \mathsf{A}[-4:\tau_0, 6:\tau_1, 0:\tau_2] \mid \{12:\tau_0, 12:\tau_1\} \mid \{8:\tau_2, 18:\tau_1\} \mid \{8:\tau_2, 18:\tau_0\}$$

$$\Gamma'' = \mathsf{A}[-4:\tau_0, 2:\tau_1, 6:\tau_2] \mid \{12:\tau_0, 12:\tau_1\} \mid \{12:\tau_2, 12:\tau_1\} \mid \{8:\tau_2, 18:\tau_0\}$$

$$\Gamma' = \mathsf{A}[2:\tau_0, 2:\tau_1, 2:\tau_2] \mid \{12:\tau_0, 12:\tau_1\} \mid \{12:\tau_2, 12:\tau_1\} \mid \{12:\tau_2, 12:\tau_0\}$$

The first two transitions enqueue s_1, s_2 *because they cause an overdraft* (eq. 13 *and* 14) *that is then covered by the last transition* s_3 (eq. 15)*. The execution of* s_3 *results in the* `green` *state* Γ'*.*

Using our mechanism the system evolves $\langle \Gamma, \Gamma, \emptyset \rangle \xrightarrow{\rho} \langle \Gamma', \Gamma', \emptyset \rangle$ *reaching a state where the agent achieves her goal (increasing her wallet) since* $\Gamma'_{\mathsf{A}}(\tau_0) > \Gamma_{\mathsf{A}}(\tau_0)$*,* $\Gamma'_{\mathsf{A}}(\tau_1) > \Gamma_{\mathsf{A}}(\tau_1)$*, and* $\Gamma'_{\mathsf{A}}(\tau_2) > \Gamma_{\mathsf{A}}(\tau_2)$ *(where each of them is* $2 > 0$ *for each token type).*

Note that the above scenario has similarities with the traditional finance scenario known as *gridlock* [6] where banks cannot settle their payments individually due to their insufficient liquidity. Through *netting*, each bank submits its payment instructions to designated queues, performing multilateral settlement exclusively for the net obligations. Back to our example, user A overcomes the stuck state reached by standard AMM protocols where individual swap incurs in an overdraft, enqueuing her actions and atomically performing them when reaching a positive balance.

The second example illustrates the scenario considered in Theorem 7.

Example 2 (r-swap and AMM creation)**.** *Consider a state*

$$\Gamma = \mathsf{A}[0:\tau_0, 0:\tau_1, 6:\tau_2] \mid \mathsf{B}[6:\tau_0, 6:\tau_1, 0:\tau_2] \mid \{8:\tau_0, 18:\tau_1\}$$

and that agents A *and* B *want to perform the following actions* $\rho = s_1 s_2 s_3$ *(where the size of* Λ *is* $\ell = 2$*) :*

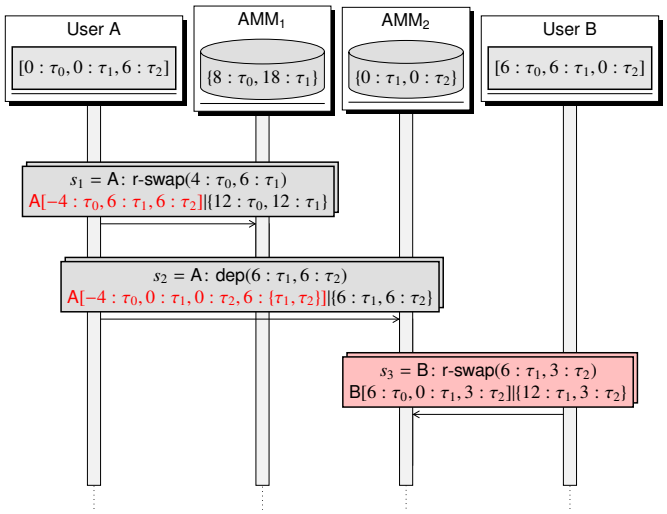

Figure 3: Flow diagram of the Lemma 7 among two Users and two AMMs.

- s_1 = A: r-swap(4 : τ_0, 6 : τ_1)
- s_2 = A: dep(6 : τ_1, 6 : τ_2)
- s_3 = B: r-swap(6 : τ_1, 3 : τ_2)

Figure 3 shows a flow diagram of the pattern composed by a r-swap *action and a deposit action that creates the* AMM$_2$ *that exchanges* τ_1 *and* τ_2.

The evolution of the scenario is composed of the following configurations:

$$\langle \Gamma, \Gamma, \emptyset \rangle \xrightarrow{s_1} \langle \Gamma, \Gamma''', \emptyset \circ s_1 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \tag{16}$$

$$\xrightarrow{s_2} \langle \Gamma, \Gamma'', \emptyset \circ s_1 \circ s_2 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \tag{17}$$

$$\xrightarrow{s_3} \langle \Gamma, \Gamma, \emptyset \rangle \quad \textit{by } [\text{NETTING}] \textit{ rule} \tag{18}$$

where

$$\Gamma''' = \mathsf{A}[-4 : \tau_0, 6 : \tau_1, 6 : \tau_2] \mid \mathsf{B}[6 : \tau_0, 6 : \tau_1, 0 : \tau_2] \mid \{12 : \tau_0, 12 : \tau_1\}$$
$$\Gamma'' = \mathsf{A}[-4 : \tau_0, 0 : \tau_1, 0 : \tau_2, 6 : \{\tau_1, \tau_2\}] \mid \mathsf{B}[6 : \tau_0, 6 : \tau_1, 0 : \tau_2] \mid \{18 : \tau_0, 8 : \tau_1\} \mid \{6 : \tau_2, 6 : \tau_1\}$$

The first transition results in a red *state, so* s_1 *enqueued in* Λ (eq. 16). *Subsequently, the system evolves by considering action* s_2, *which creates a new* AMM$_2$ *for the token type pair* (τ_1, τ_2),[2] *with* 6 *units of each token type respectively* (eq. 17). *User* B *then attempts to obtain token type* τ_2 *by swapping* 6 *units of* τ_1 *on* AMM$_2$ (eq. 18). *The action* s_2 *is added to* Λ *because the negative balance of* A *remains unresolved. Assuming that the following actions do not rectify the negative balance of token type* τ_0, *action* s_2 *saturates the pending queue* Λ, *resulting in a final configuration marked as* red. *At this point, the netting procedure is invoked and proceeds as follows: action* s_1 *is identified as the cause of the overdraft and is subsequently discarded. According to the semantic rules governing interactions between users and* AMM*s, action* s_2 *is also discarded because* A *lacks sufficient* τ_1 *tokens to perform a* deposit *action. The removal of* s_2 *consequently leads to discarding* s_3, *as* B *depends on the* AMM *created by* s_2 *for its execution. The final configuration thus reverts to the initial state* (eq. 18). *In this scenario, the actions executed by the netting mechanism are equivalent to those determined by the standard procedure.*

The following two examples illustrate the scenario considered in *Lemma 5*. The latter highlights how our mechanism deviates from the standard mechanism.

Example 3 (Simultaneous Exchange)**.** *Consider a state*

$$\Gamma = \mathsf{A}[0 : \tau_0, 0 : \tau_1, 4 : \tau_2] \mid \mathsf{B}[4 : \tau_0, 0 : \tau_1, 0 : \tau_2] \mid \{12 : \tau_0, 12 : \tau_1\} \mid \{12 : \tau_1, 12 : \tau_2\}$$

and the following sequence of actions $\rho = s_1 s_2 s_3 s_4$ *(where the max queue size is* $\ell = 3$*):*

- s_1 = A: r-swap(6 : τ_1, 4 : τ_0)
- s_2 = B: r-swap(6 : τ_1, 4 : τ_2)
- s_3 = A: r-swap(4 : τ_2, 6 : τ_1)
- s_4 = B: r-swap(4 : τ_0, 6 : τ_1)

where the first and second r-swap *allow agents* A *and* B *to obtain the token type they value more, namely* τ_1 *and* τ_2, *respectively. The scenario evolves as follows:*

$$\langle \Gamma, \Gamma, \emptyset \rangle \xrightarrow{s_1} \langle \Gamma, \Gamma'''', \emptyset \circ s_1 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \tag{19}$$

$$\xrightarrow{s_2} \langle \Gamma, \Gamma''', \emptyset \circ s_1 \circ s_2 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \tag{20}$$

$$\xrightarrow{s_3} \langle \Gamma, \Gamma'', \emptyset \circ s_1 \circ s_2 \circ s_3 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \tag{21}$$

$$\xrightarrow{s_4} \langle \Gamma', \Gamma', \emptyset \rangle \quad \textit{by } [\text{COVER}] \textit{ rule} \tag{22}$$

where

$$\Gamma'''' = \mathsf{A}[4 : \tau_0, -6 : \tau_1, 4 : \tau_2] \mid \mathsf{B}[4 : \tau_0, 0 : \tau_1, 0 : \tau_2] \mid \{8 : \tau_0, 18 : \tau_1\} \mid \{12 : \tau_1, 12 : \tau_2\}$$
$$\Gamma''' = \mathsf{A}[4 : \tau_0, -6 : \tau_1, 4 : \tau_2] \mid \mathsf{B}[4 : \tau_0, -6 : \tau_1, 4 : \tau_2] \mid \{8 : \tau_0, 18 : \tau_1\} \mid \{18 : \tau_1, 8 : \tau_2\}$$
$$\Gamma'' = \mathsf{A}[4 : \tau_0, 0 : \tau_1, 0 : \tau_2] \mid \mathsf{B}[4 : \tau_0, -6 : \tau_1, 4 : \tau_2] \mid \{8 : \tau_0, 18 : \tau_1\} \mid \{12 : \tau_1, 12 : \tau_2\}$$
$$\Gamma' = \mathsf{A}[4 : \tau_0, 0 : \tau_1, 0 : \tau_2] \mid \mathsf{B}[0 : \tau_0, 0 : \tau_1, 4 : \tau_2] \mid \{12 : \tau_0, 12 : \tau_1\} \mid \{12 : \tau_1, 12 : \tau_2\}$$

These swaps can generate an intermediate negative state (see $\Gamma'''', \Gamma''', \Gamma''$*) and negative net worth associated with* Users'

[2] We provide an illustrative example where a user executes a deposit_0 action to highlight the effects of eliminating the deposit action. This example demonstrates the subsequent changes and consequences resulting from removing the deposit action.

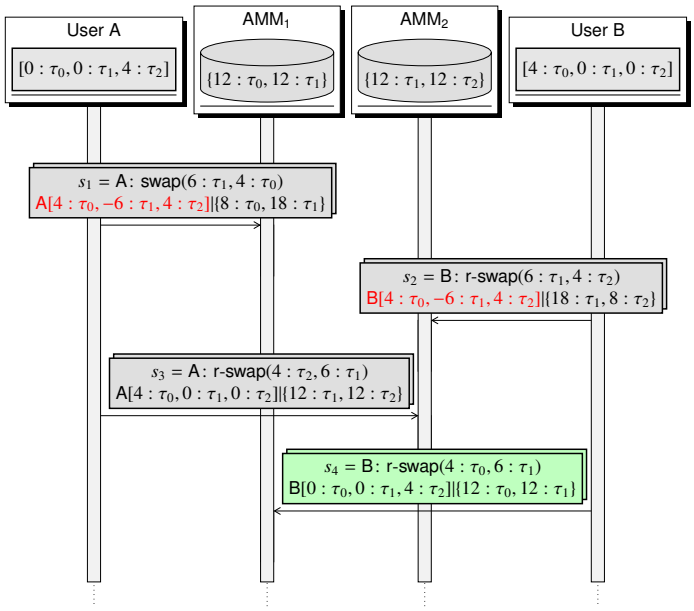

Figure 4: Interaction between two Users and two AMMs.

wallets. The execution of s_1 and s_2 is essential to the achievement of the users' goal. More precisely, agent A *wants to maximize the value of τ_0, while in the latter,* B *wants to maximize the value of τ_2. Once these* r-swap *actions are performed, the respective* AMM*s do not maintain a 1-to-1 ratio anymore. Finally, the s_3 and s_4 are performed in the other market maker to balance the previous overdraft and achieve a 1-to-1 ratio in the* AMM*s again. Once these reversed swaps are performed, the* AMM*s are not unbalanced anymore, and the users' wallets are not negative any longer.*

Analyzing each arrow of Figure 4, it is possible to visualize each relaxed swap performed in ρ. In the context of Γ, both AMM$_1$ *and* AMM$_2$ *maintain a 1-to-1 ratio among the pair token types. Simplifying the symbolic exchanges represented by the pattern ρ reveals that the final state of the system is equivalent to its initial state. Using our mechanism, the system evolves, $\langle \Gamma, \Gamma, \emptyset \rangle \xrightarrow{\rho} \langle \Gamma', \Gamma', \emptyset \rangle$ where Γ' is a* `green` *state with both users' goal achieved: $\Gamma'_{\mathsf{A}}(\tau_0) > \Gamma_{\mathsf{A}}(\tau_0) = 0$ and $\Gamma'_{\mathsf{B}}(\tau_0) > \Gamma_{\mathsf{B}}(\tau_0) = 0$. In more detail,* User A *obtains 4 tokens of type τ_0 and* User B *obtains 4 tokens of type τ_2. Using the standard mechanism, the system does not evolve $\Gamma \xrightarrow{\rho} \Gamma$, and the users do not have any increment because all the action in ρ violates liquidity constraints. Thus, they do not achieve their goal.*

Moreover, our LSM mechanism may settle transactions on the blockchain differently from standard AMM protocols: netting can prioritize transactions that the standard approach would discard, and vice versa. This distinction can impact user rewards, potentially encouraging the adoption of new or alternative market strategies. The following scenario exemplifies a case in which the netting algorithm could prevent swaps that would otherwise be executed using the standard protocol. The following example illustrates this difference.

Example 4 (Incomparability)**.** *Consider a state*

$$\Gamma = \mathsf{A}[0 : \tau_0, 0 : \tau_1, 4 : \tau_2] \mid \mathsf{B}[4 : \tau_0, 6 : \tau_1, 0 : \tau_2] \mid$$

$$\{12 : \tau_0, 12 : \tau_1\} \mid \{18 : \tau_1, 8 : \tau_2\}$$

Assume agents A *and* B *want to perform the sequence of actions $\rho = s_1 s_2 s_3$ (where max queue size is $\ell = 2$).*

- $s_1 = \mathsf{A}\colon \text{r-swap}(6 : \tau_1, 4 : \tau_0)$
- $s_2 = \mathsf{A}\colon \text{r-swap}(4 : \tau_2, 6 : \tau_1)$
- $s_3 = \mathsf{B}\colon \text{r-swap}(6 : \tau_1, 4 : \tau_0)$

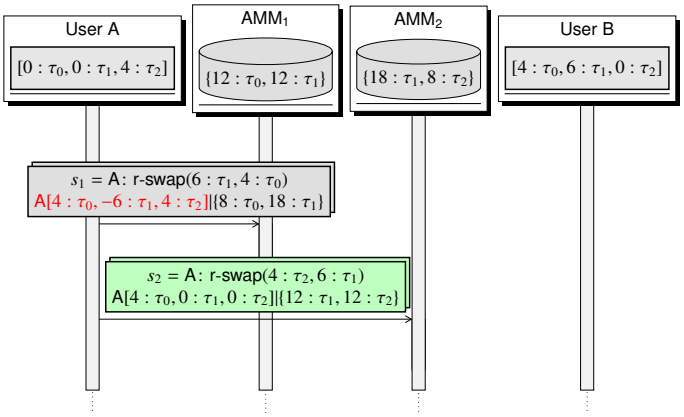

Figure 5: Flow diagram of the Example 4 among two Users and two AMMs.

where the first and second r-swap *allow* A *and* B *to obtain the token type they value more, i.e., τ_1 and τ_2, respectively. Using the standard mechanism, the evolution of the scenario is composed of the following configurations:*

$$\Gamma \xrightarrow{s_1} \Gamma \xrightarrow{s_2} \Gamma^{s} \xrightarrow{s_3} \Gamma^{s'}$$

The action s_1 is discarded because A *does not have enough balance, whereas s_2 and s_3 are performed. Thus, the system reaches the configuration:*

$$\Gamma^{s'} = \mathsf{A}[0 : \tau_0, 6 : \tau_1, 0 : \tau_2] \mid \mathsf{B}[8 : \tau_0, 0 : \tau_1, 0 : \tau_2] \mid$$

$$\{8 : \tau_0, 18 : \tau_1\} \mid \{12 : \tau_1, 12 : \tau_2\}$$

Differently using our mechanism, the scenario evolves in the following configurations:

$$\langle \Gamma, \Gamma, \emptyset \rangle \xrightarrow{s_1} \langle \Gamma, \Gamma'', \emptyset \circ s_1 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \tag{23}$$

$$\xrightarrow{s_2} \langle \Gamma', \Gamma', \emptyset \rangle \quad \textit{by } [\text{COVER}] \textit{ rule} \tag{24}$$

where

$$\Gamma'' = \mathsf{A}[4 : \tau_0, -6 : \tau_1, 4 : \tau_2] \mid \mathsf{B}[4 : \tau_0, 6 : \tau_1, 0 : \tau_2] \mid$$
$$\{8 : \tau_0, 18 : \tau_1\} \mid \{18 : \tau_1, 8 : \tau_2\}$$
$$\Gamma' = \mathsf{A}[4 : \tau_0, 0 : \tau_1, 0 : \tau_2] \mid \mathsf{B}[4 : \tau_0, 6 : \tau_1, 0 : \tau_2] \mid$$
$$\{8 : \tau_0, 18 : \tau_1\} \mid \{12 : \tau_1, 12 : \tau_2\}$$

In detail, the action s_1 is enqueued (eq. 23)*, and when* A *performs s_2 on the second* AMM*. As the Figure 5 shows, both actions are settled because s_2 covers the overdraft on* A*'s wallet* (eq. 24)*. While s_3 is not executed on the first* AMM *because the execution of the previous actions changes the ratio in the reserve, and hence the swap rate. In our mechanism, executing s_2 alters the ratio in the first* AMM*, which makes s_3 no longer executable because the* [Swap_1] *and* [Swap_2] *rules premises $v_1 = \frac{r_1 \cdot v_0}{r_0 + v_0}$ and $0 \leq v_1^* \leq v_1$ are not met. Once $s_1 s_2$ are executed, the updated reserves allow agent* B *to swap* 6 *tokens of τ_1 for only* 2 *tokens of τ_0, which differs from the original trade she aimed to perform (exchanging* 6 *tokens of τ_1 for* 4 *tokens of τ_0). Once $s_1 s_2$ are performed, the ratio of the first* AMM *is updated and s_3 is discarded because agent* B *can swap* 6 *tokens of τ_1 with* 2 *tokens of τ_0 that is a different swap amount compared to what she wants to perform (exchanging* 6 *tokens of τ_1 for* 4 *tokens of τ_0). This example highlights that the standard mechanism and ours generally settle different transactions, so they are incomparable. However, the behavior is not uncommon from what happens in ordinary* AMM*s where a user (in this case* B*) would typically decide to trade at a swap rate based on his local view or speculation on the state of* AMM*s, which can of course change if transactions of other users (in this case* A*) are executed.*

The following two examples illustrate how our netting algorithm works. The first example illustrates the behavior of Algorithm 1 as described in Section 4, while the second one what happens when we change the policy for selecting actions that lead to an overdraft, adopting the policy presented in eq. 12.

Example 5 (Netting scenario)**.** *Assume a state*

$$\Gamma = \mathsf{A}[0 : \tau_0, 0 : \tau_1, 4 : \tau_2] \mid \{12 : \tau_0, 12 : \tau_1\} \mid \{18 : \tau_1, 8 : \tau_2\} \mid \{12 : \tau_2, 12 : \tau_0\}$$

and an agent A *who wants to execute the sequence of actions $\rho = s_1 s_2 s_3$ of the form:*

- $s_1 = \mathsf{A}\colon \text{r-swap}(6 : \tau_2, 4 : \tau_0)$
- $s_2 = \mathsf{A}\colon \text{r-swap}(6 : \tau_1, 4 : \tau_0)$
- $s_3 = \mathsf{A}\colon \text{r-swap}(4 : \tau_2, 6 : \tau_1)$

Supposing that the max size of the queue is $\ell = 2$, thus, the scenario evolves through the following configurations:

$$\langle \Gamma, \Gamma, \emptyset \rangle \xrightarrow{s_1} \langle \Gamma, \Gamma''', \emptyset \circ s_1 \rangle \quad \textit{by } [\text{overdraft}] \textit{ rule} \tag{25}$$

$$\xrightarrow{s_2} \langle \Gamma, \Gamma'', \emptyset \circ s_1 \circ s_2 \rangle \quad \textit{by } [\text{overdraft}] \textit{ rule} \tag{26}$$

$$\xrightarrow{s_3} \langle \Gamma', \Gamma', \emptyset \rangle \quad \textit{by } [\text{netting}] \textit{ rule} \tag{27}$$

where

$$\Gamma''' = \mathsf{A}[4 : \tau_0, 0 : \tau_1, -2 : \tau_2] \mid \{12 : \tau_0, 12 : \tau_1\} \mid \{18 : \tau_1, 8 : \tau_2\} \mid \{18 : \tau_2, 8 : \tau_0\}$$

$$\Gamma'' = \mathsf{A}[8 : \tau_0, -6 : \tau_1, -2 : \tau_2] \mid \{8 : \tau_0, 18 : \tau_1\} \mid \{18 : \tau_1, 8 : \tau_2\} \mid \{18 : \tau_2, 8 : \tau_0\}$$

$$\Gamma' = \mathsf{A}[4 : \tau_0, 0 : \tau_1, 0 : \tau_2] \mid \{8 : \tau_0, 18 : \tau_1\} \mid \{12 : \tau_1, 12 : \tau_2\} \mid \{12 : \tau_2, 12 : \tau_0\}$$

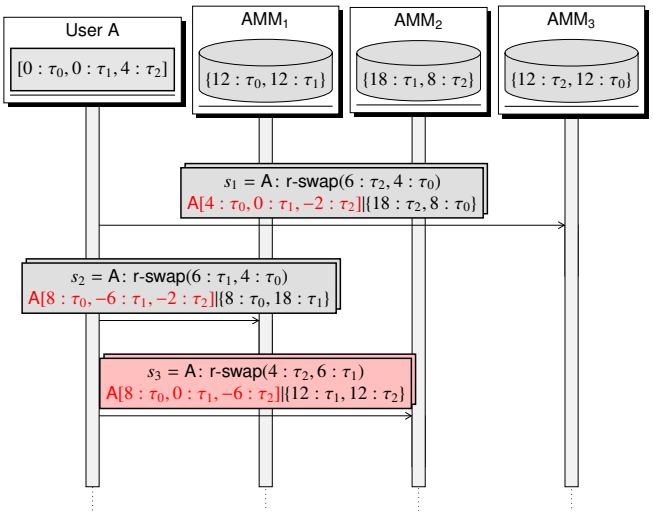

Figure 6: Flow diagram of the Example 5 among User A and three AMMs.

Upon executing the three actions illustrated in Figure 6, the system achieves a `red` *state where the user's wallet fails to meet liquidity constraints. Consequently, our mechanism triggers the* [Netting] *rule and runs the* Algorithm 1*. During this process, s_1 is identified as the first action leading to an overdraft in* A*'s wallet (see line 3 of* Algorithm 1*), and is thus removed from the queue. After this adjustment, the simulation is run again, resulting in a* `green` *state, achieved through the execution of s_2 followed by s_3. It is worth noting that without s_1, s_3 covers the overdraft caused by s_2, thus, the reached state is* `green`*.*

The example above illustrates how our netting algorithm manages transactions while considering liquidity constraints: it enables the execution of actions that subsequently cover the balance overdraft, which would typically not be feasible using a standard semantics to execute the transactions.

The decision regarding the discard action now takes into account the action that causes an overdraft in the user's wallet. The simulation can re-execute the (cleaned) actions until it reaches a `green` state or the queue becomes empty. Despite the formalization and introduction of all possible actions between users and AMMs, the netting mechanism focuses on optimizing and increasing the number of r-swap actions. Moreover, even though the mechanism handles all actions traded between users and AMMs, it prioritizes the improvement of r-swap actions.

We now illustrate the flexibility of our algorithm when we adopt the policy presented in eq. 12.

Example 6 (Netting Maximum Overdraft)**.** *Remember that in our model the codomain of the wallet is the set of real numbers. The numerical values shown below (rounded to two decimal places) arise directly from the evolution of reserves and users' wallets during the execution of actions, as determined by the application of the transition rules.*

Consider a state

$$\Gamma = \mathsf{A}[0 : \tau_0, 0 : \tau_1, 4 : \tau_2] \mid \{12 : \tau_0, 12 : \tau_1\} \mid \{18 : \tau_1, 8 : \tau_2\} \mid \{12 : \tau_2, 12 : \tau_0\}$$

and an agent A *who wants to execute the actions* $\rho = s_1 s_2 s_3 s_4 s_5$ *of the following form (where max queue size is* $\ell = 4$*):*

- s_1 = A: r-swap$(6 : \tau_2, 4 : \tau_0)$
- s_2 = A: r-swap$(6 : \tau_1, 4 : \tau_0)$
- s_3 = A: r-swap$(4 : \tau_2, 6 : \tau_1)$
- s_4 = A: r-swap$(3 : \tau_0, 4 : \tau_2)$
- s_5 = A: r-swap$(4 : \tau_0, 6 : \tau_1)$.

The scenario evolves according to the following configurations:

$$\begin{aligned}
\langle \Gamma, \Gamma, \emptyset \rangle &\xrightarrow{s_1} \langle \Gamma, \Gamma'''', \emptyset \circ s_1 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \\
&\xrightarrow{s_2} \langle \Gamma, \Gamma'''', \emptyset \circ s_1 \circ s_2 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \\
&\xrightarrow{s_3} \langle \Gamma, \Gamma''', \emptyset \circ s_1 \circ s_2 \circ s_3 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \\
&\xrightarrow{s_4} \langle \Gamma, \Gamma'', \emptyset \circ s_1 \circ s_2 \circ s_3 \circ s_4 \rangle \quad \textit{by } [\text{OVERDRAFT}] \textit{ rule} \\
&\xrightarrow{s_5} \langle \Gamma', \Gamma', \emptyset \rangle \quad \textit{by } [\text{NETTING}] \textit{ rule}
\end{aligned}$$

where

$$\begin{aligned}
&\Gamma'''' = \mathsf{A}[4 : \tau_0, 0 : \tau_1, -2 : \tau_2] \mid \\
&\{12 : \tau_0, 12 : \tau_1\} \mid \{18 : \tau_1, 8 : \tau_2\} \mid \{18 : \tau_2, 8 : \tau_0\} \\
&\Gamma'''' = \mathsf{A}[8 : \tau_0, \text{-}6 : \tau_1, \text{-}2 : \tau_2] \mid \{8 : \tau_0, 18 : \tau_1\} \mid \\
&\{18 : \tau_1, 8 : \tau_2\} \mid \{18 : \tau_2, 8 : \tau_0\} \\
&\Gamma''' = \mathsf{A}[8 : \tau_0, 0 : \tau_1, \text{-}6 : \tau_2] \mid \{8 : \tau_0, 18 : \tau_1\} \mid \\
&\{12 : \tau_1, 12 : \tau_2\} \mid \{18 : \tau_2, 8 : \tau_0\} \\
&\Gamma'' = \mathsf{A}[5 : \tau_0, 0 : \tau_1, \text{-}1.09 : \tau_2] \mid \{8 : \tau_0, 18 : \tau_1\} \mid \\
&\{12 : \tau_1, 12 : \tau_2\} \mid \{13.09 : \tau_2, 11 : \tau_0\} \\
&\Gamma' = \mathsf{A}[1 : \tau_0, 0 : \tau_1, 2.9 : \tau_2] \mid \\
&\{12 : \tau_0, 12 : \tau_1\} \mid \{18 : \tau_1, 8 : \tau_2\} \mid \{13.09 : \tau_2, 11 : \tau_0\}
\end{aligned}$$

Upon executing action s_1*, the system enters a* `red` *state where the user's wallet fails to meet liquidity constraints. Once the queue is full, our mechanism triggers* [NETTING] *rule and runs* Algorithm 1. *During the execution of the algorithm,* s_3 *is identified as the action causing the greatest overdraft in* A*'s wallet (see eq 12 and is consequently removed from the queue. After this adjustment, the simulation is run again. As Figure 6 illustrates, the algorithm executes* s_1*,* s_2*,* s_4*, and* s_5 *achieving a* `green` *state.*

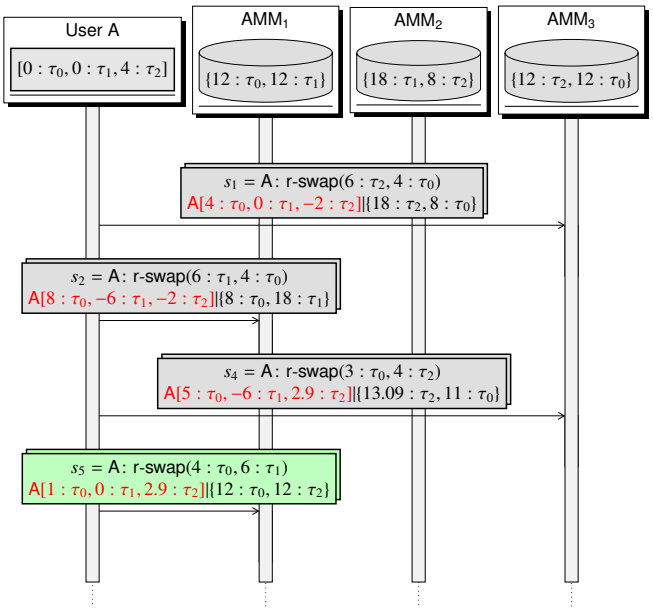

Figure 7: Flow diagram of the Example 5 among User A and three AMMs.

6. Related work and Discussion

In this section, we start by exploring Related Work in a broad sense. Section 6.1 focuses on mechanisms that have inspired our approach or addressed similar challenges. This overview highlights key solutions and frameworks from the literature that share common goals with our research or have influenced the development of our proposed mechanism.

Following this, we explore an alternative design that enables offline netting Section 6.2. Then, we delve into a more targeted discussion on how our mechanism can be adapted to align with the real practice of the blockchain and DeFi ecosystem. More specifically, Section 6.3 examines how evolving trends within the blockchain ecosystem may shape the implementation and scalability of our solution in practice.

6.1. Related Work

The widely spread interest in distributed ledger technology has fostered the development of optimized trading protocols. This section illustrates the most relevant proposals concerning *netting mechanism*, *optimal routing problems*, and *intent-centric protocols*, and compares them with our work.

6.1.1. Liquidity Saving Mechanisms and Netting Approaches

Several papers have developed decentralized inter-bank payment systems where the role of the payment instructions operator is implemented through a public ledger-based protocol, and the netting mechanism through smart contracts. As a first example of this line of work, Jasper-Ubin Project [14] investigates the possibility of achieving near-instant cross-border payments using blockchain. The project removes the single point of failure and obtains an immediate settlement without transaction reconciliation but does not give a decentralized multilateral netting. Naganuma *et al.* [15] provide a secure netting protocol using the Hyperledger Fabric channel and its access control mechanism. Their secure settlement protocol does not require a specific central server and keeps part of the payment transaction information secret. Similarly, Wang *et al.* [16] introduce a blockchain-based netting solution that relies on a central party and preserves the total amount of liquidity, revealing only the net amount of each bank. Cao *et al.* [6] propose a decentralized netting mechanism aggregating local settlements through a smart contract to compute a globally optimal settlement. To protect privacy, they use zero-knowledge proofs to

verify payment amounts confidentially. In their non-privacy-focused version, participants submit signed payment instructions with priority to the smart contract. Payments are settled immediately or queued for later netting, triggered periodically or when the queue reaches a limit. Participants calculate their nettable set and submit it with optimality proofs. The smart contract verifies and aggregates these sets until convergence or deadlock occurs, requiring additional liquidity. The protocol ensures an optimal solution if all participants act honestly. While the above papers [15, 16, 6] apply blockchain technologies to traditional financial services, our work introduces a standard financial mechanism into DeFi protocols, with the goal of implementing a liquidity-saving mechanism in the context of DeFi services. Another approach that introduces an efficient liquidity management is that proposed by Mangrove [17], a decentralized exchange protocol based on the concept of *reactive liquidity*. The main idea is to allow users to post "offers" (smart contracts) without locking funds. Offers are then performed by so-called "takers", who try to execute the offers, obtaining the needed funds if available. In that way, liquidity is only committed when needed. Uncovered transactions in our approach share some similarities in spirit with Mangrove offers, in the sense that they are enqueued without locking funds. However, both approaches differ in the way liquidity saving is achieved: roughly, Mangrove relies on a sort of smart contract call-back approach, while in our case we use netting.

6.1.2. Optimal routing problems

Another relevant research field investigates how to execute several trades of different crypto-assets on networks of multiple CFMMs. These approaches are known as *routing problems* on Decentralized Exchanges. Angeris *et al.* [18] and Diamandis *et al.* [19] pointed out that the optimal routing problem can be formulated as an efficiently solvable optimization problem. Solving such a problem means determining the most efficient path for a trade, i.e., a sequence of crypto-assets exchanges in a network of CFMMs that realizes a given trade, to maximize the user's utility and minimize its costs.

Danos *et al.* [20] introduce arbitrage scenarios within exchange networks and develop an effective global routing system. In detail, they explore how optimal routing strategies are employed to capitalize on price differentials across multiple exchanges, enhancing opportunities for profitable arbitrage. Similar to those approaches, our mechanism aims at finding a solution to maximizing a specific parameter. However, the main difference between ours and the above-mentioned papers [18, 19, 20] is the objective function to maximize. As illustrated in Section 4, our objective function consists of maximizing the number of actions in the queue, i.e., the number of transactions to settle. On the other hand, those proposals formulate the optimization problem in terms of the largest possible user's utility, and their routing problem tries to detect the most convenient and profitable route for executing trades across AMMs of the same and different token types belonging to the network.

6.1.3. Intent-centric protocols

A significant area of research in Ethereum ecosystem focuses on addressing the aggregated token swap problem, i.e., determining a sequence of swaps on AMMs that realizes a given trade by improving the liquidity of users. Various DeFi applications, such as UniswapX [21], Uniswap V4 [22], and CoW Protocol [23], tackle this challenge by adopting *intent-centric protocols*. These protocols shift the focus from transaction execution to defining users' desired outcomes, creating competitive routing marketplaces, introducing gas-free cross-chain swaps, and incorporating batch auctions to discover profitable prices.

UniswapX [21] implements *intent-centric swaps*, combining on-chain and off-chain liquidity for a competitive trading marketplace. Uniswap V4 [22] enhances liquidity with *hooks*, enabling gas-free cross-chain swaps and offering flexibility via customizable features. CoW Protocol [23] utilizes batch auctions for efficient price discovery, and CoW Swap, its decentralized exchange interface, introduces CoW Hooks that allow executing custom actions before and after trades. While these protocols aim to optimize trading across multiple AMMs, our approach introduces a distinctive LSM designed to maximize users' ability to perform specific trades that they define themselves. Here, we refer to *intent* as a goal that the user aims to achieve, specifying the exact actions she wishes to perform. This differs from the concept of intent used in previous protocols, where intent is defined more broadly as the desired outcome, with the protocol determining how that outcome is achieved.

6.2. Discussion on off-line netting

One of the key characteristics of our mechanism is that netting is delegated to users. This assumes that users will have implicit incentives to achieve netting either by submitting transactions that cover overdrafts (cf. rule [Cover]) or by filling up the queue and triggering the netting procedure (cf. rule [Netting]). Besides the absence of explicit incentives, there is an additional issue. Netting via rule [Netting] can be very expensive (e.g., in terms of gas) for the user filling up the queue and triggering the netting algorithm. In practice, a user will only fill up the queue if she knows that the netting procedure will result in enough benefit for her, to compensate for the gas cost. Since the netting procedure proposed would be openly available and is (polynomially) efficient, it is not unrealistic to expect that users can predict the outcome of the netting procedure and decide to trigger it if it makes sense for them. Still, it is natural to wonder whether we could provide an alternative design to our mechanism, where netting happens offline, and incentives to achieve netting are *explicit* and part of the blueprint of the mechanism. Below, we sketch how our model can be extended to accommodate more realistic environments.

Explicit offline netting via user-submitted netting proposals. A first feature that we could add to our mechanism is an explicit netting operation that allows users to submit netting proposals (i.e., subsequences of the current queue) at any moment. Formally speaking, by introducing a new transaction $net(\Lambda')$, this

could be formalized as below:

[OfflineNetting]

$$\frac{\Lambda' \subseteq \Lambda \quad \Gamma' \xrightarrow{\Lambda'} \Gamma'' \quad \Gamma''\texttt{green}}{< \Gamma, \Gamma', \Lambda > \xrightarrow{\mathsf{A}\colon net(\Lambda')} < \Gamma'', \Gamma'', \emptyset >}$$

The rule could be enriched with additional constraints, for example, to enforce that the submitted proposal Λ' is of a certain size (e.g., non-empty, a proportion of the current queue, etc.). However, the above proposed solution would still leave incentives to perform netting implicit in the selected transactions, which may lead to undesired behaviors. For example, a group of users may be interested in netting their own transactions, discarding the transactions of the rest of the users. To remedy this, one could consider incentive mechanisms that explicitly associate some value gain to the transactions in the queue. We discuss below two alternative approaches.

Explicit netting incentives via transaction-based rewards. One possible approach to achieve reward-based incentives would be as follows:

- Introducing a *reward factor* $\epsilon \in \mathbb{R}_{]0;1[}$ to specify a fraction of the tokens involved in the transaction, to be offered as a reward for including the transaction in the queue.
- Adding a the *reward wallet* $\Omega : \mathbb{T} \to \mathbb{R}_+$ to the contract, which is increased by withholding an ϵ fraction from every transaction, and paid out to the user performing the netting operation. To denote this reward in a specific state Γ we write: $\Omega_\Gamma(\tau)$.
- Enriching the state of the contract so that now a state Γ also consists of the current reward Ω, in addition to AMMs and users. Formally states would be of the form:

$$\Gamma = \Omega \mid \mathsf{A}_1[\sigma_{\mathsf{A}_1}] \mid \cdots \mid \mathsf{A}_n[\sigma_{\mathsf{A}_n}] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \cdots \mid \{r_w : \tau_w, r_k : \tau_k\}$$

To formalize how rewards are collected rule [RelaxedSwap] would be amended as follows, where changes w.r.t. the original rule are highlighted in **red**:

[RelaxedSwapWithReward]

$$\frac{v_1 = \dfrac{r_1 \cdot v_0}{r_0 + v_0} \quad 0 \le v_1^* \le v_1 \cdot \mathbf{(1-\epsilon)}}{\begin{array}{c}\mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \mathbf{\Omega} \mid \Gamma \xrightarrow{\mathsf{A}\colon \text{r-swap}(v_0:\tau_0, v_1^*:\tau_1)} \\ \mathsf{A}[\sigma - v_0 : \tau_0 + \mathbf{(1-\epsilon)} \cdot v_1 : \tau_1] \mid \{r_0 + v_0 : \tau_0, r_1 - v_1 : \tau_1\} \\ \mid \mathbf{\Omega + \epsilon \cdot v_1} \mid \Gamma\end{array}}$$

Other AMM transactions, such as deposits and redeems, could be modified similarly.

Then, the rule for user-submitted netting transactions could be updated to formalize the fact that the user submitting the netting proposal will collect the corresponding reward:

[OfflineNettingWithReward]

$$\frac{\begin{array}{c}\Lambda' \subseteq \Lambda \quad \Gamma' \xrightarrow{\Lambda'} \Gamma'' \quad \Gamma''\texttt{green} \quad \mathbf{\Gamma''} = \mathbf{A}[\sigma] \mid \mathbf{\Omega_{\Gamma''}} \mid \mathbf{\Gamma'''} \\ \mathbf{\Gamma^*} = \mathbf{A}[\sigma + \sum_{\tau_i \in \mathbf{dom}\ \Omega} \cdot \mathbf{\Omega_{\Gamma''}}(\tau_i) \ :\ \tau_i] \mid \mathbf{\Omega}(\tau_k) = \mathbf{0}, \tau_k \in \mathbf{dom}\ \mathbf{\Omega} \mid \mathbf{\Gamma'''}\end{array}}{< \Gamma, \Gamma', \Lambda > \xrightarrow{\mathsf{A}\colon net(\Lambda')} < \mathbf{\Gamma^*}, \mathbf{\Gamma^*}, \emptyset >}$$

It is worth noticing that the collected reward is $\Omega_{\Gamma''}$, i.e., it only comprises the reward associated with the transactions that have been included in netting; the reward of excluded transactions is lost. Therefore, users willing to perform netting transactions are incentivized to include as many transactions as possible (or rather the highest value of transactions as possible).

Explicit netting incentives via MEV. An alternative way to incentivize users to submit netting proposals with rewards directly associated with transactions in the queue would be to allow them to extract MEV [24] (see below). This can be achieved, for instance, by resorting to strategies for extracting optimal MEV from AMMs such as the Dagwood sandwiches of [25].

One way to achieve this would be to allow users to submit a supersequence on the current queue, allowing them to insert their own transactions at any place while still requiring netting to result in a `green` state. Thus, the rule for user-submitted netting transactions could be:

[OfflineNettingMEV]

$$\frac{\Lambda' \supseteq \Lambda \quad \Gamma' \xrightarrow{\Lambda'} \Gamma'' \quad \Gamma''\texttt{green} \quad \forall \mathbf{s} \in (\mathbf{\Lambda'} \setminus \mathbf{\Lambda}).\mathbf{s} = \mathsf{A}\colon \mathsf{f}(\dots)}{< \Gamma, \Gamma', \Lambda > \xrightarrow{\mathsf{A}\colon net(\Lambda')} < \Gamma'', \Gamma'', \emptyset >}$$

The premise $\forall s \in (\Lambda' \setminus \Lambda).s = \mathsf{A}\colon \mathsf{f}(\dots)$ ensures that A is not inserting transactions by other users. The premise $\Lambda' \supseteq \Lambda$ forbids A to drop transactions from the queue. This means that this kind of netting operation would only work for those cases where the first transaction is an overdraft by A, who is then incentivized to recover. Said premise could relaxed to allow A to discard overdrafts by other users, or transactions that reduce MEV strategie (e.g. redeem operations [25].)

6.3. Adapting Our LSM to Blockchain and DeFi Trends

Decentralized environments inherently present challenges such as inconsistent views of the system state and information asymmetry among participants. These conditions can affect settlement processes and may create opportunities for certain actors to exploit differences in information access or timing. In particular, issues like transaction reordering, selective inclusion or exclusion, and price slippage are well-documented in the context of decentralized applications and remain the subject of ongoing research. For example, Lührs *et al.* are developing encrypted mempools [26] to address the MEV issues. Although our primary focus is on liquidity management in DeFi protocols, we acknowledge the relevance of these challenges also in our setting. We do not address them here, but we consider them important directions for future investigation.

More specifically, Maximal Extractable Value (MEV) [27] refers to the maximum value that miners, validators, or block producers can extract from transaction ordering, inclusion, or exclusion within a block. MEV typically arises in DeFi systems, where the ability to reorder transactions can lead to profits through arbitrage, front-running, or liquidation opportunities.

While MEV can improve market efficiency by correcting price inconsistencies, it can also harm users by creating opportunities for slippage, front-running, and unfair market manipulation. In Ethereum, MEV has been traditionally earned by

miners in the Proof-of-Work (PoW) model, but after the transition to Proof-of-Stake (PoS), validators are now those who extract MEV.

Flashbots [28] is a service designed to optimize the extraction of MEV while maximizing rewards for validators. It establishes a private transaction pool, where non-mining participants (searchers) submit immutable bundles of transactions. These bundles are then relayed to participating validators, ensuring the transactions are executed efficiently and profitably, bypassing the public Ethereum mempool. Flashbots introduces a *proposer-builder separation* (PBS)[3], wherein specialized block builders optimize blocks for validators to maximize MEV extraction. According to a 2023 report [29], about 95% of Ethereum transactions are routed through Flashbots, with validators earning over 100,000 ETH in additional rewards, significantly increasing their median block reward.

Flashbots plays a significant role in the Ethereum MEV ecosystem, with more than 70% of validators using its MEV-Boost[4] service. Though MEV transactions account for less than 1% of all Ethereum transactions, Flashbots handles 60-80% of MEV-related transactions, dominating the market.

In this setting, while the liquidity-saved AMM might function correctly, Flashbots could capture all the netting rewards, reducing user benefits. For instance, user-proposed netting may not work for regular users, as Flashbots could steal or replay netting solutions, even though this outcome might still be acceptable if the primary goal of efficient netting is achieved.

This service computes the optimal block for miners or validators from user-submitted transactions, extracting MEV and providing miners with a share of these rewards through higher block fees and direct payments. Our mechanism automates the submission of transactions, reducing unnecessary trades and increasing efficiency. Once the netting mechanism is executed, the resulting transactions are sent to validators.

However, routing these transactions through Flashbots introduces several challenges. Flashbots can act as an MEV adversary by controlling transaction flow, potentially capturing all netting rewards and reducing user benefits from optimized trading. It may extract rewards from user transaction optimizations, front-run or reorder netting proposals, and *manipulate transaction* queues by controlling their order, inclusion, or exclusion, rendering them ineffective or even *stealing netting proposals* for its own gain. Another issue involves *commit-reveal schemes*. It could hide transaction details initially to prevent front-running but may still be vulnerable to censorship by Flashbots. If the reveal transaction is censored and fails to make it into a block, the commit becomes ineffective, negating the intended protections. Thus, while Flashbots enhances block profitability for validators and addresses some MEV inefficiencies, it also introduces concerns related to transaction control and value capture that must be considered when designing MEV-related mechanisms.

[3] `https://ethresear.ch/t/proposer-block-builder-separation-friendly-fee-market-designs/9725`

[4] `https://docs.flashbots.net/flashbots-mev-boost/introduction`

We plan to investigate further the alternative design for our LSM and how to adapt it to these blockchain and DeFi trends, aiming for an efficient approach that addresses the issues discussed above.

7. Conclusion

We have introduced a liquidity-optimized mechanism that supports the typical interactions between users and AMMs, such as deposit, redeem, and swap actions, without requiring upfront liquidity checks. We formalized the behavior of our mechanism through an operational semantics, and we demonstrated that in certain scenarios, our LSM allows users to achieve their intents compared to traditional AMMs. Additionally, we refined our netting algorithm and proved properties about its complexity and its behavior when dealing with deposit actions depending from swap ones. We also discussed alternative design approaches for our mechanism that allow users to compute the netting offline and propose its result to the smart contract. This offline mechanism emphasizes the importance of incentive mechanisms to address high gas fees and the costs associated with netting procedures. To validate our approach, we developed a simulator that we used to experiment with various application scenarios, providing valuable insights into the practical implications of our proposal.

In future work, we plan to build upon our concept of intent - linking user goals to transaction execution - by adapting the LSM approach to define users' desired outcomes. This will involve combining the netting mechanism with competitive routing protocols to discover profitable scenarios within AMM and user networks. We also plan to investigate and adapt our LSM to blockchain and DeFi trends and improve the design of Section 6. Another line of investigation will be to improve the design of incentive mechanisms introduced in Section 6.2 that encourage participants to stake significant collateral in exchange for solving specific problems, with rewards dynamically adjusted based on the collateral-to-bounty ratio. We aim to assess whether this approach leads to more effective and scalable solutions. We are also interested in examining whether transparency regarding the internals of the netting mechanism influences user behavior, potentially leading to possible exploitation by malicious users. Furthermore, we will study how the queue length, a crucial parameter in our mechanism, impacts the efficiency of our LSM. Finally, another line of research involves considering our mechanism work in different AMM models, such as Concentrated Liquidity which has been recently added to Uniswap v3 [30].

Acknowledgments

Work partially supported by projects SERICS (PE00000014) and PRIN PNRR 2022 AM∀DEUS (P2022EPPHM) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, and by Copenhagen Fintech project CODeM - Challenges and Opportunities of Defi Models.

References

[1] Aave Documentation, Flash loans, https://docs.aave.com/faq/flash-loans, 2021. Accessed: July, 2024.

[2] K. Qin, L. Zhou, B. Livshits, A. Gervais, Attacking the defi ecosystem with flash loans for fun and profit, in: International conference on financial cryptography and data security, Springer, 2021, pp. 3–32.

[3] Uniswap V2 Guide, Flash swaps, https://docs.uniswap.org/contracts/v2/guides/smart-contract-integration/using-flash-swaps, 2023. Accessed: July, 2024.

[4] M. M. Güntzer, D. Jungnickel, M. Leclerc, Efficient algorithms for the clearing of interbank payments, European Journal of Operational Research 106 (1998) 212–219.

[5] M. Bech, K. Soramaki, Gridlock resolution in payment systems, Danmarks Nationalbank Monetary Review (2001) 67–79. doi:10.2139/ssrn.274290.

[6] S. Cao, Y. Yuan, A. De Caro, K. Nandakumar, K. Elkhiyaoui, Y. Hu, Decentralized privacy-preserving netting protocol on blockchain for payment systems, in: Financial Cryptography and Data Security: 24th International Conference, FC 2020, Springer, 2020, pp. 137–155.

[7] M. Bartoletti, J. H.-y. Chiang, A. Lluch-Lafuente, A theory of automated market makers in defi, Logical Methods in Computer Science 18 (2022) 12.

[8] A. Junge, Liquidity saving mechanism for amms inspired by netting, https://github.com/aleksanderJunge/AMM-netting-inspired-protocol, 2024. Accessed: October, 2024.

[9] M. Renieri, L. Galletta, A. L. Lafuente, J. H.-y. Chiang, A netting protocol for liquidity-saving automated market makers, in: DLT, volume to appear of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 1–15.

[10] H. Adams, Uniswap v3 core, https://uniswap.org/whitepaper-v3.pdf, 2021. Accessed: July, 2024.

[11] A. Person, Sushi protocol documentation, https://docs.sushi.com/pdf/whitepaper.pdf, 2020. Accessed: April, 2024.

[12] M. Egorov, Stableswap-efficient mechanism for stablecoin liquidity, Retrieved Feb 24 (2019) 2021.

[13] M. Bartoletti, J. H.-y. Chiang, A. Lluch Lafuente, Maximizing extractable value from automated market makers, in: International Conference on Financial Cryptography and Data Security, Springer, 2022, pp. 3–19.

[14] B. of Canada, M. A. of Singapore, Enabling cross-border high value transfer using distributed ledger technologies, https://www.mas.gov.sg/-/media/Jasper-Ubin-Design-Paper.pdf?la=en\&hash=EF5857437C4857373A9287CD86F56D0E7C46E7FF, 2018. Accessed: July, 2024.

[15] K. Naganuma, M. Yoshino, H. Sato, N. Yamada, T. Suzuki, N. Kunihiro, Decentralized netting protocol over consortium blockchain, in: International Symposium on Information Theory and Its Applications,ISITA 2018, IEEE, 2018, pp. 174–177. doi:10.23919/ISITA.2018.8664259.

[16] X. Wang, X. Xu, L. Feagan, S. Huang, L. Jiao, W. Zhao, Inter-bank payment system on enterprise blockchain platform, in: 11th IEEE International Conference on Cloud Computing, CLOUD 2018, IEEE Computer Society, 2018, pp. 614–621. doi:10.1109/CLOUD.2018.00085.

[17] The Mangrove Team, Mangrove v5 whitepaper, https://uploads-ssl.webflow.com/60d457cf74454278fd34ae62/63f354e9461fa01543e61604_Mangrove_Whitepaper_v5.pdf, 2021. Accessed: July, 2024.

[18] G. Angeris, A. Evans, T. Chitra, S. Boyd, Optimal routing for constant function market makers, in: Proceedings of the 23rd ACM Conference on Economics and Computation, 2022, pp. 115–128.

[19] T. Diamandis, M. Resnick, T. Chitra, G. Angeris, An efficient algorithm for optimal routing through constant function market makers, arXiv preprint arXiv:2302.04938 (2023).

[20] V. Danos, H. E. Khalloufi, J. Prat, Global order routing on exchange networks, in: Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Springer, 2021, pp. 207–226.

[21] UniswapX whitepaper, Uniswapx, https://uniswap.org/whitepaper-uniswapx.pdf, 2023. Accessed: July, 2024.

[22] Uniswap V4 core whitepaper, Uniswap v4, https://github.com/Uniswap/v4-core/blob/main/docs/whitepaper-v4.pdf, 2023. Accessed: July, 2024.

[23] C. developers, Cow protocol, https://docs.cow.fi/, 2021. Accessed: July, 2024.

[24] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, A. Juels, Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability, in: 2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020, IEEE, 2020, pp. 910–927. URL: https://doi.org/10.1109/SP40000.2020.00040. doi:10.1109/SP40000.2020.00040.

[25] M. Bartoletti, J. H. Chiang, A. Lluch-Lafuente, Maximizing extractable value from automated market makers, in: I. Eyal, J. A. Garay (Eds.), Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers, volume 13411 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 3–19. URL: https://doi.org/10.1007/978-3-031-18283-9_1. doi:10.1007/978-3-031-18283-9_1.

[26] F. Lührs, L. Bezzenberger, F. Mosterts, S. Faust, A. Erwig, The road towards an encrypted mempool on ethereum, 2024. URL: https://docs.shutter.network/docs/shutter/research/the_road_towards_an_encrypted_mempool_on_ethereum, accessed: April 10, 2025.

[27] M. Bartoletti, J. H.-y. Chiang, A. Lluch Lafuente, Maximizing extractable value from automated market makers, in: International Conference on Financial Cryptography and Data Security, Springer, 2022, pp. 3–19.

[28] Flashbots, Flashbots documentation, https://docs.flashbots.net/, 2024. Accessed: September, 2024.

[29] Flashbots Collective, Flashbots transparency report: Mev-share, sgx block building & ethdenver - the flashbots ship, https://collective.flashbots.net, 2023. Accessed: September, 2024.

[30] A. Hayden, Z. Noah, S. Moody, K. River, R. Dan, Uniswap v3, https://uniswap.org/whitepaper-v3.pdf, 2021. Accessed: April, 2024.