# SoK: Lending Pools in Decentralized Finance

Massimo Bartoletti[1], James Hsin-yu Chiang[2], Alberto Lluch Lafuente[2]

[1] Università degli Studi di Cagliari, Cagliari, Italy
[2] Technical University of Denmark, DTU Compute, Copenhagen, Denmark

**Abstract.** Lending pools are decentralized applications which allow mutually untrusted users to lend and borrow crypto-assets. These applications feature complex, highly parametric incentive mechanisms to equilibrate the loan market. This complexity makes the behaviour of lending pools difficult to understand and to predict: indeed, ineffective incentives and attacks could potentially lead to emergent unwanted behaviours. Reasoning about lending pools is made even harder by the lack of executable models of their behaviour: to precisely understand how users interact with lending pools, eventually one has to inspect their implementations, where the incentive mechanisms are intertwined with low-level implementation details. Further, the variety of existing implementations makes it difficult to distill the common aspects of lending pools. We systematize the existing knowledge about lending pools, leveraging a new formal model of interactions with users, which reflects the archetypal features of mainstream implementations. This enables us to prove some general properties of lending pools, and to precisely describe vulnerabilities and attacks. We also discuss the role of lending pools in the broader context of decentralized finance and identify relevant research challenges.

## 1 Introduction

The emergence of permissionless, public blockchains has given birth to an entire ecosystem of *crypto-tokens* representing digital assets. Facilitated and accelerated by smart contracts and standardized token interfaces [1], these so-called *decentralized finance* (DeFi) applications promise an open alternative to the traditional financial system. One of the main DeFi applications are *lending pools*, which incentivize users to lend some of their crypto-assets to borrowers. Unlike in traditional finance, all the parameters of a loan, like its interests, maturity periods or token prices, are determined by a smart contract, which also defines mechanisms to incentivize honest behaviour (e.g., loans are eventually repaid), economic growth and stability. As of April 2021, the two main lending pool platforms hold \$13.5B [26] and \$6.4B [24] worth of tokens in their smart contracts.

Lending pools are inherently hard to design. Besides the typical difficulty of implementing secure smart contracts [2–4,38], lending pools feature complex economic incentive mechanisms, which make it difficult to understand when a lending pool actually achieves the economic goals it was designed for. As a matter of fact, a recent failure of the oracle price feed used by the Compound lending pool platform led to \$100M of collateral being (incorrectly) liquidated

[16]. Indeed, most current literature in DeFi is devoted to study the economic impact of these incentive mechanisms [45, 46, 52, 53, 55, 59].

The problem is made even more complex by the absence of abstract operational descriptions of the behaviour of lending pools. Current descriptions are either high-level economic models [52, 53, 59], or the actual implementations. While, on the one hand, economic models are useful to understand the macroscopic financial aspects of lending pools, on the other hand they do not precisely describe their interactions with users. Still, understanding these interactions is crucial to determine if a lending pool is vulnerable to attacks where some users deviate from the expected behaviour. Implementations, instead, reflect the exact actual behaviour, but at a level of detail that makes high-level understanding and reasoning unfeasible.

**Contributions** This paper presents a systematic analysis of the behaviour of lending pools, of their properties, vulnerabilities, and of the related literature. Based on a thorough inspection of the implementations of the two main lending pool platforms, Compound [14] and Aave [7], we synthesise a formal, operational model of the interactions between users and lending pools, encompassing their incentive mechanisms. More specifically, our contributions are:

1. a formal model of lending pools, which precisely describes their interactions as transitions of a state machine. Our model captures all the typical transactions of lending pools, and all the main economic features, like collateralization, exchange rates, token prices, and interest accrual (Section 3);
2. the formalization and proof of fundamental behavioural properties of lending pools, which were informally stated in literature, and are expected to be satisfied by any implementation (Section 4);
3. the formalization of relevant properties of the incentive mechanisms of lending pools, and a discussion of their vulnerabilities and attacks (Section 5);
4. a thorough discussion on the interplay between lending pools and other DeFi archetypes, like stable coins and automated market makers (Section 7) and the identification of relevant research challenges (Section 8).

Overall, our contributions help address the aforementioned challenges in the design of lending pools. Firstly, our formal model provides a precise understanding of the behaviour of lending pools, abstracting from low-level implementation details. Our model is faithful to mainstream lending pool implementations like Compound [14] and Aave [7]; still, for the sake of clarity, we have introduced high-level abstractions over low-level details: we discuss the differences between our model and the actual lending pool platforms in Section 6. Secondly, our formalisation of the properties of the incentive mechanisms of lending pools makes it easier to understand and analyse their vulnerabilities and attacks. In this regard, our model is directly amenable for its interpretation as an *executable specification*, thus paving the way for automatic analysis techniques, which may include mechanised proofs of contract properties and agent-based simulations of lending pools and other DeFi contracts. Due to space constraints, we provide the proofs of our statements in a separate technical report [56].

## 2 Background

Lending pools (in short, LPs) are financial applications which create a market of loans of crypto-assets, providing incentive mechanisms to equilibrate the market. We now overview the main features of LPs; a glossary of terms is in Table 1.

Users can lend assets to an LP by transferring *tokens* from their accounts to the LP. In return, they receive a *claim*, represented as tokens *minted* by the LP, which can later be redeemed for an equal or increased amount of tokens, of the same *token type* of the original deposit. Lending is incentivized by interest or fees: the depositor speculates that the claim will be redeemable for a value greater than the value of the original deposit. Users can redeem claims by transferring minted tokens to the LP, which pays back the original tokens (with accrued interest) to the redeemer, simultaneously burning the minted tokens. However, redeeming claims is not always possible, as the LP could not have a sufficient balance of the original tokens, as these may have been lent to other users.

User initiate a *loan* by borrowing tokens deposited to an LP. To incentivise users to eventually repay the loan, borrowing requires to provide a *collateral*. Collaterals can be either tokens deposited to the LP when the loan is initiated, and locked for the whole loan duration, or they can be tokens held by the borrower but *seizable* by the LP when a user fails to repay a loan. An unpaid loan of A can be *liquidated* by B, who pays (part of) A's loan in return for a discounted amount of A's collateral. For this to be possible, the value of the collateral must be greater than that of the loan. To incentivize deposits, loans *accrue* interest, which increase a user's loan amount by the *interest rate*.

| | |
|---|---|
| **Token** | A digital representation of some asset, transferable between users. |
| **Token type** | A set of tokens. Tokens of a given type are interchangeable (or *fungible*), whereas tokens of different token types are not. |
| **Native token** | The default token type of a blockchain (e.g., ETH for Ethereum). |
| **Token price** | The price of a token type $\tau$ is the amount of units of a given native cryptocurrency (or fiat currency) needed to buy one unit of $\tau$. |
| **Lender** | A user who transfers units of a token type in return for a *claim* on a full repayment in the future, which may include additional fees or interest. |
| **Claim** | A right to token units in the future. Claims are represented as tokens, which are *minted* and destroyed as claims are created and redeemed. |
| **Minting** | Creation of tokens performed by the LP upon deposits. |
| **Borrower** | A user who wishes to obtain a *loan* of token type $\tau$. The borrower is required to hold *collateral* of another token $\tau'$ to secure the loan. |
| **Collateral** | A user balance of tokens which can be seized if the user does not adequately repay a loan. |
| **Collateralization** | The ratio of deposited *collateral* value over the borrower's total loan value. |
| **Liquidation** | When the *collateralization* of user A falls below a minimum threshold it is *undercollateralized*: here, a user B can repay a fraction of A's loan, in return for a discounted amount of A's collateral *seized* by B. |
| **Interest rate** | The rate of loan growth when accruing interest. |

Table 1: Glossary of financial terms used in Lending Pools.

## 3 Lending pools

In this section we introduce a formal model of lending pools, focussing on the common features implemented by the main LP platforms. We make our model parametric w.r.t. platform-specific features, like e.g. interest rate models, and we abstract from some advanced features, like e.g. governance (we discuss the differences between our model and the main LP platforms in Section 6).

### 3.1 Lending pools basics

We assume a set of ***users*** $\mathbb{A}$, ranged over by $\mathsf{A}, \mathsf{A}', \ldots$, and a set of ***token types*** $\mathbb{T}$, ranged over by $\tau, \tau', \ldots$. Units of these token types can be freely transferred between users, deposited into LPs, and borrowed. When a user deposits units of $\tau$ into an LP, she receives in return units of a token $\{\tau\}$ *minted* by the LP. We denote with $\mathbb{T}_m = \{\{\tau\} \mid \tau \in \mathbb{T}\}$ the set of ***minted token types***. We use $v, v', r, r'$ to range over nonnegative real numbers ($\mathbb{R}_0^+$). We write $r : \tau$ to denote $r$ units of a token type $\tau$ (and similarly, we write $r : \{\tau\}$ for minted token types).

**Wallets and lending pools** We model the ***wallet*** of a user $\mathsf{A}$ as a term $\mathsf{A}[\sigma]$, where the partial map $\sigma \in \mathbb{T} \cup \mathbb{T}_m \rightharpoonup \mathbb{R}_0^+$ represents $\mathsf{A}$'s token holdings. We model a ***lending pool*** as a pair of the form $(r : \tau, \delta)$, where $r$ is the amount of tokens of type $\tau \in \mathbb{T}$ deposited in the LP, and the map $\delta \in \mathbb{A} \to \mathbb{R}_0^+$ represents the users' ***debts*** of tokens of type $\tau$.

**Blockchain states and transactions** We formalise the interaction between users and the blockchain as a labelled transition system. Labels $\mathsf{T}, \mathsf{T}', \ldots$ represent ***transactions*** (see Table 2), while ***states*** $\Gamma, \Gamma', \ldots$ are compositions of wallets, LPs, and a single ***price oracle*** $P \in \mathbb{T} \to \mathbb{R}_0^+ \setminus \{0\}$ which prices tokens. We represent states as terms of the form:

$$\mathsf{A}_1[\sigma_1] \mid \cdots \mid \mathsf{A}_n[\sigma_n] \mid (r_1 : \tau_1, \delta_1) \mid \cdots \mid (r_k : \tau_k, \delta_k) \mid P$$

where all $\mathsf{A}_i$ are distinct, and $\tau_i \neq \tau_j$ for all $i \neq j$. A state $\Gamma$ is *initial* when it only contains a price oracle and a set of wallets, holding only non-minted tokens. We treat states as sets of terms: hence, $\Gamma$ and $\Gamma'$ are equivalent when they contain the same terms; for a term $Q$, we write $Q \in \Gamma$ when $\Gamma = Q \mid \Gamma'$, for some $\Gamma'$.

| | |
|---|---|
| $\mathsf{A} : \mathsf{xfer}(\mathsf{B}, v : \tau)$ | $\mathsf{A}$ transfers $v$ units of $\tau$ to $\mathsf{B}$ |
| $\mathsf{A} : \mathsf{dep}(v : \tau)$ | $\mathsf{A}$ deposits $v$ units of $\tau$, receiving units of minted token $\{\tau\}$ |
| $\mathsf{A} : \mathsf{mxfer}(\mathsf{B}, v : \{\tau\})$ | $\mathsf{A}$ transfers $v$ units of $\{\tau\}$ to $\mathsf{B}$ |
| $\mathsf{A} : \mathsf{bor}(v : \tau)$ | $\mathsf{A}$ borrows $v$ units of $\tau$ |
| $\mathsf{int}$ | All loans accrue interest |
| $\mathsf{A} : \mathsf{rep}(v : \tau)$ | $\mathsf{A}$ repays $v$ units on $\mathsf{A}$'s debt in $\tau$ |
| $\mathsf{A} : \mathsf{rdm}(v : \{\tau\})$ | $\mathsf{A}$ redeems $v$ units of $\{\tau\}$, receiving units of $\tau$ |
| $\mathsf{A} : \mathsf{liq}(\mathsf{B}, v : \tau, \{\tau'\})$ | $\mathsf{A}$ repays $v$ units of $\mathsf{B}$'s debt in $\tau$, seizing units of $\{\tau'\}$ from $\mathsf{B}$ |

Table 2: Transactions.

**Exchange rate** The ***exchange rate*** of a token type $\tau$ in a state $\Gamma$ represents the share of deposited units of $\tau$ over the units of the associated minted tokens. Before formalising it, we define the auxiliary notion of ***supply*** of a token type $t \in \mathbb{T} \cup \mathbb{T}_m$ in a state $\Gamma$, i.e. the sum of the balances of $t$ in all the wallets in $\Gamma$, and possibly in the LPs. It is defined inductively as:

$$\begin{aligned} &sply_t(\mathsf{A}[\sigma]) = \sigma(t) \qquad && sply_t(r:\tau,\delta) = \begin{cases} r & \text{if } t = \tau \\ 0 & \text{otherwise} \end{cases} \\ &sply_t(P) = 0 && sply_t(\Gamma \mid \Gamma') = sply_t(\Gamma) + sply_t(\Gamma') \end{aligned} \tag{1}$$

Then, we define the ***exchange rate*** $ER_\tau(\Gamma)$ as:

$$ER_\tau(\Gamma) \;=\; \begin{cases} \dfrac{r + \sum_{\mathsf{A}} \delta(\mathsf{A})}{sply_{\{\tau\}}(\Gamma)} & \text{if } \Gamma = (r:\tau,\delta) \mid \Gamma',\ r > 0 \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

The idea is that, while initially there is a 1/1 correspondence between minted and deposited tokens, when interest is accrued this relation changes to the benefit of lenders.

**Net worth and collateralization** The ***value*** of $\mathsf{A}$'s tokens in a state $\Gamma$, denoted by $V_\Gamma(\mathsf{A})$, is the sum of the values of all (non-minted) tokens in $\mathsf{A}$'s wallet (the value is the product between token amount and price):

$$V_\Gamma(\mathsf{A}) \;=\; \textstyle\sum_{\tau \in \mathbb{T}} \sigma(\tau) \cdot P(\tau) \qquad \text{if } \Gamma = \mathsf{A}[\sigma] \mid P \mid \Gamma' \tag{3}$$

We define similarly the value $V^m(\mathsf{A})$ of minted tokens held by $\mathsf{A}$. To determine the value of $\{\tau\}$, its price is equated to that of the underlying token $\tau$:

$$V^m_\Gamma(\mathsf{A}) \;=\; \textstyle\sum_{\tau \in \mathbb{T}} \sigma(\{\tau\}) \cdot ER_\tau(\Gamma) \cdot P(\tau) \qquad \text{if } \Gamma = \mathsf{A}[\sigma] \mid P \mid \Gamma' \tag{4}$$

The value $V^d(\mathsf{A})$ of $\mathsf{A}$'s debt is the sum of the value of tokens borrowed by $\mathsf{A}$:

$$V^d_\Gamma(\mathsf{A}) \;=\; \textstyle\sum_{i \in I} \delta_i(\mathsf{A}) \cdot P(\tau_i) \qquad \text{if } \Gamma = \|_{i \in I}(r_i : \tau_i, \delta_i) \mid \|_j \mathsf{A}_j[\sigma_j] \mid P \tag{5}$$

The ***net worth*** of a user is the value of the tokens in her wallet (both minted and non-minted), minus the value of her debt:

$$W_\Gamma(\mathsf{A}) \;=\; V_\Gamma(\mathsf{A}) + V^m_\Gamma(\mathsf{A}) - V^d_\Gamma(\mathsf{A}) \tag{6}$$

The ***collateralization*** of a user is the ratio of the value of her minted tokens and the value of her debt:

$$C_\Gamma(\mathsf{A}) \;=\; \begin{cases} V^m_\Gamma(\mathsf{A}) \;/\; V^d_\Gamma(\mathsf{A}) & \text{if } V^d_\Gamma(\mathsf{A}) > 0 \\ +\infty & \text{otherwise} \end{cases} \tag{7}$$

$$\Gamma_0 = \mathsf{A}[100 : \tau_0] \mid \mathsf{B}[50 : \tau_1] \mid P = \{{}^{1}\!/_{\tau_0}, {}^{1}\!/_{\tau_1}\}$$
$$\xrightarrow{1.\ \mathsf{A}:\mathsf{dep}(50:\tau_0)}$$
$$\Gamma_1 = \mathsf{A}[50 : \tau_0, 50 : \{\tau_0\}] \mid (50 : \tau_0, \{\}) \mid \cdots$$
$$\xrightarrow{2.\ \mathsf{B}:\mathsf{dep}(50:\tau_1)}$$
$$\Gamma_2 = \mathsf{B}[50 : \{\tau_1\}] \mid (50 : \tau_1, \{\}) \mid \cdots$$
$$\xrightarrow{3.\ \mathsf{B}:\mathsf{bor}(30:\tau_0)} \quad \{C_{\Gamma_3}(\mathsf{B}) = 1.7\}$$
$$\Gamma_3 = \mathsf{B}[30 : \tau_0, \cdots] \mid (20 : \tau_0, \{{}^{30}\!/_{\mathsf{B}}\}) \mid \cdots$$
$$\xrightarrow{4.\ \mathsf{int}} \quad \{C_{\Gamma_4}(\mathsf{B}) = 1.5, ER_{\tau_0}(\Gamma_4) = 1.1\}$$
$$\Gamma_4 = \cdots \mid (20 : \tau_0, \{{}^{34}\!/_{\mathsf{B}}\}) \mid \cdots$$
$$\xrightarrow{5.\ \mathsf{B}:\mathsf{rep}(5:\tau_0)} \quad \{C_{\Gamma_5}(\mathsf{B}) = 1.7\}$$
$$\Gamma_5 = \mathsf{B}[25 : \tau_0, \cdots] \mid (25 : \tau_0, \{{}^{29}\!/_{\mathsf{B}}\}) \mid \cdots$$
$$\xrightarrow{6.\ \mathsf{px}} \quad \{C_{\Gamma_6}(\mathsf{B}) = 1.3 < C_{min}\}$$
$$\Gamma_6 = \cdots \mid P' = \{{}^{1.3}\!/_{\tau_0}, \cdots\}$$
$$\xrightarrow{7.\ \mathsf{A}:\mathsf{liq}(\mathsf{B},13:\tau_0,\{\tau_1\})} \quad \{C_{\Gamma_7}(\mathsf{B}) = 1.5 = C_{min}\}$$
$$\Gamma_7 = \mathsf{A}[37 : \tau_0, 19 : \{\tau_1\}, \cdots] \mid \mathsf{B}[31 : \{\tau_1\}, \cdots] \mid (38 : \tau_0, \{{}^{16}\!/_{\mathsf{B}}\}) \mid \cdots$$
$$\xrightarrow{8.\ \mathsf{A}:\mathsf{rdm}(10:\{\tau_0\})} \quad \{ER_{\tau_0}(\Gamma_8) = 1.1\}$$
$$\Gamma_8 = \mathsf{A}[48 : \tau_0, 40 : \{\tau_0\}, \cdots] \mid (27 : \tau_0, \{{}^{16}\!/_{\mathsf{B}}\}) \mid \cdots$$

Fig. 1: Interactions between two users and a lending pool.

**State-update operators** We use the standard notation $\sigma\{{}^{v}\!/_{x}\}$ to update a partial map $\sigma$ at point $x$: namely, $\sigma\{{}^{v}\!/_{x}\}(x) = v$, while $\sigma\{{}^{v}\!/_{x}\}(y) = \sigma(y)$ for $y \neq x$. Given a partial map $\sigma \in \mathbb{T} \cup \mathbb{T}_m \rightharpoonup \mathbb{R}_0^+$, a partial operation $\circ \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \rightharpoonup \mathbb{R}_0^+$, $t \in \mathbb{T} \cup \mathbb{T}_m$ and $v \in \mathbb{R}_0^+$, we define the partial map $\sigma \circ v : t$ as follows:

$$\sigma \circ v : t = \begin{cases} \sigma\{{}^{v'}\!/_{t}\} & \text{if } t \in \operatorname{dom}\sigma \text{ and } v' = \sigma(t) \circ v \in \mathbb{R}_0^+ \\ \sigma\{{}^{v}\!/_{t}\} & \text{if } t \notin \operatorname{dom}\sigma \end{cases}$$

### 3.2 An overview of lending pools behaviour

Before formalizing the behaviour of lending pools, we give some intuition through an example involving users A and B. We display their interactions in Figure 1.

In the initial state, A has 100 units of $\tau_0$, B has 50 units of $\tau_1$, and the price of both token types is 1. In the first two steps, A and B deposit 50 units of $\tau_0$ and $\tau_1$, for which they receive equal amounts of minted tokens $\{\tau_0\}$ and $\{\tau_1\}$. We denote with $\{\}$ the function $\lambda\mathsf{A}.0$ (i.e., no user has debts).

Next, B borrows $30 : \tau_0$. The 50 minted tokens of type $\{\tau_1\}$ in B's wallet serve as *collateral* for the loan. The *collateralization* of B is the ratio between the *value* of B's balance of $\{\tau_1\}$ and the value of B's debt of $\tau_0$, according to (7). Assuming

a minimum collateralization threshold of $C_{min} = 1.5$, B could borrow up to 33 units of $\tau_1$, given the collateral of $50 : \{\tau_1\}$. Nonetheless, B decides to leave some margin to manage future price volatility and the accrual of interest, which can both negatively affect collateralization. In the state $\Gamma_3$, the map $\{^{30}/_{\mathsf{B}}\}$ in the LP for $\tau_0$ represents that B's debt of $\tau_0$ is 30, while the other users have no debt.

In step 4, interest accrues on B's debt. Here, we assume that the interest rate is 12%, so B's debt grows from 30 to 34 units of $\tau_1$. In step 5, B repays 5 units of $\tau_0$ to reduce the risk of becoming *liquidated*, which can occur when B's collateralization falls below the threshold $C_{min} = 1.5$.

Despite this effort, the price is updated in step 6, increasing $P(\tau_0)$ by 30% relative to $P(\tau_1)$, thereby decreasing the relative value of B's collateral to B's loan. As a result, the collateralization of B drops below the threshold $C_{min}$.

In step 7, A liquidates $13 : \tau_0$ of B's debt, restoring B's collateralization to $C_{min}$, and simultaneously seizing $19 : \{\tau_1\}$ from B's balance. The exchange of $13 : \tau_0$ for $19 : \{\tau_1\}$ implies a liquidation discount, which ensures that the liquidation is profitable for any user performing it.

In step 8, A redeems $10 : \{\tau_0\}$, receiving $11 : \tau_0$ in exchange. Here, each unit of $\{\tau_0\}$ is now exchanged for more than 1 unit of $\tau_0$, due to accrued interest.

### 3.3 Lending pools semantics

We now present the rules which define the transitions between lending pool states. In all the rules, denote with $\Gamma_0$ the state *before* the transition, and with $\Gamma_1$ the state *after* the transition. An extended running example (Figures 2–7) illustrates all the peculiar aspects of these rules.

**Token transfer** The transaction $\mathsf{A} : \mathsf{xfer}(\mathsf{B}, v : \tau)$, represents the transfer of $v : \tau$ from A to B. Its effect on the state is specified by the following rule:

$$\frac{\sigma_{\mathsf{A}}(\tau) \geq v}{\mathsf{A}[\sigma_{\mathsf{A}}] \mid \mathsf{B}[\sigma_{\mathsf{B}}] \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{xfer}(\mathsf{B},v:\tau)} \mathsf{A}[\sigma_{\mathsf{A}} - v : \tau] \mid \mathsf{B}[\sigma_{\mathsf{B}} + v : \tau] \mid \Gamma} \text{ [XFER]}$$

Rule [XFER] states that the transfer is permitted whenever the sender has a sufficient balance. Note that the rule only allows transfers of *non*-minted tokens; transfers of minted tokens is specified by rule [MXFER] below.

**Deposit** A user A can deposit $v$ units of a (non-minted) token $\tau$ by performing the transaction $\mathsf{A} : \mathsf{dep}(v : \tau)$. Upon the first deposit of $\tau$, A receives exactly $v$ units of the minted token $\{\tau\}$:

$$\frac{\sigma(\tau) \geq v \qquad (\_ : \tau, \_) \notin \Gamma}{\mathsf{A}[\sigma] \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{dep}(v:\tau)} \mathsf{A}[\sigma - v : \tau + v : \{\tau\}] \mid (v : \tau, \lambda\mathsf{A}.0) \mid \Gamma} \text{ [DEP0]}$$

The first rule premise ensures that A's balance is sufficient. The second premise checks that no LP for $\tau$ is already present in the state. The map $\lambda\mathsf{A}.0$ represents the fact that, in the newly created LP, the debt of each user is 0.

$$\Gamma_0 = \mathsf{A}[100 : \tau_0, 300 : \tau_1] \mid \mathsf{B}[50 : \tau_0, 50 : \tau_2] \mid \mathsf{C}[100 : \tau_2] \mid P$$
$$\xrightarrow{1.\ \mathsf{A}:\mathsf{dep}(100:\tau_0)}$$
$$\Gamma_1 = \mathsf{A}[0 : \tau_0, 100 : \{\tau_0\}, \cdots] \mid (100 : \tau_0, \{\}) \mid \cdots$$
$$\xrightarrow{2.\ \mathsf{A}:\mathsf{dep}(150:\tau_1)}$$
$$\Gamma_2 = \mathsf{A}[150 : \{\tau_1\}, \cdots] \mid (150 : \tau_1, \{\}) \mid \cdots$$
$$\xrightarrow{3.\ \mathsf{B}:\mathsf{dep}(50:\tau_0)}$$
$$\Gamma_3 = \mathsf{B}[0 : \tau_0, 50 : \{\tau_0\}, \cdots] \mid (150 : \tau_0, \{\}) \mid \cdots$$
$$\xrightarrow{4.\ \mathsf{B}:\mathsf{dep}(50:\tau_2)}$$
$$\Gamma_4 = \mathsf{B}[0 : \tau_2, 50 : \{\tau_2\}, \cdots] \mid (50 : \tau_2, \{\}) \mid \cdots$$
$$\xrightarrow{5.\ \mathsf{C}:\mathsf{dep}(100:\tau_2)}$$
$$\Gamma_5 = \mathsf{C}[0 : \tau_2, 100 : \{\tau_2\}] \mid (150 : \tau_2, \{\}) \mid \cdots$$

Fig. 2: Running example: deposit actions

For further deposits of $\tau$, the LP mints new units of $\{\tau\}$. Their amount $v'$ is the ratio between the deposited amount $v$ and the exchange rate $ER_\tau(\Gamma_0)$ between $\tau$ and $\{\tau\}$, defined in (2).

$$\frac{\sigma(\tau) \geq v \qquad v' = v/ER_\tau(\Gamma_0)}{\mathsf{A}[\sigma] \mid (r : \tau, \delta) \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{dep}(v:\tau)} \mathsf{A}[\sigma - v : \tau + v' : \{\tau\}] \mid (r + v : \tau, \delta) \mid \Gamma} \text{[DEP]}$$

Figure 2 exemplifies users depositing funds to the LP. In step 1, A deposits 100 units of $\tau_0$. Since this is the first deposit of $\tau_0$, the LP mints exactly 100 units of $\{\tau_0\}$, and transfers these units to A. In step 2, A deposits 150 units of $\tau_1$; similarly to the previous case, A receives 150 units of $\{\tau_1\}$. In step 3, B deposits 50 units of $\tau_0$. Since $\tau_0$ was already deposited, the LP mints 50 units of $\{\tau_0\}$, and transfers them to B. Finally, in steps 4 and 5, B and C deposit units of $\tau_2$; after that, the balances of tokens $\tau_0$, $\tau_1$, $\tau_2$ in the LP total 150 units.

**Transfer of minted tokens** Minted tokens can be transferred between users, provided that, after the transfer, the sender has enough minted tokens to use as collateral. More specifically, we require that the *collateralization* of the sender in the target state is above a constant threshold $C_{min} > 1$.

$$\frac{\sigma_\mathsf{A}(\{\tau\}) \geq v \qquad C_{\Gamma_1}(\mathsf{A}) \geq C_{min}}{\mathsf{A}[\sigma_\mathsf{A}] \mid \mathsf{B}[\sigma_\mathsf{B}] \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{mxfer}(\mathsf{B},v:\{\tau\})} \mathsf{A}[\sigma_\mathsf{A} - v : \{\tau\}] \mid \mathsf{B}[\sigma_\mathsf{B} + v : \{\tau\}] \mid \Gamma} \text{[MXFER]}$$

**Borrow** Any user can borrow units of a token type $\tau$ from an LP, provided that the LP has a sufficient balance of $\tau$, and that the user has enough minted tokens to use as collateral.

$$\frac{r \geq v > 0 \qquad C_{\Gamma_1}(\mathsf{A}) \geq C_{min}}{\mathsf{A}[\sigma] \mid (r : \tau, \delta) \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{bor}(v:\tau)} \mathsf{A}[\sigma + v : \tau] \mid (r - v : \tau, \delta\{^{\delta(\mathsf{A})+v}/_{\mathsf{A}}\}) \mid \Gamma} \text{[BOR]}$$

$$\Gamma_5 = \mathsf{B}[50 : \{\tau_0\}, 50 : \{\tau_2\}] \mid \mathsf{C}[100 : \{\tau_2\}] \mid (150 : \tau_0, \{\}) \mid (150 : \tau_1, \{\}) \mid \cdots$$
$$\xrightarrow{6.\ \mathsf{B}:\mathsf{bor}(50:\tau_1)} \quad \{C_{\Gamma_6}(\mathsf{B}) = 2.0\}$$
$$\Gamma_6 = \mathsf{B}[50 : \tau_1, \cdots] \mid (100 : \tau_1, \{^{50}/_{\mathsf{B}}\}) \mid \cdots$$
$$\xrightarrow{7.\ \mathsf{C}:\mathsf{bor}(30:\tau_0)} \quad \{C_{\Gamma_7}(\mathsf{C}) = 3.3\}$$
$$\Gamma_7 = \mathsf{C}[30 : \tau_0, \cdots] \mid (120 : \tau_0, \{^{30}/_{\mathsf{C}}\}) \mid \cdots$$
$$\xrightarrow{8.\ \mathsf{C}:\mathsf{bor}(30:\tau_1)} \quad \{C_{\Gamma_8}(\mathsf{C}) = 1.7\}$$
$$\Gamma_8 = \mathsf{C}[30 : \tau_1, \cdots] \mid (70 : \tau_1, \{^{50}/_{\mathsf{B}}, ^{30}/_{\mathsf{C}}\}) \mid \cdots$$

Fig. 3: Running example: borrow actions

We exemplify bor transactions in Figure 3. Users B and C borrow amounts of $\tau_0$ and $\tau_1$ at steps 6–8, keeping their collateralization above $C_{min}$, which is assumed to be 1.5. C's collateralization decreases from 3.3 to 1.7 upon step 8: this is due to the increase in $V^d(\mathsf{C})$, whilst $V^m(\mathsf{C})$ remains constant at 100.

The collateralization of a user depends on the amount of minted tokens she possesses, the amount of tokens she has borrowed, and the price of all tokens involved. Hence, collateralization is potentially sensitive to all actions that can affect those values. This includes both interest accrual and changes in token prices (which are unpredictable), as we shall see. Borrowers must therefore maintain a safety margin in order to protect against potential liquidations.

**Interest Accrual** Interest accrual models the periodic application of interest to loan amounts and can be executed in any state. The action applies a token-specific interest to each loan, updating the debt mapping for *all* users.

$$\frac{\forall i \in I : \forall \mathsf{A} : \delta_i'(\mathsf{A}) = \delta_i(\mathsf{A}) \cdot (1 + I_{\Gamma_0}(\tau_i)) \qquad (\_ : \_, \_) \notin \Gamma}{\|_{i \in I}(r_i : \tau_i, \delta_i) \mid \Gamma \xrightarrow{\mathsf{int}} \|_{i \in I}(r_i : \tau_i, \delta_i') \mid \Gamma} \text{ [Int]}$$

Existing lending pool platforms deploy different algorithmic interest rate models [53]. We leave our model parametric w.r.t. interest rates, and only require that the interest rate is positive, a property that all models in [53] satisfy:

$$I_\Gamma(\tau) > 0 \tag{8}$$

We extend our running example with three interest updates in Figure 4, resulting in the increase of all loan amounts. Each subsequent execution of int *decreases* the collateralization of users B and C, since the $V^d$ of both borrowers *increases* as interest is applied (7).

**Repay** A user with a loan can repay part of it by executing a rep transaction:

$$\frac{\sigma(\tau) \geq v > 0 \qquad \delta(\mathsf{A}) \geq v}{\mathsf{A}[\sigma] \mid (r : \tau, \delta) \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{rep}(v:\tau)} \mathsf{A}[\sigma - v : \tau] \mid (r + v : \tau, \delta\{^{\delta(\mathsf{A})-v}/_{\mathsf{A}}\}) \mid \Gamma} \text{ [Rep]}$$

$$
\begin{aligned}
\Gamma_8 &= (120 : \tau_0, \{^{30}/_{\mathsf{C}}\}) \mid (70 : \tau_1, \{^{50}/_{\mathsf{B}}, ^{30}/_{\mathsf{C}}\}) \mid \cdots \\
&\xrightarrow{9.\ \mathsf{int}} \quad \{C_{\Gamma_9}(\mathsf{B}) = 1.9, C_{\Gamma_9}(\mathsf{C}) = 1.6\} \\
\Gamma_9 &= (120 : \tau_0, \{^{31}/_{\mathsf{C}}\}) \mid (70 : \tau_1, \{^{53}/_{\mathsf{B}}, ^{32}/_{\mathsf{C}}\}) \mid \cdots \\
&\xrightarrow{10.\ \mathsf{int}} \quad \{C_{\Gamma_{10}}(\mathsf{B}) = 1.8, C_{\Gamma_{10}}(\mathsf{C}) = 1.5\} \\
\Gamma_{10} &= (120 : \tau_0, \{^{32}/_{\mathsf{C}}\}) \mid (70 : \tau_1, \{^{56}/_{\mathsf{B}}, ^{34}/_{\mathsf{C}}\}) \mid \cdots \\
&\xrightarrow{11.\ \mathsf{int}} \quad \{C_{\Gamma_{11}}(\mathsf{B}) = 1.7, C_{\Gamma_{11}}(\mathsf{C}) = 1.4\} \\
\Gamma_{11} &= (120 : \tau_0, \{^{33}/_{\mathsf{C}}\}) \mid (70 : \tau_1, \{^{59}/_{\mathsf{B}}, ^{36}/_{\mathsf{C}}\}) \mid \cdots
\end{aligned}
$$

Fig. 4: Running example: interest accrual

$$
\begin{aligned}
\Gamma_{11} &= \mathsf{C}[30 : \tau_0, 30 : \tau_1, 100 : \{\tau_2\}] \mid (120 : \tau_0, \{^{33}/_{\mathsf{C}}\}) \mid (70 : \tau_1, \{^{59}/_{\mathsf{B}}, ^{36}/_{\mathsf{C}}\}) \mid \cdots \\
&\xrightarrow{12.\ \mathsf{C}:\mathsf{rep}(15:\tau_0)} \quad \{C_{\Gamma_{12}}(\mathsf{C}) = 1.9\} \\
\Gamma_{12} &= \mathsf{C}[15 : \tau_0, \cdots] \mid (135 : \tau_0, \{^{18}/_{\mathsf{C}}\}) \mid \cdots
\end{aligned}
$$

Fig. 5: Running example: repay actions

This increases the collateralization of the repaying user, as $V^d$ is reduced (7). Users must always maintain a sufficient collateralization, to cope with adverse effects of interest accruals and price updates.

In Figure 5, C is suffering from low collateralization after the last interest accrual in step 11. Here, $C_\Gamma(\mathsf{C})$ is equal to $C_{min} = 1.5$. The subsequent repayment of 15 units of $\tau_0$ increases C's collateralization back to 1.9.

**Redeem** A user without any loans can redeem minted tokens $\{\tau\}$ for the underlying tokens if enough units of $\tau$ remain in the LP. A user with a *non-zero* loan amount of any token can only redeem minted tokens such that the resulting collateralization is not below $C_{min}$. This constraint does not apply to users without loans, as minted tokens are not used as collateral.

$$
\frac{\sigma(\{\tau\}) \geq v \qquad v' = v \cdot ER_\tau(\Gamma_0) \qquad r \geq v' \qquad C_{\Gamma_1}(\mathsf{A}) \geq C_{min}}{\mathsf{A}[\sigma] \mid (r : \tau, \delta) \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{rdm}(v:\{\tau\})} \mathsf{A}[\sigma - v : \{\tau\} + v' : \tau] \mid (r - v' : \tau, \delta) \mid \Gamma} \ [\textsc{Rdm}]
$$

We exemplify rdm transactions in Figure 6. From Figure 5, B has a non-zero loan amount, hence she can only redeem $11 : \{\tau_2\}$ before her collateralization decreases to $C_{min} = 1.5$, at which B cannot further redeem. Since A has no loans, she can redeem as many tokens $\{\tau_0\}$ as the LP balance permits. For A's redeeming of $50 : \{\tau_0\}$ for $51 : \tau_0$ the exchange rate is $> 1$, because of the accrued interest during the prior execution of int. By contrast, the exchange rate for B is 1, as no loan exists on $\tau_2$, and thus no interest was accrued. The minted tokens $\{\tau_2\}$ and $\{\tau_0\}$ returned to the LP by B and A are burnt.

**Liquidation** When the collateralization of a user B is below the threshold $C_{min}$, another user A can *liquidate* part of B's loan, in return for a discounted

$$
\begin{aligned}
\Gamma_{12} &= \mathsf{A}[0:\{\tau_0\}, 100:\{\tau_0\}, \cdots] \mid (135:\tau_0, \{^{18}/_{\mathsf{C}}\}) \mid \\
&\quad \mathsf{B}[50:\{\tau_0\}, 50:\tau_1, 50:\{\tau_2\}] \mid (150:\tau_2, \{\}) \mid \cdots \\
&\xrightarrow{13.\ \mathsf{B}:\mathsf{rdm}(11:\{\tau_2\})} \quad \{C_{\Gamma_{13}}(\mathsf{B}) = 1.5,\ ER_{\tau_2}(\Gamma_{12}) = 1\} \\
\Gamma_{13} &= \mathsf{B}[11:\tau_2, 39:\{\tau_2\}, \cdots] \mid (139:\tau_2, \{\}) \mid \cdots \\
&\xrightarrow{14.\ \mathsf{A}:\mathsf{rdm}(50:\{\tau_0\})} \quad \{ER_{\tau_0}(\Gamma_{13}) = 1.02\} \\
\Gamma_{14} &= \mathsf{A}[51:\tau_0, 50:\{\tau_0\}, \cdots] \mid (84:\tau_0, \{^{18}/_{\mathsf{C}}\}) \mid \cdots
\end{aligned}
$$

Fig. 6: Running example: redeem actions

amount of minted tokens *seized* from B. A can execute liq if she has enough balance to repay a fraction of the lent token, and if B has a sufficient balance of seizable, minted tokens. The maximum seizable amount is bounded by B's balance of the minted token and by the resulting collateralization of B, which cannot exceed $C_{min}$. After this threshold, B's collateralization is restored, and B is no longer liquidatable.

$$
\frac{\begin{array}{ll}\sigma_{\mathsf{A}}(\tau) \geq v & \sigma_{\mathsf{B}}(\{\tau'\}) \geq v' \qquad \delta(\mathsf{B}) \geq v \\ C_{\Gamma_0}(\mathsf{B}) < C_{min} & C_{\Gamma_1}(\mathsf{B}) \leq C_{min} \end{array} \qquad v' = v \cdot \frac{P(\tau)}{P(\tau')} \cdot r_{liq}}{\begin{array}{l}\mathsf{A}[\sigma_{\mathsf{A}}] \mid \mathsf{B}[\sigma_{\mathsf{B}}] \mid (r:\tau, \delta) \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{liq}(\mathsf{B}, v:\tau, \{\tau'\})} \\ \mathsf{A}[\sigma_{\mathsf{A}} - v:\tau + v':\{\tau'\}] \mid \mathsf{B}[\sigma_{\mathsf{B}} - v':\{\tau'\}] \mid (r + v:\tau, \delta\{^{\delta(\mathsf{B})-v}/_{\mathsf{B}}\}) \mid \Gamma\end{array}}\ [\textsc{Liq}]
$$

where we require that:

$$
C_{min} > r_{liq} > 1 \tag{9}
$$

The constraint $r_{liq} > 1$ implies a discount applied to the seized amount received by the liquidator, as more value is received than repaid. In [Liq], there are no constraints on the collateralization of the liquidator A: its balance of minted tokens increases whilst its lent token amounts remain unchanged, thus always increasing its collateralization (7).

For the liquidations in Figure 7, we set $r_{liq} = 1.1$. After the price update in step 15, both B and C are undercollateralized. C is liquidated by A in step 16, which restores $C_\Gamma(\mathsf{C})$ to 1.5. By contrast, $C_\Gamma(\mathsf{B})$ is 0.9 after the price update. Subsequent liquidations by A seize *all* units $\{\tau_0\}$ and $\{\tau_2\}$ from B's wallet. However, B still has a debt of $11:\tau_1$. This debt is *unrecoverable*, since there is no incentive to repay or liquidate it, given the lack of collateral.

**Price updates** Finally, the price oracle can be updated non-deterministically:

$$
P \mid \Gamma \xrightarrow{\mathsf{px}} P' \mid \Gamma \quad [\textsc{Px}]
$$

## 4 Fundamental properties of lending pools

We now establish some fundamental properties of lending pools. A crucial property is that the exchange rate of any token $\tau$ either strictly increases, when users

$$
\begin{aligned}
\Gamma_{14} &= \mathsf{A}[150 : \tau_1, 50 : \{\tau_0\}, \cdots] \mid \mathsf{B}[50 : \{\tau_0\}, 39 : \{\tau_2\}, \cdots] \mid \mathsf{C}[100 : \{\tau_2\}, \cdots] \mid \\
&\quad (70 : \tau_1, \{{}^{59}/_{\mathsf{B}}, {}^{36}/_{\mathsf{C}}\}) \mid \cdots \mid P = \{{}^{1}/_{\tau_0}, {}^{1}/_{\tau_1}, {}^{1}/_{\tau_2}\} \\
&\xrightarrow{15.\ \mathsf{px}} \quad \{C_{\Gamma_{15}}(\mathsf{B}) = 0.9, C_{\Gamma_{15}}(\mathsf{C}) = 1.3\} \\
\Gamma_{15} &= \cdots \mid P' = \{{}^{1.7}/_{\tau_1}, \cdots\} \\
&\xrightarrow{16.\ \mathsf{A}:\mathsf{liq}(\mathsf{C},27:\tau_1,\{\tau_2\})} \quad \{C_{\Gamma_{16}}(\mathsf{C}) = 1.5\} \\
\Gamma_{16} &= \mathsf{A}[123 : \tau_1, 50 : \{\tau_2\}, \cdots] \mid \mathsf{C}[50 : \{\tau_2\}, \cdots] \mid (97 : \tau_1, \{{}^{59}/_{\mathsf{C}}, {}^{9}/_{\mathsf{C}}\}) \mid \cdots \\
&\xrightarrow{17.\ \mathsf{A}:\mathsf{liq}(\mathsf{B},27:\tau_1,\{\tau_0\})} \quad \{C_{\Gamma_{17}}(\mathsf{B}) = 0.7\} \\
\Gamma_{17} &= \mathsf{A}[96 : \tau_1, 100 : \{\tau_0\}, \cdots] \mid \mathsf{B}[0 : \{\tau_0\}, \cdots] \mid (124 : \tau_1, \{{}^{32}/_{\mathsf{B}}, {}^{9}/_{\mathsf{C}}\}) \mid \cdots \\
&\xrightarrow{18.\ \mathsf{A}:\mathsf{liq}(\mathsf{B},21:\tau_1,\{\tau_2\})} \quad \{C_{\Gamma_{18}}(\mathsf{B}) = 0\} \\
\Gamma_{18} &= \mathsf{A}[75 : \tau_1, 89 : \{\tau_2\}, \cdots] \mid \mathsf{B}[0 : \{\tau_2\}, \cdots] \mid (145 : \tau_1, \{{}^{11}/_{\mathsf{B}}, {}^{9}/_{\mathsf{C}}\}) \mid \cdots
\end{aligned}
$$

Fig. 7: Running example: liquidation actions

have loans on $\tau$, or remain stable otherwise. This guarantees that stocks of the minted token $\{\tau\}$ will gain value.

**Lemma 1.** *Let $\Gamma = (r : \tau, \delta) \mid \cdots$, and let $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$. Then:*

*(a) if $\mathsf{T} = \mathsf{int}$ and $\delta(\mathsf{A}) > 0$ for some $\mathsf{A}$, then $ER_\tau(\Gamma) < ER_\tau(\Gamma')$;*
*(b) otherwise, $ER_\tau(\Gamma) = ER_\tau(\Gamma')$.*

Lemma 2 establishes that the supply of any (non-minted) token is constant.

**Lemma 2.** *Let $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$. Then, for all $\tau \in \mathbb{T}$: $sply_\tau(\Gamma) = sply_\tau(\Gamma')$.*

The net worth of a user can be increased in short or long sequences of transitions. In general, there is no winning strategy (in the game-theoretic sense) for a single user who wants to increase her net worth, unless she can control price updates. However, under certain conditions, winning strategies exist. We consider first a simple 1-player game where a user can choose her next action to improve her net worth in the next state. Lemma 3 shows that liquidation is the only action that allows the user to increases her net worth in a single step.

**Lemma 3.** *Let $\Gamma \xrightarrow{\mathsf{A}:\ell(\cdots)} \Gamma'$. Then:*

*(a) if $\ell = \mathsf{liq}$, then $W_\Gamma(\mathsf{A}) < W_{\Gamma'}(\mathsf{A})$;*
*(b) otherwise, $W_\Gamma(\mathsf{A}) = W_{\Gamma'}(\mathsf{A})$.*

Since this is the winning strategy for all users, but liquidations may be limited by the amount of debts and collaterals, an adversary with the power to drop or reorder transactions could potentially monopolize liquidations for itself. We refer to Section 7 for additional discussion of such attacks.

We now consider a slightly extended game, where $\mathsf{A}$ guesses that the adversary is going to fire $\mathsf{int}$, resulting in $\Gamma_0 \xrightarrow{\mathsf{int}} \Gamma_1$, but can still perform an action

$\mathsf{A} : \ell(\cdots)$ before $\mathsf{int}$, resulting in $\Gamma_0 \xrightarrow{\mathsf{A}:\ell(\cdots)} \Gamma_0' \xrightarrow{\mathsf{int}} \Gamma_1'$. Here, $\mathsf{A}$'s goal is to choose her action $\ell$ such that $W_{\Gamma_1'}(\mathsf{A}) \geq W_{\Gamma_1}(\mathsf{A})$. Lemma 4 shows that $\mathsf{A}$ can achieve this goal by performing deposit, repay, or liquidation actions.

**Lemma 4.** *Let* $\Gamma_0 \xrightarrow{\mathsf{int}} \Gamma_1$ *and* $\Gamma_0 \xrightarrow{\mathsf{A}:\ell(\cdots)} \Gamma_0' \xrightarrow{\mathsf{int}} \Gamma_1'$. *Then:*

*(a) if* $\ell \in \{\mathsf{liq}, \mathsf{dep}, \mathsf{rep}\}$, *then* $W_{\Gamma_1'}(\mathsf{A}) \geq W_{\Gamma_1}(\mathsf{A})$;
*(b) otherwise,* $W_{\Gamma_1'}(\mathsf{A}) \leq W_{\Gamma_1}(\mathsf{A})$.

Overall, Lemmas 3 and 4 determine the set of actions to consider (together with their parameters) to maximize short-term improvements in net worth.

## 5 Lending pool safety, vulnerabilities and attacks

In this section we discuss further properties of lending pools, focusing on risks which could lead to unsecured loans or exploitation by malicious actors. In particular, we consider user collateralization and availability of token funds in LPs (utilization): if these can be targeted by an attacker, the motivation is to limit the LP functionality (denial-of-service) or to make the victim incur losses, which in some cases may imply a gain for the attacker. We consider attackers with the ability to perform some of the actions of the LP model, or even update the price oracle. More powerful attackers that can drop or reorder transactions are discussed in Section 7.

### 5.1 Collateralization bounds and risks

The lending pool design assumes that loans are *secured* by collateral: liquidations thereof are incentivised in order to recover loans if the borrowing users fail to repay. However, collateral liquidation is exposed to risks. First, the incentive to liquidate is only effective if the liquidator values the seized collateral higher than the value of the repaid loan amount, implying a profit. Second, large fluctuations in token price may reduce the relative value of the collateral, eventually making loans partially unrecoverable. Further, an attacker with the ability to update token prices can force users to become undercollateralized, and then seize the collateral of victims without repaying any loans.

**LP-minted token risk** Lending pools must determine the appropriate levels of collateralization based on token prices given by the oracle. However, the value of minted tokens is unpredictable, since they are not determined by price oracles (recall that the domain of $P$ does not include minted tokens). The definition of collateralization in (7) values minted tokens at the same price as their underlying token, just like LP implementations [9,18] do. However, since minted tokens are only redeemable if the LP has sufficient funds (see rule [RDM]), it may happen that users value minted tokens at a lower price than their underlying tokens. This happens e.g. when the funds in LPs are low, which may prevent users from performing redeem actions. Lending pool designs do not account for this, running the risk of incorrectly pricing minted tokens.

**Safe collateralization** Assuming a correct valuation of minted tokens, undercollateralized loans should be swiftly liquidated, given the incentivization provided by the liquidation discount. Furthermore, the user collateral value should be high enough, such that the user's loan amount is sufficiently repaid by liquidations to recover the user collateralization back to $C_{min}$. Therefore, we introduce two notions of safe collateralization.

Inspired by [55], we say that a state is *$\varepsilon$-collateralization safe* when the ratio between the value of the debt of undercollateralized users and the total value of the debt is less than $\varepsilon$:

$$\frac{\sum_{C_\Gamma(\mathsf{A})<C_{min}} V_\Gamma^d(\mathsf{A})}{\sum_{\mathsf{A}} V_\Gamma^d(\mathsf{A})} \leq \varepsilon \tag{10}$$

If the liquidation incentive is effective, a value below $\varepsilon$ should not persist, as users promptly execute liquidations. The efficiency of liquidations has been studied in [59]. Note that large volumes of seized collaterals which are immediately sold on external markets may delay further liquidations, as investigated in [52], due to the external market's finite capacity to absorb such a sell-off.

The notion of $\varepsilon$-collateralization safety does not account for undercollateralized loans which are *non-recoverable*, as previously illustrated in Figure 7. The set of non-recoverable, undercollateralized users are those with a collateralization below $r_{liq}$. The non-recoverable value of a user's debt is defined as $V_\Gamma^{nrd}$ in (11). It represents the remaining value of the debt of a user $\mathsf{A}$ should it be fully liquidated, such that no further collateral can be seized.

$$V_\Gamma^{nrd}(\mathsf{A}) = \begin{cases} V_\Gamma^d(\mathsf{A}) - \frac{V_\Gamma^m(\mathsf{A})}{r_{liq}} & \textit{iff } C_\Gamma(\mathsf{A}) < r_{liq} \\ 0 & \textit{otherwise} \end{cases} \tag{11}$$

From (9) and (11) it follows that when $\mathsf{A}$'s collateralization is below $r_{liq}$, the discounted value of the collateral can no longer reach the remaining debt value.

We say that a state is *strongly $\varepsilon$-collateralization safe* when the fraction of the non-recoverable debt value over the total debt value is below $\varepsilon$:

$$\frac{\sum_{\mathsf{A}} V_\Gamma^{nrd}(\mathsf{A})}{\sum_{\mathsf{A}} V_\Gamma^d(\mathsf{A})} \leq \varepsilon \tag{12}$$

Condition (12) is stronger than (10): if a state is strongly $\varepsilon$-collateralization safe, then it is also $\varepsilon$-collateralization safe. Given equal denominators of (10) and (12), this is a consequence of comparing numerators: note that the numerator of (10) is greater than that of (12), as $V_\Gamma^d(\mathsf{A})$ is greater than $V_\Gamma^{nrd}(\mathsf{A})$, and the set $\{\mathsf{A} \mid C_\Gamma(\mathsf{A}) < C_{min}\}$ is a superset of $\{\mathsf{A} \mid C_\Gamma(\mathsf{A}) < r_{liq}\}$ by (11).

Strong price volatility is a risk for $\varepsilon$-collateralization safety, as a sharp drop in price can immediately reduce a previously over-collateralized user to become under-collateralized: such an immediate drop leaves the user with no opportunity to maintain its collateralization with repayments.

**Attacks on safe collateralization** Malicious agents which can perform price updates can therefore influence the evolution of the LP to lead it to a state that is not $\varepsilon$-collateralization safe or not strongly $\varepsilon$-collateralization safe.

For example, an attacker controlling the price oracle could act as follows. First, she would perform price updates to make any user undercollateralized. The attacker can then perform liquidations on these users and benefit from the discount resulting from both the price update and $r_{liq}$. The attacker has maximized her profits by updating $P$ such that $V_\Gamma^m(\mathsf{B})$ in (7) is close to zero, where $\mathsf{B}$ is a user under attack. In this case, $\mathsf{A} : \mathsf{liq}(\mathsf{B}, v : \tau, \{\tau'\})$ can be performed with a small $v$, and repeated liquidations for different minted tokens can be executed to seize the full balance of $\mathsf{B}$'s collateral.

As a matter of fact, a recent failure of the oracle price feed used by the Compound LP implementation led to $100M of collateral being (incorrectly) liquidated [16]. Though it is unclear whether this was an intentional exploit or not, it illustrates the feasibility of such a price oracle attack.

### 5.2 Utilization bounds and risks

The ***utilization*** of a token $\tau$ is the fraction of units of $\tau$ currently lent to users:

$$U_\tau(\Gamma) \;=\; \frac{\sum_{\mathsf{A}} \delta(\mathsf{A})}{r + \sum_{\mathsf{A}} \delta(\mathsf{A})} \qquad \text{if } \Gamma = (r : \tau, \delta) \mid \Gamma' \tag{13}$$

This notion plays a crucial role in the incentive mechanism of LPs, as explained in [53]: as a matter of fact, it is often used as a key parameter of interest rate models in implementations [25, 27] and literature [53].

**Over- and under-utilization** Note that $U_\tau(\Gamma)$ ranges between 0 and 1. We say that $\tau$ is *under-utilized* if its utilization is 0 and *over-utilized* when it is 1. A state is under-utilized (resp. over-utilized) if it contains under-utilized (resp. over-utilized) tokens.

Under-utilization occurs when some units of $\tau$ have been deposited, but not borrowed by any user. This implies that interest accrual does not increase the debt of any user, as so the exchange rate of $\tau$ in (2) remains constant, thereby not resulting in any gain for lenders.

On the other hand, over-utilization occurs when some users have borrowed units of $\tau$, but the LP has no deposited funds of $\tau$. In this case, users can neither borrow nor redeem.

Under- and over-utilization should be avoided. An optimal utilization of a token type $\tau$ strikes a balance between the competing objectives of interest maximization and the ability for users to borrow $\tau$ tokens or to redeem $\{\tau\}$ tokens. The interest rate models described in [53] intend to incentivize actions of both borrowers and lenders to discover an equilibrium between under- and over-utilization. Informally, this is achieved with interest rate models which rise and fall with utilization: increasing utilization and interest rates incentivize deposits and repayment of loans. Decreasing utilization and interest rates incentivize redeems and additional loan borrowing.

We now discuss under- and over-utilization attacks: note that the first kind of attacks is weaker than the second kind, as funds can still be safely recovered in case of under-utilization.

$$\Gamma_0 = \mathsf{A}[100 : \tau_0] \mid \mathsf{B}[100 : \tau_1] \mid \mathsf{C}[50 : \tau_0] \mid P = \{1/\tau_0, 1/\tau_1\}$$

$$\xrightarrow{1.\mathsf{A}:\mathsf{dep}(100:\tau_0)}$$

$$\Gamma_1 = \mathsf{A}[100 : \{\tau_0\}] \mid (100 : \tau_0, \{\}) \mid \cdots$$

$$\xrightarrow{2.\ \mathsf{B}:\mathsf{dep}(100:\tau_1)}$$

$$\Gamma_2 = \mathsf{B}[100 : \{\tau_1\}] \mid (100 : \tau_1, \{\}) \mid \cdots$$

$$\xrightarrow{3.\ \mathsf{B}:\mathsf{bor}(50:\tau_0)} \quad \{C_{\Gamma_3}(\mathsf{B}) = 2\}$$

$$\Gamma_3 = \mathsf{B}[50 : \tau_0, 100 : \{\tau_1\}] \mid (50 : \tau_0, \{50/\mathsf{B}\}) \mid \cdots$$

$$\xrightarrow{4.\ \mathsf{C}:\mathsf{dep}(50:\tau_0)}$$

$$\Gamma_4 = \mathsf{C}[50 : \{\tau_0\}] \mid (100 : \tau_0, \{50/\mathsf{B}\}) \mid \cdots$$

$$\xrightarrow{5.\ \mathsf{A}:\mathsf{rdm}(100:\{\tau_0\})}$$

$$\Gamma_5 = \mathsf{A}[100 : \tau_0] \mid \mathsf{C}[50 : \{\tau_0\}] \mid (0 : \tau_0, \{50/\mathsf{B}\}) \mid \cdots$$

Fig. 8: An over-utilization attack.

**Under-utilization attacks** Under-utilization attacks can be achieved by an attacker interested in reducing the interest accrual for depositors, or in discouraging the borrowing of a token $\tau$. The attacker can temporarily reduce utilization by repaying large amounts of loans. The effectiveness of this approach will depend on the amounts of $\tau$ repaid by the attacker, as a lowered utilization can also reduce the interest rate (in certain models [53]), thereby incentivizing additional borrowing. An attacker which can update the price oracle can lower the collateralization of borrowers arbitrarily, thereby incentivizing repayments and liquidations to target lower utilization of specific tokens.

**Over-utilization attacks** Over-utilization attack could be achieved by an attacker interested in preventing redeem or borrow actions on $\tau$. The attacker can do this by redeeming all units of $\tau$, while avoiding loans to be repaid or liquidated. We illustrate an over-utilization attack in Figure 8. Users A and C initially hold the entire supply of $\tau_0$ in their wallets. User A colludes with B to steal C's balance of $\tau_0$: in steps 1-2, A and B deposit $100 : \tau_0$ and $100 : \tau_1$, respectively. User B uese her balance of $100 : \{\tau_1\}$ as a collateral to borrow $50 : \tau_0$ from the LP in step 3. At this point, A and B are acting as lender and borrower of $\tau_0$, for which the utilization is 0.5. User C, having observed an opportunity to earn interest on $\tau_0$ decides to deposit $50 : \tau_0$ in step 4. However, user A still has a balance of $100 : \{\tau_0\}$, which she redeems in step 5. Now, users A and B have removed all units of $\tau_0$ from the LP, pushing the utilization of $\tau_0$ to 1, and preventing C from redeeming the minted tokens in his wallet. Of course, B cannot redeem her minted tokens of type $\{\tau_1\}$, since her loan has not been repaid, but this can be considered the cost of the attack.

## 6 Differences between our model and LP platforms

We have synthesised our model from informal descriptions in the literature and the implementation and documentation of lending pools Compound [27] and Aave [25]. To distill a usable, succinct model we have abstracted some implementation details, that could be incorporated in the model at the cost of a more complex presentation. We discuss here some of the main abstractions we made.

The original implementations of Compound and Aave gave administrators control over the economic parameters of the LP, i.e. $C_{min}$, $r_{liq}$, and the interest rate function. This made administrators of such early versions privileged users, who could in principle prevent honest depositors, borrowers and liquidators from withdrawing funds. A Compound administrator, for example, can replace application logic which computes collateralization and authorizes supported tokens [13]. Later versions of these platforms have introduced *governance tokens* (respectively, COMP and AAVE), which are allocated to initial investors or to LP users, who earn units of such tokens upon each interaction. Governance tokens allow holders to propose, vote for, and apply changes in economic parameters, including interest rate functions. By contrast, our model assumes that economic parameters are fixed, and omits governance tokens.

In implementations, adding a new token type to the LP must be authorized by the governance mechanisms. By contrast, in our model any user can add a new token type to the LP by just performing the first deposit of tokens of that type. Implementations also allow administrators or governance to assign weights to each token type. This is intended to adjust collateralization and liquidation thresholds $C_{min}$ and $r_{liq}$ for the predicted price volatility of token types present in a user's loan and collateral. Further, implementations require users to pay *fees* upon actions. These fees are accumulated in a reserve controlled by the governance mechanisms of the LP, and intended to act as a buffer in case of unforeseen events. Our model does not feature token-specific weights and fees.

User liquidations in implementations are limited to repay a maximum fraction of the loan amount [5, 15]. However, this implementation constraint can be bypassed by a user employing multiple accounts, so we omit it in our model.

Lending pool platforms implement the update of interest accrual in a *lazy* fashion: since smart contracts cannot trigger transactions, periodic interest accrual would rely on a trusted user to reliably perform such actions, introducing a source of corruption. Therefore, interest accrual is performed whenever a user performs an action which requires up-to-date loan amounts. Here, the interest rate in implementations is not recomputed for each time period. Instead, a single interest rate is applied to the period since the last interest accrual [8, 17] in order to reduce the cost of execution, leading to inaccuracies in loan interest.

**Comparison with other LP models** Besides the actual LP platforms, we compare our model with other models of LPs in the literature.

The *liquidation model* of [52] is meant to simulate interactions between lending pool liquidations and token exchange markets in times of high price volatility. Unlike in our model, [52] performs liquidations in aggregate, and it omits individual user actions. The interest rate functions of [53] formalize various interest

rate strategies used by LP implementations, and can be seen as complementary to our work. Indeed, even if we did not incorporate such functions directly in our model (for brevity), they could be easily included as instances of $I_\Gamma(\tau)$ in rule [Int]. The work [59] introduces an LP state model, which is instantiated with historical user transactions observable in the Compound implementation deployed on Ethereum. The model abstraction facilitates the observation of state effects of each interaction, and investigates the (historical) latency of user liquidations following the undercollateralization of borrowing accounts. Aforementioned work prioritizes high-level analysis over model fidelity: indeed, the lending pool properties and attacks we present are a direct consequence of the precision in our lending pool semantics.

The emergent behaviour of lending pools in times of high price volatility is examined in [52] by simulation of a lending pool liquidation model. Here, a large price drop can cause many accounts to become undercollateralized: assuming liquidators sell off collateral at an external market for units of the repaid token type, the authors suggest that limited market demand for collateral tokens may prevent liquidations from being executed, thereby posing a risk to *$\varepsilon$-collateralization safety* as we have defined in eqs. (10) and (12).

Lending pool behaviour at the user level is modelled in [55], which simulates agents interacting with the Compound implementation to examine the evolution of *liquidatable* and *undercollateralized* debt, notions similar to *(strong) $\varepsilon$-collateralization safety* (10) (12). [45, 46] examine the competition for user deposits between staking in proof-of-stake systems and lending pools: in the case where lending pools are believed to be more profitable, users may shift deposits away from the staking contract of the underlying consensus protocol towards lending pools, thereby endangering the security of the system.

Lending pool interest rate behaviour is examined in [53], where empirical behaviour of interest rate models in Compound [27], Aave [25] and dYdX [19] are analyzed. In particular, the authors observe a statistically significant coupling in interest rates between deployed lending pools, suggesting that the dynamic interest models are effective in discovering a global interest rate equilibrium for a given token. Our formal model is parameterized by the interest rate, that must always be positive (8): since this property holds for all interest rate functions in [53], our model can be instantiated with them.

## 7 DeFi archetypes beyond lending pools

We now discuss the interplay between lending pools and other DeFi archetypes, like algorithmic stable coins, automated market makers, margin trading and flash loans, which are all predominantly deployed on the Ethereum blockchain [41]. We refer to [66] for an overview of these DeFi archetypes.

**Algorithmic stable coins** MakerDAO [21] is the leading algorithmic stable coin and is credited with being one of the earliest DeFi projects. It incorporates several features found in lending pools, such as deposits, minting, and collateralization. As of April 2021, $7.5B [30] worth of crypto-tokens are locked in the

MakerDAO implementation. Users are incentivized to interact with the smart contract to mint or redeem DAI tokens. This, in turn, adjusts the supply of DAI such that a stable value against the reference price (e.g., USD) is maintained. Synthetic tokens are similar to algorithmic stable coins but may track an asset price such as gold or other real-world assets. Reference asset prices are determined by price oracles.

The work [58] introduces a taxonomy for various price stabilization mechanisms, providing insight into the functionality of such contracts. The work [52] uncovers a vulnerability in the governance design of MakerDAO, allowing an attacker to utilize flash loans to steal funds from the contract. The empirical performance of MakerDAO's oracles is studied in [51], which also proposes alternate price feed aggregation models to improve oracle accuracy. Finally, [48] investigates the optimal bidding strategy for collateral liquidators in MakerDAO, which is executed by through user auctions.

Stable coins which track prices of real-world currencies (e.g. USD) exhibit a price stability useful for lending pools: users with stable collateral or loan values have a lower likelihood of suddenly becoming undercollateralized.

**Automated market makers** Automated market makers (AMMs) allow users to exchange units of a token $\tau$ for units of another token $\tau'$ and vice-versa. AMMs do not match opposing actions of buyers and sellers: users simply exchange tokens with an AMM, where the exchange rate is determined algorithmically as a function of the AMM state. Hence, the dynamic exchange rate of an AMM is affected with each user interaction. As of April 2021, leading AMMs Uniswap [34] and Curve Finance [28] hold \$5.3B [33] and \$4.6B [28] worth of tokens and feature an estimated \$1.3B [33] and \$180M [28] worth of token exchanges every day.

The work [36] investigates algorithmic exchange rate models and defines the user arbitrage problem, where a profit-seeking agent must determine the optimal set of AMMs (with differing exchange rates) to interact with: given such arbitrage opportunities will be exploited by rational users, it is expected that exchange rates across AMMs remain consistent. AMM price models can fail: the *constant product* exchange rate model implemented by Uniswap [34] and Curve [29] is simple, but can theoretically reach a state where the the exchange rate is arbitrarily high. The work [65] proposes bounded exchange rate models to address this issue.

A theory of AMMs is proposed in [57], formally specifying their possible interactions and their economic mechanisms. This allows [57] to develop a concurrency theory of AMMs: in particular, it shows that sequences of deposit and redeem actions can be ordered interchangeably, resulting in observationally equivalent states. Be leveraging the formal model, [57] establishes fundamental properties of AMMs, like e.g. the preservation of deposited token supplies, and *token liquidity*, which ensures that deposited tokens cannot be frozen in an AMM. Further, it devises a general solution to the arbitrage problem, the main game-theoretic foundation behind the economic mechanisms of AMMs.

The work [35] suggests that AMMs track global average token prices effectively. As such, AMMs can inform price oracles: such oracles, however, only

update price information with each new block [23] computed from time-weighted price averages of AMMs over the past block interval. This increases the cost of manipulating prices of the oracle, as the manipulated price must be sustained over a period of time. We note that lending pool implementations do not rely on oracles which derive prices from AMM states.

AMMs suffer from *front-running attacks*, where an attacking user observes the victim's unconfirmed token exchange transaction, and sequences its own transaction prior to that of the victim. A front-running attack on an AMM user takes advantage of the update in exchange rate resulting from the victim's token exchange, who ends up paying a higher price, as illustrated in [67]. Front-running of smart contracts is investigated more generally in [50]: mitigations such as commit-and-reveal schemes are proposed, which come with an increased cost for user-contract interactions. In the context of AMMs, [47] introduces the notion of gas auctions, where adversarial users compete to front-run a given AMM exchange transaction by outbidding each others transaction fee.

We note that similar attacks can be modeled with an attacker that can drop or reorder transactions in our lending pool model. Such an attacker can trivially defer attempts of a borrower to repay a loan: subsequent interest accrual will eventually cause the user to become undercollateralized, so that the attacker can liquidate the victim. Such an attacker can also monopolize all liquidations for herself, preventing other users from executing such an action. The work [47] suggests that miners may be incentivized to perform such attacks due to gain resulting from liquidation discounts.

**Margin trading** An important use case of lending pools are *leveraged* long or short positions initiated by users, also referred to as margin trading. In a leveraged long position of $\tau$ against $\tau'$, the user speculates that the price of the former will increase against the price of the latter: a user borrows $\tau'$ at a lending pool against collateral deposited in $\tau$, and then exchanges the borrowed units of $\tau'$ back to $\tau$ at a token exchange or an AMM. The user will now earn an amplified profit if the price of $\tau$ appreciates relative to $\tau'$, since both the borrowed balance and redeemable collateral in $\tau$ appreciates in value whilst only the loan repayable with $\tau'$ decreases in value. A leveraged short position simply reverses the token types. Margin trading contracts such as bZx Fulcrum [11] combine lending and AMM functionalities to offer margin trades through a single smart contract. However, since such margin trading contracts perform large token exchanges at external AMMs, attackers can use such actions to manipulate AMM prices, as shown in [60]. Furthermore, the scope of such attacks is magnified when performed with flash loans.

**Flash loans** Any smart contract holding tokens can expose flash loan functionality, allowing users to borrow and return a loan within a single atomic transaction group. Atomic transaction groups are sequences of actions from a single user, which must execute to completion or not execute at all. They can be implemented in Ethereum by user-defined smart contracts [6], and they are natively supported by Algorand [39]. As such, flash loans are guaranteed to be repaid or not executed at all. The work [64] introduces a framework to identify

flash loan transactions on the Ethereum blockchain for an analysis of their intended use-cases, which include arbitrage transactions, account liquidations (in lending pools or stable coins) and attacks on smart contracts. Our model can be easily extended to encompass flash loan semantics.

Flash loans have been used in several recent attacks [10, 12, 20, 22, 60]. The flash loan attack on bZx Fulcrum described in [60] involves sending the borrowed tokens to a margin trading contract, which, in turn, initiates a large token exchange at an external AMM: here, the large amount of exchanged tokens causes a significant shift in dynamic AMM exchange rate, which represents an arbitrage opportunity exploited by the attacker in several execution steps involving other contracts. Flash loans provide attackers with access to very large token values to initiate attacks.

## 8 Research challenges

Our model already allows us to formally establish properties of LPs (Section 4), and to precisely describe potential attacks to LPs as sequences of user actions (Section 5). However, lending pools operate within a wider DeFi ecosystem, composed by a set of collaborating or competing agents, interacting through possibly separate contract execution environments enabled by miners, who may have transaction ordering privileges and their own goals [47]. We highlight some open research challenges for the compositional security in DeFi systems, where we expect lending pool applications to play a central role.

**Agent strategies** As shown in Lemma 3, there exist *rational* lending pool actions which always increase the net worth of the agent. We contrast such risk-free rational strategies against those which are *speculative*, driven by an agent's expectation of a future system state which is not guaranteed: depositing or borrowing from an LP are speculative strategies, as they are motivated by an expectation of future interest, which are, in turn, regulated by future actions of borrowers and depositors.

Whereas there appears to be a clear path towards formal specification of rational strategies in DeFi systems, the specification of speculative agent behaviour in DeFi remains an open question. For individual DeFi archetypes, agent-based models have been proposed [23, 55] with a focus on rational behaviour, yet the specification of economically speculative strategies in a richer composition of DeFi application remains an open research challenge.

Classical agent-based models from economic disciplines feature specification techniques of economically (speculative) agent behaviour: here, we also observe that stochastic model checking tools from formal methods are increasingly deployed [63] in the economic research community and suggest that stochastic model checking of agent-based models of DeFi systems may provide a path forward towards the automatic analysis of agent strategies.

**A model of transaction concurrency** As exemplified by Lemma 4, actions in lending pools and DeFi are generally not concurrent. In particular, the exploitation of non-concurrency in AMM's has received much attention, where

an actor with transaction ordering privileges can benefit from ordering its own transaction before and after that of the victim [54,67] for financial benefit. More generally, the ability of miners to extract value beyond transaction fees from specific sequences of DeFi interactions has been denoted miner-extractable-value (MEV) [47]. For LP applications, a rational miner is incentivized to perform liquidation actions itself, thereby invalidating liquidation attempts by other users: this may support the security of LP's, as loans are quickly liquidated. However, it also highlights the challenges in developing a formal model of a DeFi system composed of different DeFi applications. Such a model must feature a notion of *incentive-consistent* action sequences in the presence of rational agents with transaction ordering privileges, such that any miner interaction with DeFi applications are intended and beneficial the security of the DeFi system.

Towards the goal of exploring action concurrency in a composed DeFi system, [62] models user functionality enabled by composing an AMM and LP: here, an AMM pair offers swaps between two stable coin types, which are provided by depositors. The deposited stable coins, however, are forwarded to lending pools, thus enabling AMM depositors to also earn interest in addition to swap fees. The resulting agent model is implemented as communicating sequential processes, allowing the exploration of different action sequences.

We note, however, that such analysis is further complicated by atomic chains of transactions, such as those obtained by nested contract calls in Ethereum. Here, the sequencing of individual actions within the call-chain is determined by the authorizing user: this can result in DeFi exploits amplified by flash loans [42,60,64]. As transactions, call-chains must also exhibit consistency with miner transaction ordering incentives: here, a lack of formal models to integrate call-chain semantics with formal models of MEV remains apparent.

A model of transaction ordering may ultimately facilitate the automated analysis of a DeFi system specification which includes lending pools, given that it narrows the set of *valid* interaction sequences. Given sufficiently specified agent strategies, such a theory may pave the way towards novel model checking techniques in DeFi.

**Cryptographic protocol composition** Cryptographic protocols play an increasingly central role in DeFi systems, as they allow DeFi applications to keep private selected parts of the application state: public execution introduces incentives (MEV) which challenge DeFi security, but the public execution of user actions also compromises privacy. The popularity of crypto-asset mixers [32] powered by ZK-SNARK proofs on the Ethereum blockchain foreshadows the emergence of privacy-focused DeFi applications, which in turn, may open new approaches to mitigate MEV. Private order-matching has been proposed with multi-party-computation techniques [43], and we foresee similar techniques for DeFi applications. Furthermore, advanced cryptographic protocols improve scalability: many DeFi applications have migrated to ZK-rollups [31] in order to absorb the increased user demand on the Ethereum blockchain.

For the secure composition of cryptographic protocols deployed for both privacy and scalability, the formal methods community may contribute both

classical information flow [44] analysis techniques and cryptographic protocol composition analysis [49]: as a multitude of privacy-focused and scalable applications are composed in a single system, we highlight the formal analysis of safe cryptographic protocol composition in DeFi as an new research frontier.

**Domain-specific languages** Since the analysis of security aspects of DeFi applications will invariably involve specifications of agents and miners, higher abstractions of DeFi specification will arguably be of interest to the DeFi and formal methods communities. Domain-specific languages with formal semantics (e.g. [37,40,61]) provide suitable specification means for such abstractions. Moreover, they fulfill two purposes: firstly, they enable formal reasoning and security proofs. Secondly, DeFi-specific languages can provide built-in security guarantees, given a foundational theory of the underlying DeFi system.

**Acknowledgements** Massimo Bartoletti is partially supported by Conv. Fondazione di Sardegna & Atenei Sardi project F74I19000900007 *ADAM*. James Hsin-yu Chiang is supported by the PhD School of DTU Compute. Alberto Lluch Lafuente is partially supported by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu).

## References

1. ERC-20 token standard (2015), https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
2. Understanding the DAO attack (June 2016), http://www.coindesk.com/understanding-dao-hack-journalists/
3. Parity Wallet security alert (July 2017), https://paritytech.io/blog/security-alert.html
4. A Postmortem on the Parity Multi-Sig library self-destruct (November 2017), https://goo.gl/Kw3gXi
5. Aave maximum liquidation amount (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/lendingpool/LendingPoolLiquidationManager.sol#L181
6. Aave v1 flashloan receiver interface (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/flashloan/interfaces/IFlashLoanReceiver.sol#L11
7. Aave v1 implementation (2020), https://github.com/aave/aave-protocol/tree/efaeed363da70c64b5272bd4b8f468063ca5c361
8. Aave v1 simplified interest (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/libraries/CoreLibrary.sol#L423
9. Aave valuation of atokens (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/lendingpool/LendingPoolDataProvider.sol#L114
10. Akropolis Defi attack (2020), https://cryptonews.com/news/defi-akropolis-drops-20-following-a-usd-2m-heavy-hack-8299.htm
11. bzx fulcrum website (2020), https://fulcrum.trade

12. Coindesk: Value DeFi attack (2020), https://www.coindesk.com/value-defi-suffers-6m-flash-loan-attack
13. Compound comptroller setter (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/CToken.sol#L1152
14. Compound implementation (2020), https://github.com/compound-finance/compound-protocol/tree/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6
15. Compound maximum liquidation amount (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/ComptrollerG5.sol#L510
16. Compound oracle attack (2020), https://news.bitcoin.com/100-million-liquidated-on-defi-protocol-compound-following-oracle-exploit
17. Compound simplified interest (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/CToken.sol#L423
18. Compound valuation of ctokens (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/ComptrollerG5.sol#L753
19. dydx website (2020), https://dydx.exchange
20. Harvest Finance flashloan attack post-mortem (2020), https://medium.com/harvest-finance/harvest-flashloan-economic-attack-post-mortem-3cf900d65217
21. Makerdao website (2020), https://https://makerdao.com
22. Origin Dollar attack (2020), https://cryptonews.com/news/4th-major-defi-hack-in-a-month-origin-dollar-loses-usd-7m-8331.htm
23. Uniswap oracle template (2020), https://github.com/Uniswap/uniswap-v2-periphery/blob/dda62473e2da448bc9cb8f4514dadda4aeede5f4/contracts/examples/ExampleOracleSimple.sol
24. Aave markets website (2021), https://app.aave.com/markets
25. Aave website (2021), https://www.aave.com
26. Compound markets website (2021), https://compound.finance/markets
27. Compound website (2021), https://www.compound.finance
28. Curve statistics (2021), https://www.curve.fi/dailystats
29. Curve website (2021), https://www.curve.fi
30. Defi pulse website (2021), https://defipulse.com
31. Starkware (2021), https://starkware.co/
32. Tornado (2021), https://tornado.cash/
33. Uniswap statistics (2021), https://info.uniswap.org
34. Uniswap website (2021), https://www.uniswap.org
35. Angeris, G., Chitra, T.: Improved price oracles: Constant function market makers. arXiv preprint arXiv:2003.10001 (2020), https://arxiv.org/abs/2003.10001
36. Angeris, G., Kao, H.T., Chiang, R., Noyes, C., Chitra, T.: An analysis of uniswap markets. Cryptoeconomic Systems Journal (2019), https://ssrn.com/abstract=3602203
37. Arusoaie, A.: Certifying Findel derivatives for blockchain. Journal of Logical and Algebraic Methods in Programming **121**, 100665 (2021). https://doi.org/https://doi.org/10.1016/j.jlamp.2021.100665
38. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: Principles of Security and Trust (POST). LNCS, vol. 10204, pp. 164–186. Springer (2017). https://doi.org/10.1007/978-3-662-54455-6_8

39. Bartoletti, M., Bracciali, A., Lepore, C., Scalas, A., Zunino, R.: A formal model of Algorand smart contracts. In: Financial Cryptography (2021), (to appear) https://arxiv.org/abs/2009.12140
40. Bartoletti, M., Zunino, R.: BitML: a calculus for Bitcoin smart contracts. In: ACM CCS (2018). https://doi.org/10.1145/3243734.3243795
41. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)
42. Cao, Yixin and Zou, Chuanwei and Cheng, Xianfeng: Flashot: A Snapshot of Flash Loan Attack on DeFi Ecosystem. arXiv preprint arXiv:2102.00626 (2021), https://arxiv.org/abs/2102.00626
43. Carsten Baum and Bernardo David and Tore Frederiksen: P2DEX: Privacy-Preserving Decentralized Cryptocurrency Exchange. Cryptology ePrint Archive, Report 2021/283 (2021), https://eprint.iacr.org/2021/283
44. Cecchetti, Ethan and Yao, Siqiu and Ni, Haobin and Myers, Andrew C: Compositional Security for Reentrant Applications. arXiv preprint arXiv:2103.08577 (2021), http://arxiv.org/abs/2103.08577
45. Chitra, T.: Competitive equilibria between staking and on-chain lending. arXiv preprint arXiv:2001.00919 (2019), https://arxiv.org/abs/2001.00919
46. Chitra, T., Evans, A.: Why stake when you can borrow? Available at SSRN 3629988 (2020), http://dx.doi.org/10.2139/ssrn.3629988
47. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: IEEE Symposium on Security and Privacy. pp. 910–927. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00040
48. Darlin, M., Papadis, N., Tassiulas, L.: Optimal Bidding Strategy for Maker Auctions. arXiv preprint arXiv:2009.07086 (2020), https://arxiv.org/abs/2009.07086
49. Dolev, Danny and Yao, Andrew: On the security of public key protocols. IEEE Transactions on information theory **29**(2), 198–208 (1983)
50. Eskandari, S., Moosavi, S., Clark, J.: SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In: Financial Cryptography. pp. 170–189. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-43725-1_13
51. Gu, W.C., Raghuvanshi, A., Boneh, D.: Empirical measurements on pricing oracles and decentralized governance for stablecoins. Available at SSRN 3611231 (2020), http://dx.doi.org/10.2139/ssrn.3611231
52. Gudgeon, L., Pérez, D., Harz, D., Livshits, B., Gervais, A.: The decentralized financial crisis. In: Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 1–15. IEEE (2020). https://doi.org/10.1109/CVCBT50464.2020.00005
53. Gudgeon, L., Werner, S., Perez, D., Knottenbelt, W.J.: Defi protocols for loanable funds: Interest rates, liquidity and market efficiency. In: ACM Conference on Advances in Financial Technologies. pp. 92–112 (2020). https://doi.org/10.1145/3419614.3423254
54. Kaihua Qin and Liyi Zhou and Arthur Gervais: Quantifying Blockchain Extractable Value: How dark is the forest? arXiv preprint arXiv:2101.05511 (2021), https://arxiv.org/abs/2101.05511
55. Kao, H.T., Chitra, T., Chiang, R., Morrow, J.: An Analysis of the Market Risk to Participants in the Compound Protocol https://scfab.github.io/2020/FAB2020_p5.pdf
56. Massimo Bartoletti and James Hsin-yu Chiang and Alberto Lluch-Lafuente: SoK: Lending Pools in Decentralized Finance. arXiv preprint arXiv:2012.13230 (2020), https://arxiv.org/abs/2012.13230

57. Massimo Bartoletti and James Hsin-yu Chiang and Alberto Lluch-Lafuente: A theory of Automated Market Makers in DeFi. arXiv preprint arXiv:2102.11350 (2021), https://arxiv.org/abs/2102.11350
58. Moin, A., Sekniqi, K., Sirer, E.G.: Sok: A classification framework for stablecoin designs. In: Financial Cryptography and Data Security. LNCS, vol. 12059, pp. 174–197. Springer (2020). https://doi.org/10.1007/978-3-030-51280-4_11
59. Perez, D., Werner, S.M., Xu, J., Livshits, B.: Liquidations: Defi on a knife-edge. In: Financial Cryptography (2021), (to appear) https://arxiv.org/abs/2009.13235
60. Qin, K., Zhou, L., Livshits, B., Gervais: Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit. In: Financial Cryptography (2021), (to appear) https://arxiv.org/abs/2003.03810
61. Seijas, P.L., Thompson, S.J.: Marlowe: Financial contracts on blockchain. In: ISoLA. LNCS, vol. 11247, pp. 356–375. Springer (2018). https://doi.org/10.1007/978-3-030-03427-6_27
62. Tolmach, P., Li, Y., Lin, S.W., Liu, Y.: Formal Analysis of Composable DeFi Protocols. In: 1st Workshop on Decentralized Finance (2021), (to appear) https://arxiv.org/abs/2103.00540
63. Vandin, Andrea and Giachini, Daniele and Lamperti, Francesco and Chiaromonte, Francesca: Automated and Distributed Statistical Analysis of Economic Agent-Based Models. arXiv preprint arXiv:2102.05405 (2021), https://arxiv.org/abs/2102.05405
64. Wang, D., Wu, S., Lin, Z., Wu, L., Yuan, X., Zhou, Y., Wang, H., Ren, K.: Towards understanding flash loan and its applications in defi ecosystem. arXiv preprint arXiv:2010.12252 (2020), https://arxiv.org/abs/2010.12252
65. Wang, Y.: Automated market makers for decentralized finance (defi). arXiv preprint arXiv:2009.01676 (2020), https://arxiv.org/abs/2009.01676
66. Werner, S.M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., Knottenbelt, W.J.: Sok: Decentralized Finance (DeFi). arXiv preprint arXiv:2101.08778 (2021), https://arxiv.org/abs/2101.08778
67. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-Frequency Trading on Decentralized On-Chain Exchanges. arXiv preprint arXiv:2009.14021 (2020), https://arxiv.org/abs/2009.14021