SoK: Lending Pools in Decentralized Finance

No Author Given

No Institute Given

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, such as the correct handling of funds, and to precisely describe vulnerabilities and attacks. We also discuss the role of lending pools in the broader context of decentralized finance.

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 includes mechanisms to incentivize honest behaviour (e.g., loans are eventually repaid), economic growth and stability. Existing lending pool platforms are already handling large volumes of crypto-assets: as of writing, the two main platforms currently hold $1.7B [17] and $1.4B [5] 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, 35], 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 [19]. Indeed, most current literature in DeFi is devoted to study the economic impact of these incentive mechanisms [39, 40, 46–48, 50].

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 [46, 47, 50], 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 the interactions between a lending pool and its 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 throughout inspection of the implementations of the two main lending pool platforms, Compound [16] and Aave [8], 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 price, 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 automatic market makers (Section 6).

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 [16] and Aave [8]; 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 7. 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 automated analysis techniques, which may include mechanised proofs of contract properties and agent-based simulations of lending pools and other DeFi contracts.

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 LP terms is in Table 1.

Users can lend assets to a 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 that 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 a 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 τ is the amount of units of a given native cryptocurrency (or fiat currency) needed to buy one unit of τ.
Exchange rate	Given two token types τ and τ', the ratio τ/τ' at which a user can exchange units of token type τ' for units of τ in a blockchain interaction.
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 τ. The borrower is required to hold *collateral* of another token τ' 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, operational model of lending pools. We do this incrementally, starting from a basic model of blockchains, on top of which we will specify the behaviour lending pools.

3.1 A basic model of blockchains

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 $\tau, \tau', \ldots$. We denote with $\mathbb{T}_f \subseteq \mathbb{T}$ the subset of tokens types that can be freely transferred between users, only assuming a sufficient balance of the sender ($\mathbb{T}_f$ includes e.g. the native blockchain tokens).

We render blockchain states as partial maps $\sigma \in \mathbb{A} \rightharpoonup (\mathbb{T} \rightharpoonup \mathbb{Q}^+)$, where $\sigma\mathsf{A}$ represents A's token balance (a partial map from token types to nonnegative rational numbers). Hereafter, we abbreviate $\sigma\mathsf{A}$ as σ_A. We use the standard notation $f\{v/x\}$ to update a partial map f at point x: namely, $f\{v/x\}(x) = v$, while $f\{v/x\}(y) = f(y)$ for $y \neq x$.

Given a partial map $f \in \mathbb{T} \rightharpoonup \mathbb{Q}^+$, a token type $\tau \in \mathbb{T}$ and a partial binary operation $\circ \in \mathbb{Q}^+ \times \mathbb{Q}^+ \rightharpoonup \mathbb{Q}^+$, we define the partial map $f \circ v : \tau$ as follows:

$$f \circ v : \tau = \begin{cases} f\{f(\tau) \circ v/\tau\} & \text{if } \tau \in \operatorname{dom} f \text{ and } f(\tau) \circ v \text{ is defined} \\ f\{v/\tau\} & \text{if } \tau \notin \operatorname{dom} f \end{cases} \tag{1}$$

We adopt the notation $v : \tau$ to denote v units of token τ throughout the paper.

We model the interaction between users and the blockchain as a state transition system, with labels ℓ which represent transactions. Our basic model has only one kind of transaction, $\mathsf{Trf}_\mathsf{A}(\mathsf{B}, v : \tau)$, which represents the transfer of $v : \tau$ from A to B. Its effect on the state is specified by the following rule:

$$\frac{① \ \sigma_\mathsf{A}(\tau) \geq v \quad ② \ \tau \in \mathbb{T}_f \quad ③ \ \sigma'_\mathsf{A} = \sigma_\mathsf{A} - v : \tau \quad ④ \ \sigma'_\mathsf{B} = \sigma_\mathsf{B} + v : \tau}{\sigma \xrightarrow{\mathsf{Trf}_\mathsf{A}(\mathsf{B}, v:\tau)} \sigma\{\sigma'_\mathsf{A}/\mathsf{A}\}\{\sigma'_\mathsf{B}/\mathsf{B}\}} \ [\textsc{Trf}]$$

We decorate rule preconditions with circled numbers, e.g. ①, to simplify their reference in the text. Rule [Trf] states that the transfer is permitted whenever the sender has a sufficient balance ①, and the transferred token type is free ②.

3.2 Lending pool states

We now extend our basic blockchain model with lending pools, focussing on the common features implemented by the main 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 (see Section 7 for a discussion on the differences between our model and the existing platforms).

We model states Γ as terms of the form $\sigma \mid \pi \mid p$, where σ is the token balance of users, π is the lending pool state, and $p \in \mathbb{T}_f \to \mathbb{Q}^+$ models an oracle who prices the free tokens. Lending pool states π are triples (π_f, π_l, π_m), where:

$\mathsf{Dep}_{\mathsf{A}}(v : \tau)$	A deposits v units of a free token τ, receiving minted tokens
$\mathsf{Bor}_{\mathsf{A}}(v : \tau)$	A borrows v units of free token τ
Int	All loans accrue interest
$\mathsf{Rep}_{\mathsf{A}}(v : \tau)$	A repays v units on A's loan in τ
$\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$	A redeems v units of minted τ, receives deposited tokens
$\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$	A repays v units of B's loan in τ, seizing $v' : \tau'$ from B
$\mathsf{Mtrf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$	A transfers v units of minted τ to B
$\mathsf{Trf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$	A transfers v units of free τ to B

Table 2: Lending pool actions.

Actions	σ_{A}				σ_{B}			π_f		π_l B	π_m		p	
	τ_0	τ_1	τ'_0	τ'_1	τ_0	τ_1	τ'_1	τ_0	τ_1	τ_0	τ_0	τ_1	τ_0	τ_1
0. Initial State	100	–	–	–	–	50	–	–	–	–	–	–	1	1
1. $\mathsf{Dep}_{\mathsf{A}}(50 : \tau_0)$	**50**	–	**50**	–	–	50	–	50	–	–	τ'_0**:50**	–	1	1
2. $\mathsf{Dep}_{\mathsf{B}}(50 : \tau_1)$	50	–	50	–	–	0	**50**	50	50	–	τ'_0:50	τ'_1**:50**	1	1
3. $\mathsf{Bor}_{\mathsf{B}}(30 : \tau_0)$	50	–	50	–	**30**	0	50	20	50	**30**	τ'_0:50	τ'_1:50	1	1
4. Int	50	–	50	–	30	0	50	20	50	**34**	τ'_0:50	τ'_1:50	1	1
5. $\mathsf{Rep}_{\mathsf{B}}(5 : \tau_0)$	50	–	50	–	**25**	0	50	**25**	50	**29**	τ'_0:50	τ'_1:50	1	1
6. Px	50	–	50	–	25	0	50	25	50	29	τ'_0:50	τ'_1:50	**1.3**	1
7. $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, 13 : \tau_0, 19 : \tau'_1)$	**37**	–	50	**19**	25	0	**31**	**38**	50	**16**	τ'_0:50	τ'_1:50	1.3	1
8. $\mathsf{Rdm}_{\mathsf{A}}(10 : \tau'_0)$	**48**	–	**40**	19	25	0	31	**27**	50	16	τ'_0**:40**	τ'_1:50	1.3	1

Table 3: Interactions between two users and a lending pool.

- $\pi_f \in \mathbb{T}_{\mathsf{f}} \rightharpoonup \mathbb{Q}^+$ records the balance of free token types deposited in the LP;
- $\pi_l \in \mathbb{A} \rightharpoonup (\mathbb{T}_{\mathsf{f}} \rightharpoonup \mathbb{Q}^+)$ records the amount and type of tokens lent to users;
- $\pi_m \in \mathbb{T}_{\mathsf{f}} \rightharpoonup ((\mathbb{T} \setminus \mathbb{T}_{\mathsf{f}}) \times \mathbb{Q}^+)$ records the amount of tokens minted by the LP upon deposits. Namely, $\pi_m(\tau) = (\tau', n)$ means that n units of a token type τ', minted by the LP to represent claims of deposited tokens of type τ, are currently held by users. We require that different free tokens are associated to different minted tokens:

$$\pi_m(\tau_1) = (\tau', n_1) \ \wedge \ \pi_m(\tau_2) = (\tau', n_2) \implies \tau_1 = \tau_2 \tag{2}$$

We denote with $\mathbb{T}_\pi$ the set of tokens minted by a LP in state π. For a minted token $\tau' \in \mathbb{T}_\pi$, we denote with $u_\pi(\tau')$ the *underlying* free token. Formally:

$$\mathbb{T}_\pi = \{fst(\pi_m(\tau)) \mid \tau \in \mathbb{T}_{\mathsf{f}}\} \qquad u_\pi(\tau') = \tau \quad \text{if } fst(\pi_m(\tau)) = \tau' \tag{3}$$

Note that (2) ensures that $u_\pi(\tau'_1) \neq u_\pi(\tau'_2)$ when $\tau'_1 \neq \tau'_2$. We say that a state $\sigma \mid \pi \mid p$ is *initial* if π_f, π_l, π_m have empty domain, and $\operatorname{dom} \sigma_{\mathsf{A}} \subseteq \mathbb{T}_{\mathsf{f}}$ for all A.

3.3 An overview of lending pools behaviour

Lending pools support several actions, summarized in Table 2. Before formalizing their behaviour, we give some intuition through an example involving two users A and B (see Table 3). A and B start by depositing 50 units of free tokens τ_0 and τ_1, for which they receive equal amounts of freshly minted tokens τ'_0 and τ'_1.

Next, B borrows $30 : \tau_0$. Here, the 50 minted tokens of type τ_1' in B's balance serve as *collateral* for the loan. The *collateralization* of B is the ratio between the *value* of B's balance of τ_1' and the value of B's loan of τ_0 (the value of a token balance is the product between the number of units of the token and its price). Assuming a minimum collateralization threshold of $C_{min} = 1.5$ and equal token prices for τ_0 and τ_1, B could borrow up to 34 units of τ_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 action 4, interest accrues on the loan made by B. Here, the interest rate is 12%, so B's loan amount grows from 30 to 34 units of τ_1. In action 5, B repays 5 units of τ_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 action 6, such that $p(\tau_0)$ increases 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 action 7, A liquidates $13 : \tau_0$ of B's loan, 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 the user performing it.

In action 8, A then *redeems* $10 : \tau_0'$, receiving $11 : \tau_0$ in exchange. Here, each unit of τ_0' is now exchanged for more than 1 unit of τ_0, due to accrued interest.

3.4 Lending pool transitions

We now present the full set of rules which formalize the behaviour of lending pools. To illustrate them, we provide an extended running example (Tables 4–9).

Deposit A user A can deposit v units of a token τ by performing the transaction $\mathsf{Dep}_\mathsf{A}(v : \tau)$, provided that the balance is sufficient ①. In return, A receives v' units of a token τ' minted by the LP. Upon the first deposit of τ, the LP creates a fresh (non-free) token type τ' ②; freshness ensures that condition (2) is preserved by the new state. For further deposits of τ, the LP mints new units of τ'. In both cases, the amount of minted units of τ' are recorded in the π_m component of the state ⑤. Note that premises ② and ⑤ require that τ must be a free token type. The amount v' ③ is the ratio between the deposited amount v and the exchange rate $ER_\pi(\tau)$ between τ and τ', defined in (4).

$$\frac{\textcircled{1}\ \sigma_\mathsf{A}(\tau) \geq v \qquad \textcircled{2}\ \tau' := \begin{cases} \text{fresh} \notin \mathbb{T}_\mathsf{f} & \text{if } \tau \notin \operatorname{dom} \pi_m \\ \pi_m(\tau) & \text{otherwise} \end{cases} \qquad \textcircled{3}\ v' := {}^{v}/_{ER_\pi(\tau)} \\ \textcircled{4}\ \pi_f' := \pi_f + v : \tau \qquad \textcircled{5}\ \pi_m' := \begin{cases} \pi_m\{{}^{(\tau', v')}/_{\tau}\} & \text{if } \tau \notin \operatorname{dom} \pi_m \\ \pi_m\{{}^{(\tau', v'' + v')}/_{\tau}\} & \text{if } \pi_m(\tau) = (\tau', v'') \end{cases}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Dep}_\mathsf{A}(v:\tau)} \sigma\{{}^{\sigma_\mathsf{A} - v:\tau + v':\tau'}/_{\mathsf{A}}\} \mid (\pi_f', \pi_l, \pi_m') \mid p} \ [\textsc{Dep}]$$

The main idea of the exchange rate 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. For a free token τ, the exchange

Table 4: Running example: deposit actions

Actions	σ_{A}				σ_{B}				σ_{C}		π_f			π_m		
	τ_0	τ_1	τ_0'	τ_1	τ_0	τ_2	τ_0'	τ_2'	τ_2	τ_2'	τ_0	τ_1	τ_2	τ_0	τ_1	τ_2
0. Initial state	100	300	-	-	50	50	-	-	100	-	-	-	-	-	-	-
1. $\mathsf{Dep}_{\mathsf{A}}(100:\tau_0)$	**0**	300	**100**	-	50	50	-	-	100	-	**100**	-	-	τ_0'**:100**	-	-
2. $\mathsf{Dep}_{\mathsf{A}}(150:\tau_1)$	0	**150**	100	**150**	50	50	-	-	100	-	100	**150**	-	τ_0':100	τ_1'**:150**	-
3. $\mathsf{Dep}_{\mathsf{B}}(50:\tau_0)$	0	150	100	150	**0**	50	**50**	-	100	-	**150**	150	-	τ_0'**:150**	τ_1':150	-
4. $\mathsf{Dep}_{\mathsf{B}}(50:\tau_2)$	0	150	100	150	0	**0**	50	**50**	**100**	-	150	150	**50**	τ_0':150	τ_1':150	τ_2'**:50**
5. $\mathsf{Dep}_{\mathsf{C}}(100:\tau_2)$	0	150	100	150	0	0	50	50	**0**	100	150	150	**150**	τ_0':150	τ_1':150	τ_2'**:50**

rate $ER_\pi(\tau)$ represents the share of deposited units of τ over the units of the associated minted tokens. If any loans remain pending, not all minted tokens can be redeemed, as only a fraction of the deposited free tokens remain in the LP balance. Formally:

$$ER_\pi(\tau) = \frac{\pi_f(\tau) + \sum_{\mathsf{A}} (\pi_l\, \mathsf{A})\, \tau}{snd(\pi_m(\tau))} \;\; if\ \pi_f(\tau) > 0 \qquad ER_\pi(\tau) = 1 \;\; if\ \pi_f(\tau) = 0 \quad (4)$$

where we assume that the items A for which $\pi_l\, \mathsf{A}$ or $(\pi_l\, \mathsf{A})\tau$ are undefined do not contribute to the summation (we will adopt this convention through the paper).

Table 4 exemplifies users depositing funds to the LP. In transaction 1, A deposits 100 units of τ_0. Since this is the first deposit of τ, the LP mints exactly 100 units of a *fresh* token type, say τ_0', and transfers these units to A. In transaction 2, A deposits 150 units of τ_1; similarly to the previous case, A receives 150 units of a fresh token type τ_1'. In transaction 3, B deposits 50 units of τ_0. Since τ_0 was already deposited, the LP mints 50 units of the existing token type τ_0', and transfers them to B. Finally, in transactions 4 and 5 B and C deposit units of τ_2; after that, the balances of tokens τ_0, τ_1, τ_2 in the LP total 150 units.

Borrow Any user can borrow units of a free token type τ from the LP, provided that the LP has a sufficient balance of τ ①, and that the user has enough minted tokens to use as collateral ④. More specifically, we require that the *collateralization* of the user is above a constant threshold $C_{min} > 1$.

$$\frac{\text{① } \pi_f(\tau) \geq v > 0 \qquad \text{② } f_{\mathsf{A}} = \begin{cases} \pi_l \mathsf{A} + v:\tau & \text{if } \mathsf{A} \in \mathrm{dom}\, \pi_l \\ \{v/\tau\} & \text{otherwise} \end{cases} \qquad \text{③ } \pi' := (\pi_f - v:\tau,\ \pi_l\{f_{\mathsf{A}}/\mathsf{A}\},\ \pi_m) \qquad \text{④ } C_{\sigma|\pi'|p}(\mathsf{A}) \geq C_{min}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Bor}_{\mathsf{A}}(v:\tau)} \sigma\{\sigma_{\mathsf{A}} + v:\tau/\mathsf{A}\} \mid \pi' \mid p} \;\; [\textsc{Bor}]$$

To define the collateralization of users, we introduce a few auxiliary notions. The value $V^l(\mathsf{A})$ of A's loans is the sum (over all free token types τ) of the *value* of τ-tokens lent to A (the value is the product between token amount and price). For instance, if A has borrowed only 10 tokens of type τ, and the price of $1:\tau$ is $2:\tau_n$, then the value of A's loan is $20:\tau_n$. Formally:

$$V^l_\Gamma(\mathsf{A}) \;=\; \textstyle\sum_{\tau \in \mathbb{T}_{\mathsf{f}}} (\pi_l\, \mathsf{A})\tau \cdot p(\tau) \qquad \text{if } \Gamma = \sigma \mid \pi \mid p \quad (5)$$

Table 5: Running example: borrow actions

Actions	σ_{B}					σ_{C}				π_l			π_f			p			C_Γ	
										B		C								
	τ_0	τ_1	τ_2	τ_0'	τ_1'	τ_0	τ_1	τ_2	τ_2'	τ_1	τ_0	τ_1	τ_0	τ_1	τ_2	τ_0	τ_1	τ_2	B	C
5. $\mathsf{Dep}_{\mathsf{C}}(100:\tau_2)$	0	-	0	50	50	-	-	0	100	-	-	-	150	150	**150**	1	1	1	-	-
6. $\mathsf{Bor}_{\mathsf{B}}(50:\tau_1)$	0	**50**	0	50	50	-	-	0	100	**50**	-	-	150	**100**	100	1	1	1	**2.0**	-
7. $\mathsf{Bor}_{\mathsf{C}}(30:\tau_0)$	0	50	0	50	50	**30**	-	0	100	50	**30**	-	**120**	100	100	1	1	1	2.0	**3.3**
8. $\mathsf{Bor}_{\mathsf{C}}(30:\tau_1)$	0	50	0	50	50	30	**30**	0	100	50	30	**30**	120	**70**	70	1	1	1	2.0	**1.7**

The value $V^m(\mathsf{A})$ of minted tokens held by A is the summation (over all minted token types τ) of the *value* of A's balance of minted tokens. To determine the value of an minted token τ', its price is equated to that of the underlying free token τ, as minted tokens do not exist in the domain of p:

$$V^m_\Gamma(\mathsf{A}) \;=\; \textstyle\sum_{\tau\in\mathbb{T}\setminus\mathbb{T}_{\mathsf{f}}} \sigma_{\mathsf{A}}(\tau)\cdot ER_\pi(u_\pi(\tau))\cdot p(u_\pi(\tau)) \qquad \text{if } \Gamma=\sigma\mid\pi\mid p \tag{6}$$

The collateralization of a user is the ratio of the value of minted to lent tokens:

$$C_\Gamma(\mathsf{A}) \;=\; V^m_\Gamma(\mathsf{A}) \;/\; V^l_\Gamma(\mathsf{A}) \tag{7}$$

We exemplify Bor transactions in Table 5. Users B and C borrow amounts of τ_0 and τ_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 $V^l(\mathsf{C})$, whilst $V^m(\mathsf{C})$ remains constant at 100.

As we have seen, user collateralization depends on the amount of minted tokens he possesses, the amount of tokens, and the price of all tokens involved. Therefore, 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). Borrowers must therefore maintain a safety margin in order to protect against potential liquidation.

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 $I_\pi(\tau)$ to each loan, updating the π_l mapping for *all* users.

$$\frac{\pi_l'(\mathsf{A}) := f'_{\mathsf{A}} \;\text{ if } \mathsf{A}\in\operatorname{dom}\pi_l, \text{ where } \; f'_{\mathsf{A}}(\tau) := (I_\pi(\tau)+1)\cdot(\pi_l\,\mathsf{A})\tau \;\text{ if } \tau\in\operatorname{dom}(\pi_l\mathsf{A})}{\sigma\mid\pi\mid p \;\xrightarrow{\mathsf{Int}}\; \sigma\mid(\pi_f,\pi_l',\pi_m)\mid p}\;[\textsc{Int}]$$

Existing lending pool platforms deploy different algorithmic interest rate models [47]. 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 [47] satisfy:

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

We extend our running example with three interest updates in Table 6, resulting in the increase of all loan amounts. Each subsequent execution of Int *decreases* the collateralization of users B and C, since the V^l of both borrowers *increases* as interest is applied (7).

Table 6: Running example: interest accrual

Actions	π_l			I_π			p			C_Γ	
	B	C									
	τ_1	τ_0	τ_1	τ_0	τ_1	τ_2	τ_0	τ_1	τ_2	B	C
8. $\mathsf{Bor}_\mathsf{C}(30:\tau_1)$	50	30	**30**	2.0%	5.3%	0%	1	1	1	2.00	**1.67**
9. Int	**53**	**31**	**32**	2.1%	5.5%	0%	1	1	1	**1.89**	**1.59**
10. Int	**56**	**32**	**34**	2.1%	5.6%	0%	1	1	1	**1.79**	**1.52**
11. Int	**59**	**33**	**36**	2.2%	5.8%	0%	1	1	1	**1.69**	**1.45**

Table 7: Running example: repay actions

Actions	σ_A				σ_B					σ_C				π_f				π_l			C_Γ	
																		B	C			
	τ_0	τ_1	τ_0'	τ_1'	τ_0	τ_1	τ_2'	τ_0'	τ_2'	τ_0	τ_1	τ_2	τ_2'	τ_0'	τ_1'	τ_2'		τ_1	τ_0	τ_1	B	C
11. Int	0	150	100	150	0	50	0	50	50	30	30	0	100	120	70	150		**59**	**33**	**36**	**1.7**	**1.5**
12. $\mathsf{Rep}_\mathsf{C}(15:\tau_0)$	0	150	100	150	0	50	0	50	50	**15**	30	0	100	**135**	70	150		59	**18**	36	1.7	**1.9**

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

$$\frac{① \ \sigma_\mathsf{A}(\tau) \geq v > 0 \qquad ② \ (\pi_l \, \mathsf{A}) \, \tau \geq v \qquad ③ \ \pi_l' = \pi_l\{^{\pi_l \mathsf{A} - v:\tau}/_\mathsf{A}\}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Rep}_\mathsf{A}(v:\tau)} \sigma\{^{\sigma_\mathsf{A} - v:\tau}/_\mathsf{A}\} \mid (\pi_f + v:\tau, \pi_l', \pi_m) \mid p} \ [\textsc{Rep}]$$

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

In Table 7, C is suffering from low collateralization after the last interest accrual in transaction 11. Here, $C_\Gamma(\mathsf{C})$ is equal to $C_{min} = 1.5$. The subsequent repayment of 15 units of τ_0 increases C's collateralization back to 1.9.

Redeem A user without any loans can redeem minted tokens τ ① for the underlying tokens if enough units of $u_\pi(\tau)$ remain in the LP ②. 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} ③. This constraint does not apply to users without loans, as minted tokens are not used as collateral.

$$\frac{\begin{array}{l} ① \ \sigma_\mathsf{A}(\tau) \geq v > 0 \qquad v' := v \cdot ER_\pi(u_\pi(\tau)) \qquad ② \ \pi_f(u_\pi(\tau)) \geq v' \\ ③ \ (\exists \tau'.(\pi_l \mathsf{A})\tau' > 0) \Rightarrow C_{\sigma'|\pi'|p}(\mathsf{A}) \geq C_{min} \qquad \sigma_\mathsf{A}' := \sigma_\mathsf{A} - v:\tau + v':u_\pi(\tau) \\ \pi_f' := \pi_f - v':u_\pi(\tau) \qquad \pi_m' := \pi_m\{^{(\tau, v''-v)}/_{u_\pi(\tau)}\} \text{ where } (\tau, v'') := \pi_m(u_\pi(\tau)) \end{array}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Rdm}_\mathsf{A}(v:\tau)} \sigma\{^{\sigma_\mathsf{A}'}/_\mathsf{A}\} \mid (\pi_f', \pi_l, \pi_m') \mid p} \ [\textsc{Rdm}]$$

We exemplify Rdm transactions in Table 8. From Table 7, B has a non-zero loan amount, hence he can only redeem $11:\tau_2'$ before his collateralization decreases to $C_{min} = 1.5$, at which B cannot further redeem. Since A has no loans, she can redeem as many tokens τ_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 τ_2', and thus no interest was accrued. The tokens τ_2' and τ_0' returned to the LP by B and A are burnt and subtracted from π_m.

Table 8: Running example: redeem actions

Actions	σ_{A}				σ_{B}					π_f			π_m			C_Γ	
	τ_0	τ_1	τ'_0	τ'_1	τ_0	τ_1	τ_2	τ'_0	τ'_2	τ_0	τ_1	τ_2	τ_0	τ_1	τ_2	B	C
12. $\mathsf{Rep}_{\mathsf{C}}(15:\tau_0)$	0	150	100	150	0	50	0	50	50	**135**	70	150	τ'_0:150	τ'_1:150	τ'_2:150	1.7	**1.9**
13. $\mathsf{Rdm}_{\mathsf{B}}(11:\tau'_2)$	0	150	100	150	0	50	**11**	50	**39**	135	70	**139**	τ'_0:150	τ'_1:150	τ'_2**:139**	**1.5**	1.9
14. $\mathsf{Rdm}_{\mathsf{A}}(50:\tau'_0)$	**51**	150	**50**	150	0	50	11	50	39	**84**	70	139	τ'_0**:100**	τ'_1:150	τ'_2:139	1.5	1.9

Liquidation When the collateralization of a user B is below the threshold C_{min} ⑥, another user A can *liquidate* part of B's loan ⑧, in return for a discounted amount of minted tokens *seized* from B ⑩. A can execute Liq if it has enough balance to repay a fraction of the lent token ①, and if B has a sufficient balance of seizable, minted tokens ④. The maximum seizable amount is bounded by ④ or the resulting collateralization of B ⑦, which cannot exceed C_{min}. After this threshold, B's collateralization is restored, and B is no longer liquidatable.

$$\frac{\begin{array}{lll} ①\ \sigma_{\mathsf{A}}(\tau) \geq v & ②\ (\pi_l\,\mathsf{B})\,\tau \geq v & ③\ \tau' \in \mathbb{T}_\pi \\ ④\ \sigma_{\mathsf{B}}(\tau') \geq v' & ⑤\ v' = v \cdot \frac{p(\tau)}{p(u_\pi(\tau'))} \cdot r_{liq} & \\ ⑥\ C_{\sigma|\pi|p}(\mathsf{B}) < C_{min} & ⑦\ C_{\sigma'|\pi'|p}(\mathsf{B}) \leq C_{min} & \\ ⑧\ \pi'_l := \pi_l\,\mathsf{B} - v:\tau & ⑨\ \sigma'_{\mathsf{A}} := \sigma_{\mathsf{A}} - v:\tau + v':\tau' & ⑩\ \sigma'_{\mathsf{B}} := \sigma_{\mathsf{B}} - v':\tau' \end{array}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Liq}_{\mathsf{A}}(\mathsf{B},v:\tau,v':\tau')} \sigma\{\sigma'_{\mathsf{A}}/\mathsf{A}\}\{\sigma'_{\mathsf{B}}/\mathsf{B}\} \mid (\pi_f, \pi'_l, \pi_m) \mid p}\ [\textsc{Liq}]$$

For the execution of Liq, where $v:\tau$ and $v':\tau'$ are repaid and seized amounts respectively, the constraint on v and v' is given in ⑤, where:

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

The constraint $r_{liq} > 1$ implies a discount applied to the seized amount received by the liquidator, as more value is received than repaid:

For the liquidations in Table 9, we set $r_{liq} = 1.1$. After the price update in action 15, both B and C are undercollateralized. C is liquidated by A in transaction 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 units of both τ'_0 and τ'_2 until B's balance of minted tokens is empty. However, B still has a loan amount of $11:\tau_1$, which is *unrecoverable*. Both B and potential liquidators have no incentive to repay or liquidate given the lack of collateral.

Table 9: Running example: liquidation actions

Actions	σ_{A}					σ_{B}					σ_{C}				π_f			π_l			p		C_Γ	
																		B		C				
	τ_0	τ_1	τ'_0	τ'_1	τ'_2	τ_0	τ_1	τ_2	τ'_0	τ'_2	τ_0	τ_1	τ_2	τ'_0	τ_0	τ_1	τ_2	τ_1	τ_0	τ_1	τ_0, τ_1	τ_2	B	C
15. Px	51	150	50	150	-	0	50	11	50	39	15	30	0	100	84	70	138	59	18	36	1	**1.7**	**0.9**	**1.2**
16. $\mathsf{Liq}_{\mathsf{A}}(\mathsf{C}, 27:\tau_0, 50:\tau'_2)$	51	**123**	50	150	**50**	0	50	11	50	39	15	30	0	**50**	84	**97**	138	59	18	**9**	1	1.7	0.9	**1.5**
17. $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, 27:\tau_0, 50:\tau_0)$	51	**96**	**100**	150	50	0	50	11	**0**	39	15	30	0	50	84	**124**	138	**32**	18	9	1	1.7	**0.7**	1.5
18. $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, 21:\tau_0, 39:\tau'_2)$	51	**75**	100	150	**89**	0	50	11	0	**0**	15	30	0	50	84	**145**	138	**11**	18	9	1	1.7	**0**	1.5

Transfer of minted tokens Minted tokens can be transferred between users. Unlike free tokens transfers (rule [Trf] at page 4), this requires that the sender retains a collateralization level above C_{min}.

$$\frac{\sigma_{\mathsf{A}}(\tau) \geq v \qquad \tau \in \mathbb{T}_\pi \qquad \sigma' = \sigma\{\sigma_{\mathsf{A}} - v{:}\tau/\mathsf{A}\}\{\sigma_{\mathsf{B}} + v{:}\tau/\mathsf{B}\} \qquad C_{\sigma'|\pi|p}(\mathsf{A}) \geq C_{min}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Mtrf}_{\mathsf{A}}(\mathsf{B}, v:\tau)} \sigma' \mid \pi \mid p} \text{ [MTRF]}$$

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

$$\sigma \mid \pi \mid p \xrightarrow{\mathsf{Px}} \sigma \mid \pi \mid p' \quad \text{[PX]}$$

4 Fundamental properties of lending pools

We now establish some fundamental properties of lending pools. These properties hold for all *reachable states*, i.e. states Γ such that $\Gamma_0 \to^* \Gamma$ for some initial Γ_0.

The first property states that the component π_m of the state correctly records the balance of all minted tokens held by users. This is formalized by Lemma 1.

Lemma 1. *Let $\sigma \mid \pi \mid p$ be a reachable state. For all $\tau \in \mathbb{T}_\pi$:*

$$\textstyle\sum_{\mathsf{A}} \sigma_{\mathsf{A}}(\tau) \;=\; snd(\pi_m(u_\pi(\tau))) \tag{10}$$

Another crucial property is that the exchange rate of a minted token must either strictly increase, when users are borrowing the underlying token, or remain stable otherwise. This guarantees a depositor that her deposit will grow.

Lemma 2. *Let $\sigma \mid \pi \mid p$ be a reachable state, let $\sigma \mid \pi \mid p \xrightarrow{\ell} \sigma' \mid \pi' \mid p'$, and let $\tau \in \mathbb{T}_\pi$. Then: (a) if $\ell = \mathsf{Int}$ and $\exists \mathsf{A} : (\pi_l\, \mathsf{A})\, \tau > 0$, then $ER_\pi(\tau) < ER_{\pi'}(\tau)$; (b) otherwise, $ER_\pi(\tau) = ER_{\pi'}(\tau)$.*

As a direct consequence of Lemma 2 we have that, in any computation, the exchange rate of any token type is increasing.

We also establish a preservation property of the *supply* of any free token τ, i.e. the sum of all user balances and lending balance of τ:

$$sply_\Gamma(\tau) \;=\; \pi_f(\tau) + \textstyle\sum_{\mathsf{A}} \sigma_{\mathsf{A}}(\tau) \qquad \text{if } \Gamma = \sigma \mid \pi \mid p \tag{11}$$

Lemma 3 establishes that the supply of any free token is constant.

Lemma 3. *Let $\Gamma_0 \to^* \Gamma$, for Γ_0 initial. For all $\tau \in \mathbb{T}_{\mathsf{f}}$: $sply_{\Gamma_0}(\tau) = sply_\Gamma(\tau)$.*

While the supply of an LP remains constant, users act to increase their own share. We define the *net worth* $W_\Gamma(\mathsf{A})$ as the value of the amount of tokens in A's wallet or lent by A, minus the value of A's loans. Formally, if $\Gamma = \sigma \mid \pi \mid p$:

$$W_\Gamma(\mathsf{A}) = \textstyle\sum_{\tau \in \mathbb{T}_{\mathsf{f}}} \big(\sigma_{\mathsf{A}}(\tau) + \sigma_{\mathsf{A}}(fst(\pi_m(\tau)) \cdot ER_\pi(\tau) - (\pi_l\, \mathsf{A}\, \tau)\big) \cdot p(\tau)$$

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 that wants to increase her net worth, unless she can control price updates: actually, with just one price update the net worth of any user can be reduced to 0. However, under certain conditions, winning strategies can be found. 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. Here, liquidation is the only action by an honest user A that increases her net worth in just one transition.

Lemma 4. *Let Γ be a reachable state and $\Gamma \xrightarrow{\ell} \Gamma'$ with $\ell = \_{\mathsf{A}}(\cdots)$. Then: (a) $W_\Gamma(\mathsf{A}) > W_{\Gamma'}(\mathsf{A})$ if $\ell = \mathsf{Liq}_{\mathsf{A}}(\cdots)$; (b) $W_\Gamma(\mathsf{A}) = W_{\Gamma'}(\mathsf{A})$ otherwise.*

Since this is the winning strategy for all users, but liquidations may be limited by loan or collateral amounts, an adversary who has the power to drop or reorder transactions can potentially monopolize liquidations for itself. We refer to Section 6 for additional discussion of such attacks.

We now consider a slightly extended game, where A guesses that the next (adversarial) action is going to be ℓ, resulting in $\Gamma_0 \xrightarrow{\ell} \Gamma_1$ but can still perform an action ℓ' before ℓ, resulting in $\Gamma_0 \xrightarrow{\ell'} \Gamma'_0 \xrightarrow{\ell} \Gamma'_1$. The goal of A is to choose ℓ' such that $W_{\Gamma'_1}(\mathsf{A}) > W_{\Gamma_1}(\mathsf{A})$. We show that if $\ell = \mathsf{Int}$, i.e. A is expecting interest accrual to happen next, her choice is limited to deposits, repays and liquidations.

Lemma 5. *Let Γ_0 be a reachable state, and let $\Gamma_0 \xrightarrow{\ell} \Gamma_1$ and $\Gamma_0 \xrightarrow{\ell'} \Gamma'_0 \xrightarrow{\ell} \Gamma'_1$ be such that $\ell = \mathsf{Int}$ and $\ell' = \_{\mathsf{A}}(\cdots)$. Then: (a) $W_{\Gamma'_1}(\mathsf{A}) \geq W_{\Gamma_1}(\mathsf{A})$ if ℓ' is one of $\mathsf{Liq}_{\mathsf{A}}(\cdots)$ or $\mathsf{Dep}_{\mathsf{A}}(\cdots)$ or $\mathsf{Rep}_{\mathsf{A}}(\cdots)$; (b) $W_{\Gamma'_1}(\mathsf{A}) \leq W_{\Gamma_1}(\mathsf{A})$ otherwise.*

Overall, Lemmas 4 and 5 determine the set of actions to consider (together with their parameter) to maximize improvements in short-term net worth.

5 Lending pool safety, vulnerabilities and attacks

We discuss further properties of lending pools, focusing on potential risks which could lead to unsecured loans or exploitations by malicious actors. In particular, we focus on user collateralization and the availability of free token funds in lending pools (utilization). In the case where these can be targeted by an attacker, the motivation is to limit the lending pool functionality (denial-of-service) or cause the victim to incur losses, which in some cases may imply a gain for the attacker. We restrict our attention to attacker models where the attacker has 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 6.

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 should the borrowing users fail

to repay. However, collateral liquidation is exposed to risks. Firstly, 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. Secondly, large fluctuations in token price may reduce the relative value of the collateral such that the loan becomes partially unrecoverable. Furthermore, 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 The lending pool must determine the appropriate levels of collateralization based on token prices given by the price oracle. However, the value of LP-minted tokens is indeterminable since they are not featured in in $\mathrm{dom}\,(p)$. The definition of collateralization in eq. (7) values units of LP-minted tokens at the same price as their underlying counterpart, as do lending pool implementations [10, 21]. However, since LP-minted tokens represent claims on free tokens, which are only redeemable if sufficient funds remain in the lending pool (② in [RDM]), it is possible that users value minted tokens at a lower price than their underlying counterparts when LP-minted tokens cannot be redeemed during times of low lending pool funds (utilization). Lending pool designs do not account for this and thus run the risk of incorrectly pricing LP-minted tokens and collateral.

Safe collateralization Assuming a correct valuation of LP-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 [48], we say that a LP state is *ε-collateralization safe* when the ratio of the loan value of undercollateralized accounts to the total loan value of the lending pool is below the threshold ε:

$$\frac{\sum_{C_\Gamma(\mathsf{A})<C_{min}} V_\Gamma^l(\mathsf{A})}{\sum_{\mathsf{A}} V_\Gamma^l(\mathsf{A})} \leq \varepsilon \tag{12}$$

If the liquidation incentive is effective, a value below ε should not persist, as users are quick to execute liquidations. The efficiency of lending pool liquidations has been studied in [50]. We note that sufficiently large volumes of seized collateral which are immediately sold on external markets may delay further liquidations, as investigated in [46], due to the external market's finite capacity to absorb such a sell-off.

However, *ε-collateralization safety* does not account for undercollateralized loans which are *non-recoverable*, as previously illustrated in the example of Table 9. The set of non-recoverable, undercollateralized accounts are those with a collateralization below r_{liq}. The non-recoverable loan value of an account is given by V_Γ^{nrl}. It represents the remaining loan value of a user A should it be

fully liquidated, such that no further collateral can be seized.

$$V_\Gamma^{nrl}(\mathsf{A}) = \begin{cases} V_\Gamma^l(\mathsf{A}) - \frac{V_\Gamma^m(\mathsf{A})}{r_{liq}} & \textit{iff } C_\Gamma(\mathsf{A}) < r_{liq} \\ 0 & \textit{otherwise} \end{cases} \tag{13}$$

Equation (13) illustrates that for the case where an account collateralization is below r_{liq}, the discounted value of the collateral can no longer equal or exceed the remaining loan value, a consequence of (7) and (9). We say that a LP state is *strongly ε-collateralization safe* when the fraction of the total loan value of a lending pool which is not recoverable is below ε:

$$\frac{\sum_{\mathsf{A}} V_\Gamma^{nrl}(\mathsf{A})}{\sum_{\mathsf{A}} V_\Gamma^l(\mathsf{A})} \leq \varepsilon \tag{14}$$

The condition (14) is actually stronger than (12), i.e. if a state is strongly ε-collateralization safe, then it is also ε-collateralization safe. Given equal denominators of (12) and (14), this is a consequence of comparing numerators: Here, it can be observed that the numerator of (12) is greater than that of (14), as $V_\Gamma^l(\mathsf{A})$ is necessarily greater than $V_\Gamma^{nrl}(\mathsf{A})$ by definition and the set $\{\mathsf{A} \mid C_\Gamma(\mathsf{A}) < C_{min}\}$ is a superset of $\{\mathsf{A} \mid C_\Gamma(\mathsf{A}) < r_{liq}\}$ (13).

Strong price volatility is a risk to ε-collateralization safety, as a sharp drop in price can immediately reduce a previously sufficiently collateralized user to become undercollateralized below the threshold of C_{min}: such an immediate drop leaves the user with no opportunity maintain its collateral 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 ε-collateralization safe or strongly ε-collateralization safe.

For example, a malicious agent controlling the price oracle could act as follows. First, she would perform price updates to push any account collateralization below C_{min}, such that it becomes undercollateralized. The attacker can then perform liquidations on these accounts 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 $V_\Gamma^l(\mathsf{B})$ in (7) is zero, where B is an account under attack. In this case, $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$ can be performed with $v = 0$, and repeated liquidations can be executed to seize the full balance of B's LP-tokens.

As a matter of fact, a recent failure of the oracle price feed utilized by the Compound lending pool implementation lead to $100M of collateral being (incorrectly) liquidated [19]: though it is unclear whether this was an intentional exploit, it illustrates the feasibility of such a price oracle attack.

5.2 Utilization bounds and risks

The notion of *utilization* plays a fundamental role in the incentive model of lending pools as explained in [47]. As a matter of fact, it is often used as a key parameter of interest rate models in implementations [11, 22] and literature [47].

The utilization of a token type in a lending pool is the fraction of previously deposited funds currently lent to borrowing users. Formally:

$$U_\pi(\tau) \;=\; \frac{\sum_{\mathsf{A}} (\pi_l\, \mathsf{A})\, \tau}{\pi_f(\tau) + \sum_{\mathsf{A}} (\pi_l\, \mathsf{A})\, \tau} \tag{15}$$

Over- and under-utilization The value of $U_\pi(\tau)$ ranges between 0 and 1. We say that τ is *under-utilized* if its utilization is 0 and *over-utilized* when it is 1. We say that an LP state is under(over)-utilized if there is at least one under(over)-utilized token.

Under-utilization occurs when some units of τ have been deposited, but not lent to any user. This implies that action Int does not increase the loan value of any account, so that the exchange rate of τ in (4) remains constant, thereby not resulting in any gain for lenders.

On the other hand, over-utilization occurs when some users have borrowed τ, but the lending pool has no deposited funds of τ. In this case users can neither borrow or redeem.

Under- and over-utilization are not desirable and should be avoided. An optimal utilization rate eq. (15) of a free token type τ strikes a balance between the competing objectives of interest maximization and the ability for users to borrow or redeem tokens of type $\tau' = \mathit{fst}(\pi_m(\tau))$. In particular, the lending pool interest rate models described in [47] intend to incentivize actions of both borrowers and lenders to discover a utilization 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 proceed to discuss under- and overutilization attacks: here, we note that the former is weaker than the latter, as funds can still be safely recovered in a case of underutilization.

Under-utilization attacks Under-utilization can be achieved by a group of malicious users interested in reducing interest accrual for depositors or discouraging borrowing of a token τ. Here, the attacker can temporarily reduce utilization by repaying large amounts of loans, though the effectiveness of this approach will depend on the amounts of τ repaid by the attacker, as a lowered utilization can also reduce the interest rate (in certain models [47]), 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 could be achieved by a group of malicious users interested in preventing redeems or borrows of τ. The malicious users can do this by redeeming all units of τ while avoiding loans to be repaid or liquidated. We illustrate an over-utilization attack in Table 10. Here, users A and C initially hold the entire supply of τ_0 in their balances. A colludes with B to steal C's balance of τ_0: in actions 0-2, both A and B deposit units of $100 : \tau_0$

Actions	σ_{A}		σ_{A}			σ_{C}		π_f		π_l B	π_m		U_π	
	τ_0	τ_0'	τ_0	τ_1	τ_1'	τ_0	τ_0'	τ_0	τ_1	τ_0	τ_0	τ_1	τ_0	τ_1
0. Initial State	100	–	–	100	–	50	–	–	–	–	–	–	–	–
1. $\mathsf{Dep}_{\mathsf{A}}(100:\tau_0)$	**0**	**100**	–	100	–	50		**100**	–	–	τ_0'**:100**	–	**0**	–
2. $\mathsf{Dep}_{\mathsf{B}}(100:\tau_1)$	0	100	–	**0**	**100**	50		100	**100**	–	τ_0':100	τ_1'**:100**	0	**0**
3. $\mathsf{Bor}_{\mathsf{B}}(50:\tau_0)$	0	100	**50**	0	100	50		50	100	**50**	τ_0':100	τ_1':100	**0.5**	0
4. $\mathsf{Dep}_{\mathsf{C}}(50:\tau_0)$	0	100	50	0	100	**0**	**50**	100	100	50	τ_0'**:150**	τ_1':100	**0.3**	0
5. $\mathsf{Rdm}_{\mathsf{A}}(100:\tau_0')$	**100**	**0**	50	0	100	0	50	**0**	100	50	τ_0'**:50**	τ_1':100	**1.0**	0

Table 10: Over-utilization attack.

and $100:\tau_1$ respectively. B utilizes her balance of $100:\tau_1'$ as collateral to borrow $50:\tau_0$ from the lending pool in action 3. At this point, A and B are acting as lender and borrower of τ_0, for which the utilization is 0.5. C, having observed an opportunity to earn interest on τ_0 decides to deposit $50:\tau_0$ in action 4. However, user A still has a balance of redeemable $100:\tau_0'$, which she redeems in action 5. Now, users A and B have removed all units of τ_0 from the lending pool, pushing the utilization of τ_0 to 1 and preventing C from redeeming her funds. Of course, user B cannot redeem his balance of τ_1' since her loan has not been repaid, but this can be considered the cost of the attack.

6 DeFi archetypes: lending pools and beyond

We now discuss the interplay between lending pools and other DeFi applications, like algorithmic stable coins, automatic market makers, margin trading and flash loans, which are all predominantly deployed on the Ethereum blockchain [38].

Lending pools The emergent behaviour of lending pools in times of high price volatility is examined in [46] 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 ε-*collateralization safety* as we have defined in eqs. (12) and (14).

Lending pool behaviour at the user level is modelled in [48], which simulates agents interacting with the Compound implementation to examine the evolution of *liquidatable* and *undercollateralized* debt, notions similar to *(strong)* ε-*collateralization safety* (12) (14). [39,40] 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 [47], where empirical behaviour of interest rate models in Compound [22], Aave [11] and dYdX [25] 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 [47], our model can be instantiated with them.

Algorithmic stable coins MakerDAO [27] 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. 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 authors of [49] introduce a taxonomy for various price stabilization mechanisms, providing insight into the functionality of such contracts. [46] 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 [45], which also proposes alternate price feed aggregation models to improve oracle accuracy. Finally, [42] 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.

Automatic market makers Leading automatic market makers Uniswap [31] and Curve Finance [23] hold $1.6B [30] and $1.5B [23] worth of tokens and feature an estimated $320M [30] and $36M [23] worth of token exchange transactions every day. An automatic market maker (AMM) is organized in token pairs (τ, τ'), which users can interact with to exchange units of τ for τ' or vice-versa. AMM's do not match opposing actions of buyers and sellers: users simply exchange tokens with a AMM pair, where the exchange rate is determined algorithmically as a function of the AMM pair balance. Hence, the dynamic exchange rate of an AMM token pair is affected with each user interaction.

The work in [33] investigates alternative, algorithmic exchange rate models and defines the user arbitrage problem, where a profit-seeking agent must determine the optimal set of AMM pairs (with differing exchange rates) to interact with: given such arbitrage opportunities will be exploited by rational users, it is expected that exchange rates across AMM's remain consistent. AMM price models can fail: The *constant product* exchange rate model implemented by Uniswap [31] and Curve [24] is simple, but can theoretically reach a state where the the exchange rate is arbitrarily high. [54] proposes bounded exchange rate models to address this.

[32] suggests that AMM's track global average token prices effectively. As such, AMM's can inform price oracles: such oracles, however, only update price information with each new block [29] computed from time-weighted price averages of AMM pairs 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.

AMM's suffer from front-running, where an attacking user observes the victim's announced, yet 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 change in exchange rate resulting from the victim's token exchange, who ends up paying a higher price, as illustrated in [55]. Front-running of smart contracts is investigated more generally in [44]: mitigations such as commit-and-reveal schemes are proposed, which come with an increased cost for user-contract interactions. In the context of AMM's, [41] 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: [41] 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 τ against τ', the user speculates that the price of the former will increase against the price of the latter: a user borrows τ' at a lending pool against collateral deposited in τ, and then exchanges the borrowed units of τ' back to τ at a token exchange or an AMM. The user will now earn an amplified profit if the price of τ appreciates relative to τ', since both the borrowed balance and redeemable collateral in τ appreciates in value whilst only the loan repayable with τ' decreases in value. A leveraged short position simply reverses the token types. Margin trading contracts such as bZx Fulcrum [13] 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 AMM's, attackers can use such actions to manipulate AMM prices, as shown in [51]. Furthermore, the scope of such attacks is magnified when performed with flash loans.

Flash loans Any smart contract holding balances of tokens can expose flash loan functionality to users: here, a user can borrow and return a loan within a single atomic transaction group. Informally, we describe an atomic transaction group as an a sequence of actions from a single user, which must execute to completion or not execute at all. Atomic transaction groups can be implemented in Ethereum by user-defined smart contracts [7], but can also be supported explicitly, such as in Algorand [36]. As such, flash loans are guaranteed to be repaid or not executed at all. The work in [53] introduces an initial 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. We note that our model can be easily extended to encompass flash loan semantics.

Flash loans have been utilized in recent attacks in DeFi contracts [51] [14] [26] [28] [12]. The flashloan attack on bZx Fulcrum described in [51] 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.

7 Conclusions

We have provided a systematization of knowledge on lending pools and their role in DeFi, by leveraging a new model which enables formal definitions of lending pool properties, vulnerabilities, and attacks. This work represents a first step towards the rigorous analysis of DeFi contracts, improving existing literature with a precise executable semantics of interactions beween users and LPs.

Differences between our model and LP implementations We have synthesised our model from informal descriptions in the literature and actual implementation and documentation of lending pools Compound [22] and Aave [11]. To distill a usable, succinct model we have abstracted away 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 [15]. 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 [6,18]. 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 [9,20] in order to reduce the cost of execution, leading to inaccuracies in loan interest.

Comparison with other LP models There are few models of lending pools in the literature. The *liquidation model* of [46] is meant to simulate interactions between lending pool liquidations and token exchange markets in times of high price volatility. Unlike in our presented model, [46] performs liquidations in aggregate, and it omits individual user actions. The interest rate functions of [47] 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 $I_\pi(\tau)$ in rule [Int]. The work [50] introduces a 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.

Future work 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). This paves the way for future automatic analyses of LPs, which could exploit e.g., model checking or automated theorem proving tools. Following the same approach we used in Section 3, we could devise formal models of other DeFi archetypes, like e.g. stable coins, automatic market makers, and flash loans. In this perspective, it would be possible to extend the scope of analysis techniques to attacks that exploit the interplay between different archetypes, which so far have been found manually by adversaries, as documented in [51]. A complementary line of research is the design of domain-specific languages for DeFi contracts, in the spirit of the works [34,37,43,52] on languages for financial derivatives. By leveraging primitives specifically tailored to DeFi, these languages could simplify the task of analysing DeFi contracts: actually, this task is overwhelmingly complex for current LP implementations, which amount to thousands of lines of Solidity code.

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 markets website (2020), https://app.aave.com/markets
6. Aave maximum liquidation amount (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/lendingpool/LendingPoolLiquidationManager.sol#L181
7. Aave v1 flashloan receiver interface (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/flashloan/interfaces/IFlashLoanReceiver.sol#L11
8. Aave v1 implementation (2020), https://github.com/aave/aave-protocol/tree/efaeed363da70c64b5272bd4b8f468063ca5c361
9. Aave v1 simplified interest (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/libraries/CoreLibrary.sol#L423
10. Aave valuation of atokens (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/lendingpool/LendingPoolDataProvider.sol#L114
11. Aave website (2020), https://www.aave.com
12. Akropolis Defi attack (2020), https://cryptonews.com/news/defi-akropolis-drops-20-following-a-usd-2m-heavy-hack-8299.htm
13. bzx fulcrum website (2020), https://fulcrum.trade
14. Coindesk: Value DeFi attack (2020), https://www.coindesk.com/value-defi-suffers-6m-flash-loan-attack
15. Compound comptroller setter (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/CToken.sol#L1152
16. Compound implementation (2020), https://github.com/compound-finance/compound-protocol/tree/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6
17. Compound markets website (2020), https://compound.finance/markets
18. Compound maximum liquidation amount (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/ComptrollerG5.sol#L510
19. Compound oracle attack (2020), https://news.bitcoin.com/100-million-liquidated-on-defi-protocol-compound-following-oracle-exploit
20. Compound simplified interest (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/CToken.sol#L423
21. Compound valuation of ctokens (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/ComptrollerG5.sol#L753
22. Compound website (2020), https://www.compound.finance

23. Curve statistics (2020), https://www.curve.fi/dailystats
24. Curve website (2020), https://www.curve.fi
25. dydx website (2020), https://dydx.exchange
26. Harvest Finance flashloan attack post-mortem (2020), https://medium.com/harvest-finance/harvest-flashloan-economic-attack-post-mortem-3cf900d65217
27. Makerdao website (2020), https://https://makerdao.com
28. Origin Dollar attack (2020), https://cryptonews.com/news/4th-major-defi-hack-in-a-month-origin-dollar-loses-usd-7m-8331.htm
29. Uniswap oracle template (2020), https://github.com/Uniswap/uniswap-v2-periphery/blob/dda62473e2da448bc9cb8f4514dadda4aeede5f4/contracts/examples/ExampleOracleSimple.sol
30. Uniswap statistics (2020), https://info.uniswap.org
31. Uniswap website (2020), https://www.uniswap.org
32. Angeris, G., Chitra, T.: Improved price oracles: Constant function market makers. arXiv preprint arXiv:2003.10001 (2020), https://arxiv.org/pdf/2003.10001
33. 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
34. Arusoaie, A.: Certifying Findel derivatives for blockchain. CoRR **abs/2005.13602** (2020), https://arxiv.org/abs/2005.13602
35. 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
36. 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
37. Biryukov, A., Khovratovich, D., Tikhomirov, S.: Findel: Secure derivative contracts for Ethereum. In: Financial Cryptography Workshops. LNCS, vol. 10323, pp. 453–467. Springer (2017). https://doi.org/10.1007/978-3-319-70278-0_28
38. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)
39. Chitra, T.: Competitive equilibria between staking and on-chain lending. arXiv preprint arXiv:2001.00919 (2019), https://arxiv.org/pdf/2001.00919
40. Chitra, T., Evans, A.: Why stake when you can borrow? Available at SSRN 3629988 (2020), https://arxiv.org/pdf/2006.11156
41. 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
42. Darlin, M., Papadis, N., Tassiulas, L.: Optimal Bidding Strategy for Maker Auctions. arXiv preprint arXiv:2009.07086 (2020), https://arxiv.org/pdf/2009.07086
43. Egelund-Müller, B., Elsman, M., Henglein, F., Ross, O.: Automated execution of financial contracts on blockchains. Business & Information Systems Engineering **59**(6), 457–467 (2017). https://doi.org/10.1007/s12599-017-0507-z
44. 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

45. 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
46. 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
47. 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
48. 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
49. 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
50. 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
51. 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/pdf/2003.03810
52. 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
53. 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/pdf/2010.12252
54. Wang, Y.: Automated market makers for decentralized finance (defi). arXiv preprint arXiv:2009.01676 (2020), https://arxiv.org/pdf/2009.01676
55. 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/pdf/2009.14021

A Supplementary material

Proof of Lemma 1

The proof is by induction on the length of the trace from an initial state to the state $\sigma \mid \pi \mid p$. The base case is when $\sigma \mid \pi \mid p$ is an initial configuration. Then, (10) trivially holds since $\mathbb{T}_\pi$ is empty. Now assume as induction hypothesis that (10) holds for a reachable configuration Γ. We can show that the equation also holds for all configurations Γ' such that $\Gamma \xrightarrow{\ell} \Gamma'$ by considering all possible cases for ℓ. First note, that there are a number of cases where the state components involved in (10) are not affected at all. These are: $\mathsf{Trf}_\mathsf{A}(\mathsf{B}, v : \tau)$, Px, Int, $\mathsf{Bor}_\mathsf{A}(v : \tau)$, and $\mathsf{Rep}_\mathsf{A}(v : \tau)$. If ℓ is $\mathsf{Dep}_\mathsf{A}(v : \tau)$ or $\mathsf{Rdm}_\mathsf{A}(v : \tau)$ we note that the transition will increase and decrease both sides of (10) equally. Last, if ℓ is $\mathsf{Liq}_\mathsf{A}(\mathsf{B}, v : \tau, v' : \tau')$ or $\mathsf{Mtrf}_\mathsf{A}(\mathsf{B}, v : \tau)$ the sum in the lhs of (10) is kept constant (minted tokens are just transferred from one user to another one) and the rhs is not affected. □

Proof of Lemma 2

We show that for a single transition $\sigma \mid \pi \mid p \xrightarrow{\ell} \sigma' \mid \pi' \mid p'$ the following holds:

(a) $ER_\pi(\tau) < ER_{\pi'}(\tau)$, if $\ell = \mathsf{Int}$ and $\exists \mathsf{A} : (\pi_l\, \mathsf{A})\, \tau > 0$
(b) $ER_\pi(\tau) = ER_{\pi'}(\tau)$, otherwise.

Part (a) is easy to see, since the execution of Int strictly increases the existing loans on τ which are used in the numerator of the first case in (4), without affecting the denominator.

For part (b) there are a number of cases where the state components involved in ER are not affected at all. These are: Int (if $\nexists \mathsf{A} : (\pi_l\, \mathsf{A})\, \tau > 0$), $\mathsf{Trf}_\mathsf{A}(\mathsf{B}, v : \tau)$, Px, and $\mathsf{Mtrf}_\mathsf{A}(\mathsf{B}, v : \tau)$. If ℓ is $\mathsf{Bor}_\mathsf{A}(v : \tau)$, $\mathsf{Rep}_\mathsf{A}(v : \tau)$, or $\mathsf{Liq}_\mathsf{A}(\mathsf{B}, v : \tau_0, \tau_1)$ we note that the transition will increase and decrease the summands in the numerator of the first case in (4) equally. Last, If ℓ is $\mathsf{Dep}_\mathsf{A}(v : \tau)$ or $\mathsf{Rdm}_\mathsf{A}(v : \tau)$ we note that the transition will increase and decrease the numerator and denominator of the first case in (4) in quantities proportional to ER_π. □

Proof of Lemma 3

The proof is by induction on the length of the trace $\sigma \mid \pi \mid p \to^* \sigma' \mid \pi' \mid p'$. The base case is when $\sigma \mid \pi \mid p = \sigma' \mid \pi' \mid p'$. Then the lemma trivially holds since $sply_{\sigma,\pi}(\tau) = sply_{\sigma',\pi'}(\tau)$. Now, assume as induction hypothesis that the lemma holds for all executions of length n. We show that it also holds for executions of length $n + 1$. In particular we show that for a single transition $\sigma \mid \pi \mid p \xrightarrow{\ell} \sigma' \mid \pi' \mid p'$ token supplies remain constant by considering all possible cases for ℓ, where state components of Equation (11) are affected. These are: $\mathsf{Dep}_\mathsf{A}(v : \tau)$, $\mathsf{Bor}_\mathsf{A}(v : \tau)$, $\mathsf{Rdm}_\mathsf{A}(v : \tau)$ and $\mathsf{Liq}_\mathsf{A}(\mathsf{B}, v : \tau, v' : \tau')$. If ℓ is $\mathsf{Dep}_\mathsf{A}(v : \tau)$, $\mathsf{Bor}_\mathsf{A}(v : \tau)$ or $\mathsf{Liq}_\mathsf{A}(\mathsf{B}, v : \tau, v' : \tau')$, changes applied to $\sigma_\mathsf{A}(\tau)$ and $\pi_f(\tau)$ in Equation (11) cancel out. If ℓ is $\mathsf{Rdm}_\mathsf{A}(v : \tau)$, the same is true for user balance and lending pool balance of token $u_\pi(\tau)$. □

Proof of Lemma 4

By inspecting the formalization of the transitions it is easy to see which actions can increase or decrease in just one transition the net worth of a user on a specific token or in total, and to which extent. Indeed is is easy to see that the only actions that can modify the total net worth of a user A are Int, $\mathsf{Trf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$, $\mathsf{Mtrf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$, Px and $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$. Only the latter uses an action of the form $\_{\mathsf{A}}(\dots)$. The rest of the actions of the form $\_{\mathsf{A}}(\dots)$ are $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$, $\mathsf{Bor}_{\mathsf{A}}(v : \tau)$, $\mathsf{Rep}_{\mathsf{A}}(v : \tau)$, and $\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$. Inspecting their effect on wallets and loans we can notice that they simply exchange tokens in a way that keeps the net worth constant: $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$ and $\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$ simply swap amounts of free tokens and corresponding minted tokens proportionally to the exchange rate, while $\mathsf{Bor}_{\mathsf{A}}(v : \tau)$ and $\mathsf{Rep}_{\mathsf{A}}(v : \tau)$ simply move exact amounts of free tokens between wallets and loans. □

Proof of Lemma 5

The main idea is that interest accrual affects net worth by increasing the share of deposited tokens (on which there is at least one non-empty loan) and increasing loan amounts. The only actions that increase a user's deposits are $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$ and $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$, while the only action that decreases loans is $\mathsf{Rep}_{\mathsf{A}}(v : \tau)$. □